
29 PAGES OF EXPERT TUTORIALS
web designer
FREE 79 MINUTES OF JS VIDEOS
jQuery
JavaScript
@WebDesignerM
/WebDesignerUK
WEBGL & 3D
Build for the browser with ThreeJS
NEW RULES
UX
TOP TOOLS
Code a facial recognition app
5 hot new CSS properties
Add particle effects with particle.js
Learn to use the CSS Paint API
How to design and build for today's emerging experiences
SAY HELLO TO JSX
Discover the power of this JavaScript syntax extension
FUTURE
ISSUE 274

Welcome to the issue

THE WEB DESIGNER MISSION

To be the most accessible and inspiring voice for the industry, offering cutting-edge features and techniques vital to building future-proof online content

Steven Jenkins
Editor

Design for emerging experiences

As the web landscape changes so does the user experience and to stay competitive you need to embrace the new. One thing that doesn't change is the user. If they have a poor user experience they will simply look elsewhere. So what are the emerging experiences that you need to consider today and what are the tools needed to achieve the right results?

In our lead feature (p42) Laura Yarrow from Experience UX reveals the seven key themes that you should be designing for: Inclusivity and accessibility, immersion, trust and transparency, coherence, conversation, collaboration and efficiency. Alongside these key themes she reveals the tools that you will need to ensure design success. If you want to keep your users happy make sure you take a look.

WebGL and 3D have been part of the web experience for a while, but better support and better frameworks means that bigger and better experiences are just a few steps away. Richard Mattka gives you a rundown on how to use ThreeJS to build 3D for the browser today. Ever heard of JSX? This issue we give you an insight on how to build components faster with the help of the JavaScript syntax extension and the React library

As always, enjoy the issue,

> Inclusive design considers as many people's needs and abilities as possible, instead of a 'one size fits all' approach to the experience

Highlight

> Not only is R&D a great collaborative process, but the projects we create through it give us a unique edge

FX Digital are digital explorers crafting engaging and exciting experiences. **Web Designer** finds out more. **Page 34**

Follow us on Twitter for all the news & conversation @WebDesignerMag
Visit our blog for opinion, freebies & more www.creativebloq.com

This issue's panel of experts

Welcome to that bit of the mag where we learn more about the featured writers and contributors...

Laura Yarrow

Laura is a UX architect at Bournemouth-based agency Experience UX. With a background in web development and a love of new technology. This issue Laura shares the future themes UX practitioners will find themselves designing for, and the tools that will support their work. **Page 42**

> Over the next year, expect to see inclusive and ethical design become an expected part of the UX Design process

Simon Jones

Simon leads software engineering teams working with a variety of frontend technologies, including React. This issue, he covers the basics of how React, and associated technologies like JSX, can help you build web apps. **Page 74**

Mark Shufflebottom

Mark is a Professor of Interaction Design at Sheridan College and is also an Adobe Education Leader. In this tutorial, Mark is creating a facial recognition photo app that dynamically adds a variety of props onto users' faces. **Page 52**

Richard Mattka

Richard is an award-winning interactive technologist on experiences for Wonder Woman, Independence Day Resurgence, Google, and Sony. This issue he explores WebGL 3D with Three.js and reveals insider tips for beautiful results. **Page 66**

Frank Kagumba

Frank is a frontend developer dealing in CMS, HTML5, CSS3 and JavaScript. This issue he will teach you how to create a fullscreen landing page using HTML5 and CSS3 and animate it using the particle.js library. **Page 56**

Daniel Crisp

Dan is a senior frontend developer, currently wrangling CSS at Potato for a high-profile client. This issue he gives a practical guide to some exciting new CSS features that you can try out for yourself today. **Page 80**

Luke Harrison

Luke is a web developer from Sheffield, UK who is all about scalable and efficient frontend architecture. In this issue, he explores how to create a simple, usable modal component in React, leveraging its portal functionality. **Page 60**

Matt Crouch

Matt is a frontend developer at Vidsy in London. Now that we are starting to see some of the Houdini styling magic appear in users' browsers, Matt takes a look at the first piece to land in Chrome – the CSS Paint API. **Page 86**

Leon Brown

Leon is a freelance web developer and trainer who assists web developers in creating efficient code for projects. This issue he recreates a host of techniques as inspired by the top-class sites seen in Lightbox. **Page 14**

Follow us!

Facebook: *www.facebook.com/WebDesignerUK*
Twitter: *https://twitter.com/webdesignermag*

web designer

Future PLC Richmond House, 33 Richmond Hill, Bournemouth, Dorset, BH2 6EZ

Editorial
Editor **Steve Jenkins**
steve.jenkins@futurenet.com
01202 586233
Designer **Harriet Knight**
Editor in Chief **Amy Hennessey**
Senior Art Editor **Will Shum**

Contributors
Neal Thoms, Laura Yarrow, Simon Jones, Mark Shufflebottom, Frank Kagumba, Luke Harrison, Richard Mattka, Daniel Crisp, Matt Crouch, Leon Brown, Mark Billen, David Howell, Philip Morris, Rob Mead-Green, Ryan Wells

Photography
James Sheppard
All copyrights and trademarks are recognised and respected

Advertising
Media packs are available on request
Commercial Director **Clare Dove**
clare.dove@futurenet.com
Senior Advertising Manager **Mike Pyatt**
michael.pyatt@futurenet.com
01225 687538
Account Director **George Lucas**
george.lucas@futurenet.com
Account Manager **Chris Mitchell**
chris.mitchell@futurenet.com

International
Web Designer is available for licensing.
Contact the International department to discuss partnership opportunities
International Licensing Director **Matt Ellis**
matt.ellis@futurenet.com

Subscriptions
Email enquiries **contact@myfavouritemagazines.co.uk**
UK orderline & enquiries **0344 848 2852**
Overseas order line and enquiries **+44 (0) 344 848 2852**
Online orders & enquiries **www.myfavouritemagazines.co.uk**
Head of subscriptions **Sharon Todd**

Circulation
Head of Newstrade **Tim Mathers**

Production
Head of Production **Mark Constance**
Production Project Manager **Clare Scott**
Advertising Production Manager **Joanne Crosby**
Digital Editions Controller **Jason Hudson**
Production Manager **Nola Cokely**

Management
Managing Director **Aaron Asadi**
Commercial Finance Director **Dan Jotcham**
Editorial Director **Paul Newman**
Head of Art & Design **Greg Whittaker**

Printed by William Gibbons, 28 Planetary Road, Willenhall, WV13, 3XT

Distributed by Marketforce, 5 Churchill Place, Canary Wharf, London, E14 5HU www.marketforce.co.uk Tel: 0203 787 9060

ISSN 1745-3534

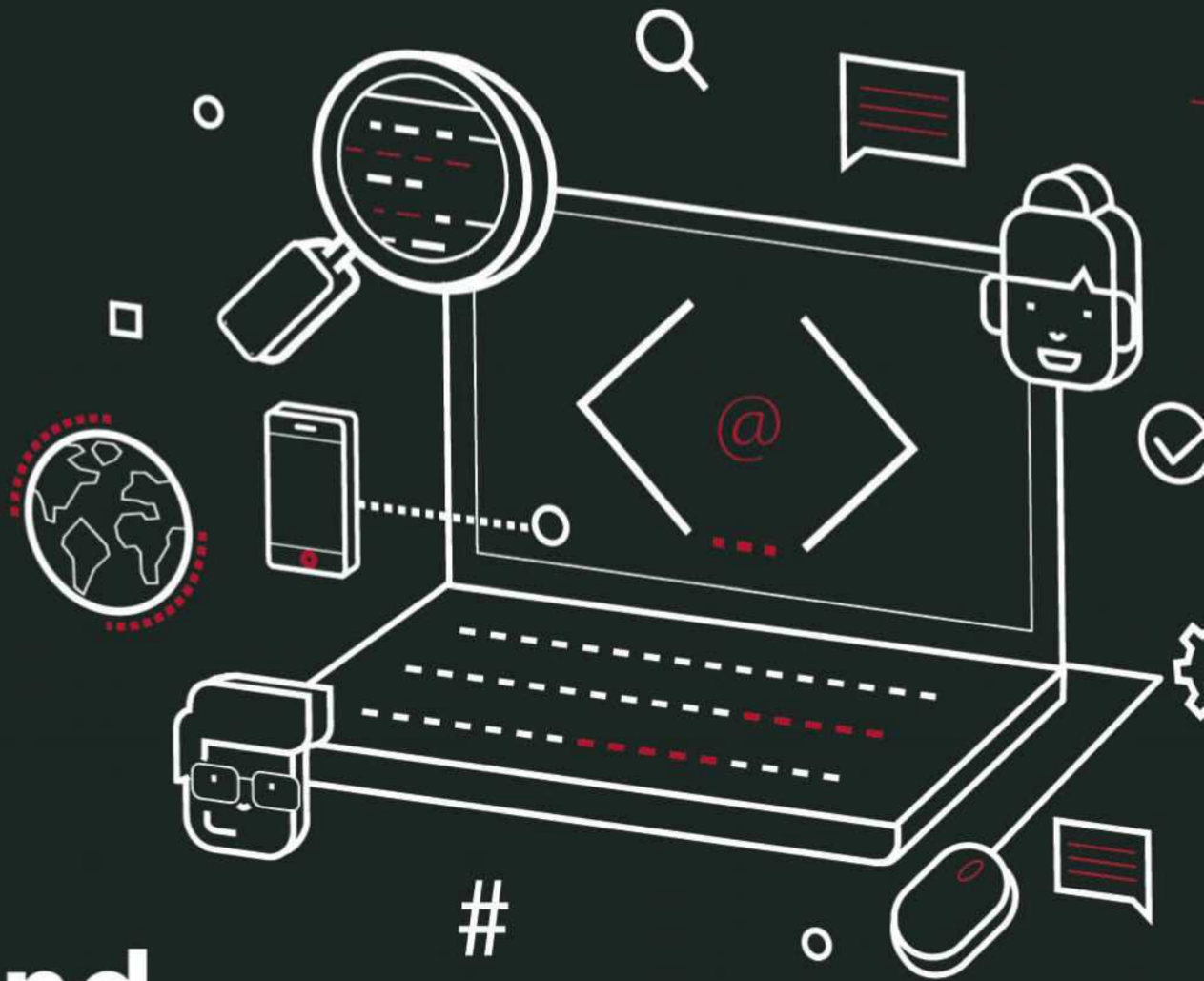
We are committed to only using magazine paper which is derived from responsibly managed, certified forestry and chlorine-free manufacture. The paper in this magazine was sourced and produced from sustainable managed forests, conforming to strict environmental and socioeconomic standards. The manufacturing paper mill holds full FSC (Forest Stewardship Council) certification and accreditation

FUTURE
Connectors. Creators. Experience Makers.

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

Chief executive **Zillah Byng-Thorne**
Chairman **Richard Huntingford**
Chief financial officer **Penny Ladkin-Brand**

Tel +44 (0)1225 442 244



<Land Your First Software Developer Job />

Far more than just
a **Coding Bootcamp**.

We'll secure you a career.

www.northcoders.com/web-designer

NORTH CODERS

Meta

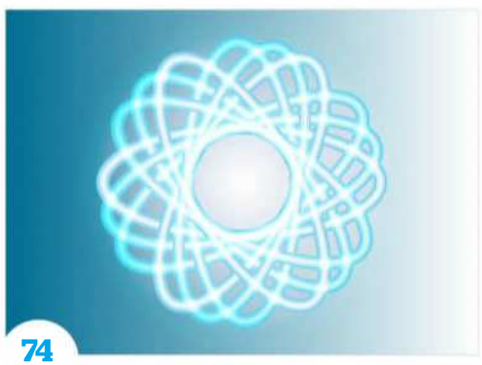
Inside issue 274

Contents

Cutting-edge features, techniques and inspiration for web creatives

Chat with the team and other readers and discuss the latest tech, trends and techniques. Here's how to stay in touch...
steve.jenkins@futurenet.com • @WebDesignerMag • www.creativebloq.com

66
WebGL 3D and ThreeJS
Create realistic 3D for the browser

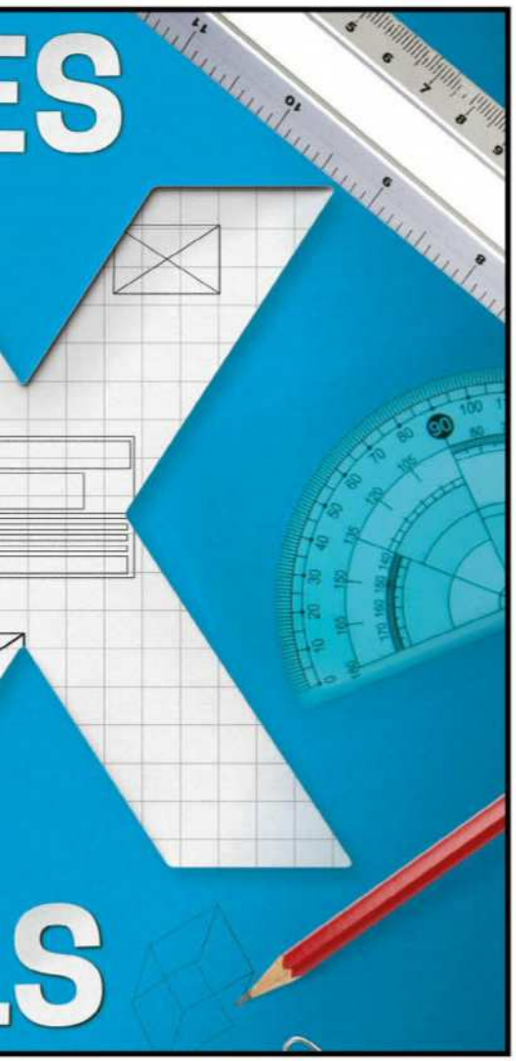


74
Say hello to JSX
How to build with React and the JS extension

FileSilo

96 Get the latest must-have resources and videos

- 79 minutes of Beginner's JavaScript video guides from StudioWeb
- 14 pink Photoshop filters
- 10 wooden picture frame mockups



> Shared hosting would seem to have a major advantage – cost

COMMENT – Neal Thoms – p11

34 **ProFile:** FX Digital
Creating exciting experiences

14 **Lightbox:** Guccy SS18
Adding a touch of style to design

Tutorials

Web gurus take you step-by-step through professional techniques

Web Workshop

Web Developer

CONTACT US AT: steve.jenkins@futurenet.com | @WebDesignerMag

Header

The tools, trends and news to inspire your web projects

Choosing the right framework

Selecting the right tool is essential for success. Web Designer takes a closer look at which JS framework you should be using

When it comes to frameworks there are a myriad of choices with new ones appearing all the time, adding confusion to which one to use. Indeed, frameworks are appearing at such a rate that designers and developers simply don't have time to learn the intricate workings of the newest addition.

One answer is to ignore the vast majority of frameworks and concentrate on what works best for your needs. This sounds simple, but choosing, learning and familiarising yourself with a major framework isn't as easy as it seems. It wasn't that long ago that jQuery was the JavaScript library that every budding developer had to learn. But fast forward to today and vanilla JavaScript and a few major frameworks are at the forefront of web development.

Whatever project you are working on, it is important that you choose the right tools. You or your company may sit in the Angular camp, but what if this isn't the best tool for the job, or there's a new framework that might be a better fit? For newcomers to web development, choosing the right one can be a daunting task. For seasoned developers, it might be time to learn something new. The big question is: how do you decide?

> *"Vanilla JavaScript and a few major frameworks are at the forefront of web development*

A recent Whitepaper from Progress, "Choosing the Right JavaScript Framework for Your Next Web Application", should help point you in the right direction. The paper chooses three major contenders in the shape of Angular, React and Vue.js, and states that its goal is "to help you match your current needs and context against the capabilities, strengths and limitations of the most popular JavaScript frameworks available today." The paper goes on to say: "By the end of this document, you should have the knowledge you need to confidently select a tool and get started building that next great application".

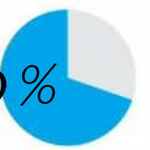
Like all good research, the paper reveals the criteria on which it is based. It looks at popularity, support, performance factors, ecosystem, beyond-the-browser options, IDE and tooling support, companion and CLI tools, licensing, security, and talent pool and resources. The last one being an interesting choice. It looks at how easy it is to hire developers (which indicates how well used a framework is) and how easy it is to learn. See the report at **bit.ly/2GA9AYN**.

STAT ATTACK

FACEBOOK

Have its recent indiscretions had an impact?

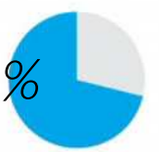
Worldwide



69.72%

Down 6% on the previous month

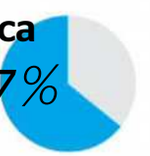
Europe



71.33%

Staying very much the same

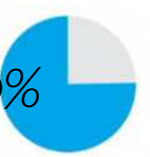
North America



64.27%

Down almost 14% on the previous month

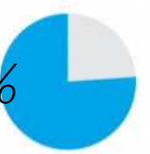
Africa



75.43%

Down 3% on the previous month

Asia

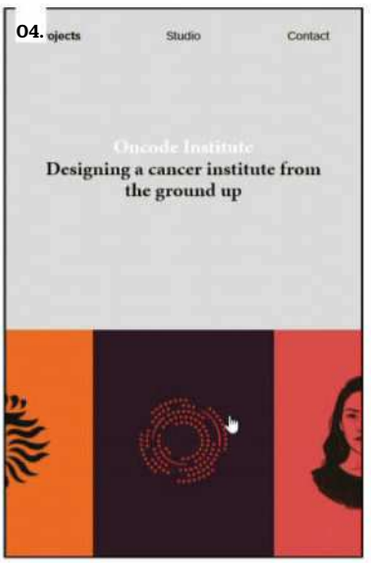


76.12%

Up nearly 1% on the previous month

Source: gs.statcounter.com (correct as of March 2018)

Sites of the month



01. Kontrapunkt
bit.ly/2DDrTFQ
A font that resounds to sound. Play onsite to see how it works.

02. Webflow Ecommerce
webflow.com/ecommerce
Animations abound. Scroll down to see the magic unfold.

03. Playground
bit.ly/2GuCdlR
Discover the delights of 3D in this interactive game.

04. Oncode Institute
momkai.com
Transition animations with a touch of the out there.

Graphics
Gibberish Worlds 6
bit.ly/2GRjRie
Collection of surreal illustrations from the very talented hand of Ori Toor.

Colour picker
Cool Kids
bit.ly/2H3aAI5

#DA57AO
#7C2A5F
#E2A1C8
#249ACF
#186385

Typesetter
Zesta
bit.ly/2H5dJAX
Zesta is a family of high-contrast serif fonts, intended for use in display typography.

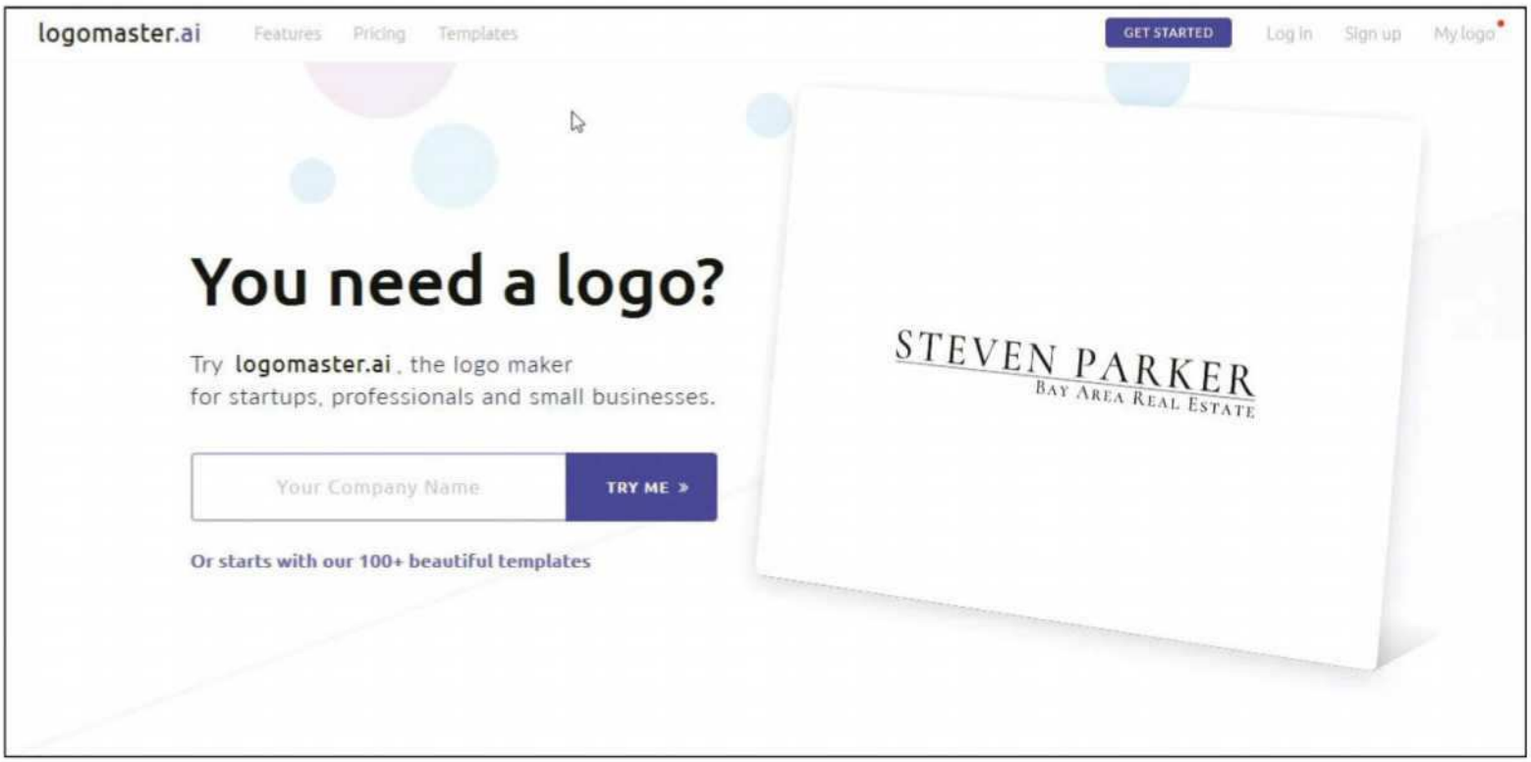
ABCabc
0123456

WordPress
StuffPost
bit.ly/2uE9rOd
A strong contemporary WordPress theme that's perfect for news-led sites.

CONTACT US AT: steve.jenkins@futurenet.com | @WebDesignerMag

webkit

Discover the must-try resources that will make your site a better place



Logomaster

logomaster.ai

Need a logo fast? Then this neat tool is a quick option for getting a business card built in minutes. Kick off by adding your company name and hitting the Try Me button. Then it's a quick step process of selecting a category, logo style, colour, slogan and fine-tuning to get the final result. When finished save for further editing or buy to keep.

Checkbot

checkbot.io

This is a browser extension that checks for broken links, duplicate titles, redirect chains, invalid HTML/JS/CSS and more. Plus, it can boost your SEO.

Duotone

duotone.shapefactory.co

Need to create complementary dutones in double-quick time. Upload an image, choose two colours (or pick from preset options) and download.

CSS Gradient

cssgradient.io

Need a code for the perfect CSS gradient? Check out this tool for the answer. Decide size, colour, type, angle and copy the generated code.

TOP 5 Web conferences – June 2018

Get yourself a seat at the biggest and best conferences coming your way soon

Kerning

2018.kerning.it

A two-day conference (with two days of workshops) in Italy and dedicated solely to typography and web typography.

UX Scotland

uxscotland.net/2018

A hands-on, practical UX & design conference where participants can connect and learn from their peers.

CSS Day

cssday.nl/2018

Head to Amsterdam for two days of CSS and UX insight from leading experts, including Vitaly Friedman and Sara Soueidan.

An Event Apart

bit.ly/2D70LAt

A host of well-known speakers focusing on digital design, UX, content, code, and much more for three days.

Webinale

webinale.de/en

A five-day event in Berlin aimed at digital professionals, trendsetters and web makers with over 70 speakers.

Cloud hosting or shared hosting?

Pay more, pay less, but which option offers the best for your business

Neal Thoms
Content Editor
fasthosts.co.uk

> “Shared hosting would seem to have a major advantage – cost. But the downside is that it may have limitations that become more apparent as time goes by”

To have a live website on the internet, you're going to need web hosting, but of the two most common hosting choices today – cloud hosting and shared hosting – which one is the best for your business? So, what's the difference? Put simply, cloud hosting is a service that is dedicated to you, while shared hosting means that you have to share the server resources with other users.

At first glance, shared hosting would seem to have a major advantage – cost. But the downside is that it may have limitations that become more apparent as time goes by and requirements grow. Think of it like sharing a flat with several other people. You may save money by pooling resources, but sometimes you want the place to yourself. That's when cloud hosting comes into its own.

With several customers sharing one set of physical hardware, individual users can't always rely on a guaranteed level of server performance. This might not be an issue for some sites, but when it comes to running intensive, business-critical processes, many users find that shared hosting facilities can cramp their style. Ninety-nine per cent of the time this is fine; but what happens when you're sharing your space with sites that have a spike in demand at a certain time of day or night, or even at certain times of the week or year? Will your website functionality suffer as a result?

The server only has a finite amount of resources to parcel out, so applications will regularly struggle to get what they need. The end result could be inconsistent performance, slow loading times and a frustrating experience for your customer. If your website is a major source of sales or new business enquiries, this could be a problem. What if you're the one drawing down more resources than everyone else? Then your host may send you a warning notice or two and ultimately even suspend your account. Another issue might be if you need a dedicated IP address. You might want this if you need to maintain a secure site via an SSL certificate or if you need to access your website via FTP on a regular basis. In both these cases, you'll want a dedicated server.

Or do you need server root access so that you can edit all the files on your server (including the mission-critical ones), install your own software and applications or change the system configuration? If the answer is yes, then you'll need access to the server, something you won't have with shared hosting.

With cloud hosting, every single resource you configure is dedicated to you alone. Cloud hosting companies provide virtualised resources on an on-demand, as-needed basis. Instead of paying upfront for a fixed configuration on a single server, the user can pay as they go for what they actually use. With cloud hosting, the load can also be balanced across a number of different servers, meaning that if an individual server goes down, there is no lost information or downtime for the customer. In this way, cloud hosting can be much more flexible and resilient to the fluctuating demands of business. To think about it another way, cloud hosting can be a kind of 'dedicated hosting' option for users who want the increased power of their own server, but don't want to have to deal with the day-to-day issue of server management – checking the status of the server and apps, monitoring for any new or recurring issues and so on. Cloud hosting also eliminates the need to ever migrate your server in the future when it inevitably reaches end-of-life, a pain all web designers can relate to!

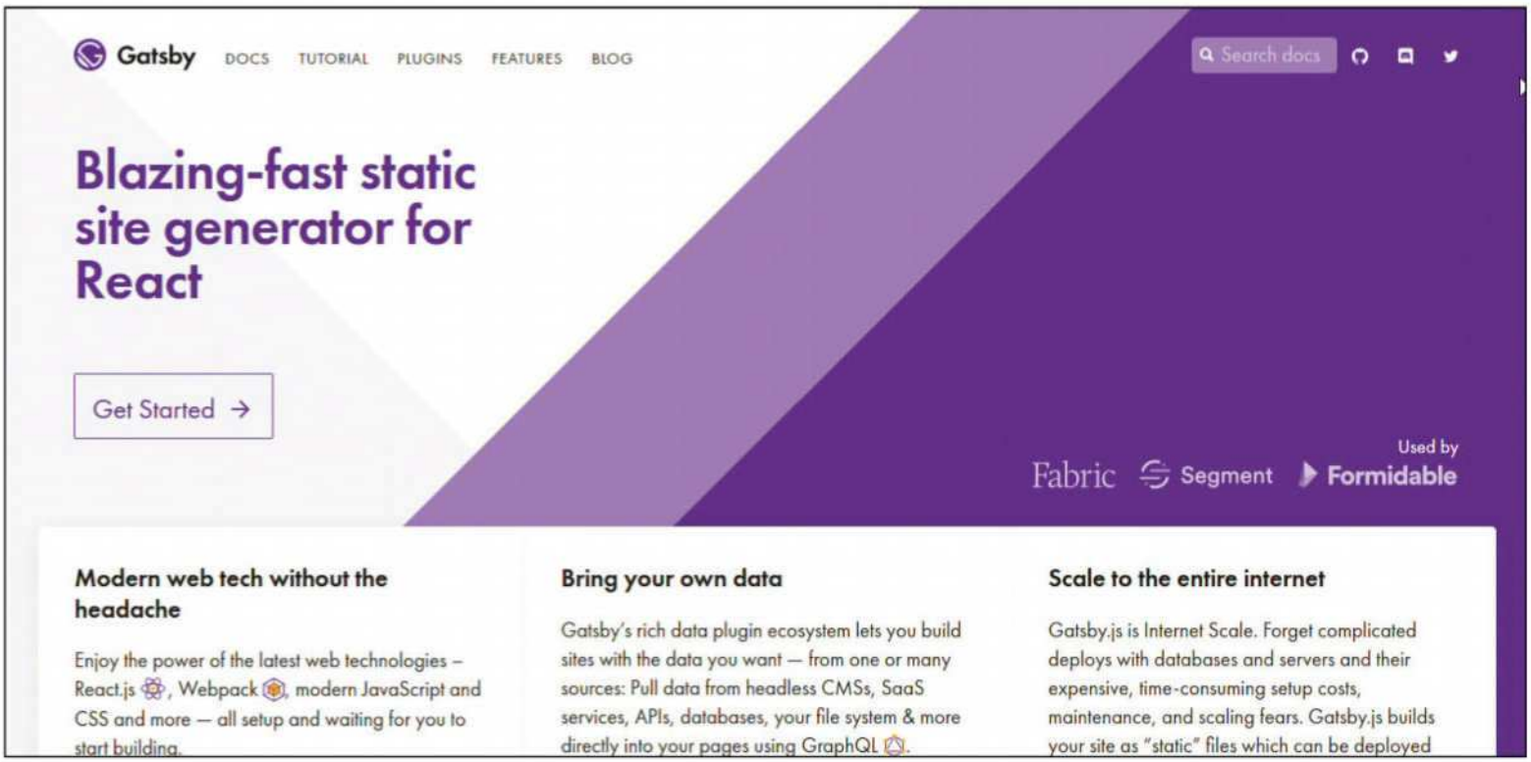
Cloud hosting is a great option for many people, but maybe you still feel it's too much of a new and untried concept to risk putting your business on? This is simply untrue. While the term 'cloud hosting' might be relatively new in the web hosting world, it is built on established and tested technologies like virtualisation, so it is more mature than you might think.

If your business is already heavily invested in the cloud – for example you use Dropbox, Google Drive, Amazon Web Services, Salesforce, Office 365, etc. – then you already know many of the benefits of cloud hosting. Providing you can see a benefit from the additional value that cloud hosting delivers over shared hosting (greater resilience, reliability, scalability, future-proofing, etc.), then the additional cost of cloud hosting may be a no-brainer for you.

CONTACT US AT: steve.jenkins@futurenet.com | @WebDesignerMag

webkit

Discover the must-try resources that will make your site a better place



GatsbyJS

gatsbyjs.org

Gatsby is a blazing-fast static site generator for React. So what does it have to offer? A static PWA (Progressive Web App) generator, it builds sites fast using the power of the latest web technologies – React.js, Webpack, modern JavaScript and CSS, and more. And it enables you to pull in data from a host of sources using GraphQL.

Pigment

bit.ly/2ImNyF4

Explore colour combinations via the use of pigment and lighting. A grid of colour combos can be tweaked and seen fullscreen and copied.

Dragon Drop

bit.ly/2q07mqc

Dragon Drop is a drag-and-drop list reorder plugin. It offers straightforward reordering of list items plus more complex options.

Majestic

bit.ly/2EeasMF

Testing with Jest? This app provides a UI for running tests with Jest. Features include support for Create-React-App out of the box.

TOP 5 Codepens

Head over to the code playground and get inspiration for your projects

Airplane Toggle

bit.ly/2Eep2Ue

Toggle switches with added character can say so much in such a small space. This is a great example of the art.

Transforming Nutrition Facts

bit.ly/2H36Y2i

Who would have thought that labels could be so interesting. This codepen demonstrates how to reformat information for different shapes and sizes.

3D Hex Mosaic

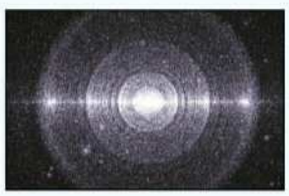
bit.ly/2q3Dfyr

A large image is overlaid with a maze of hexagons that move in the direction of the cursor and reveal more on hover.

Sausage Factory

bit.ly/2H4O5vV

The power of ThreeJs is keenly demonstrated here with shaders in what looks like a giant sausage factory.



The Big Bang

bit.ly/2H1rWyx

GPGPU particles using texture caches for position and velocity. Take a look – it looks even better in action.

SPRING SALE!

TRY 5 ISSUES FOR £5*

BIG SAVINGS ON OUR BEST-SELLING MAGAZINES

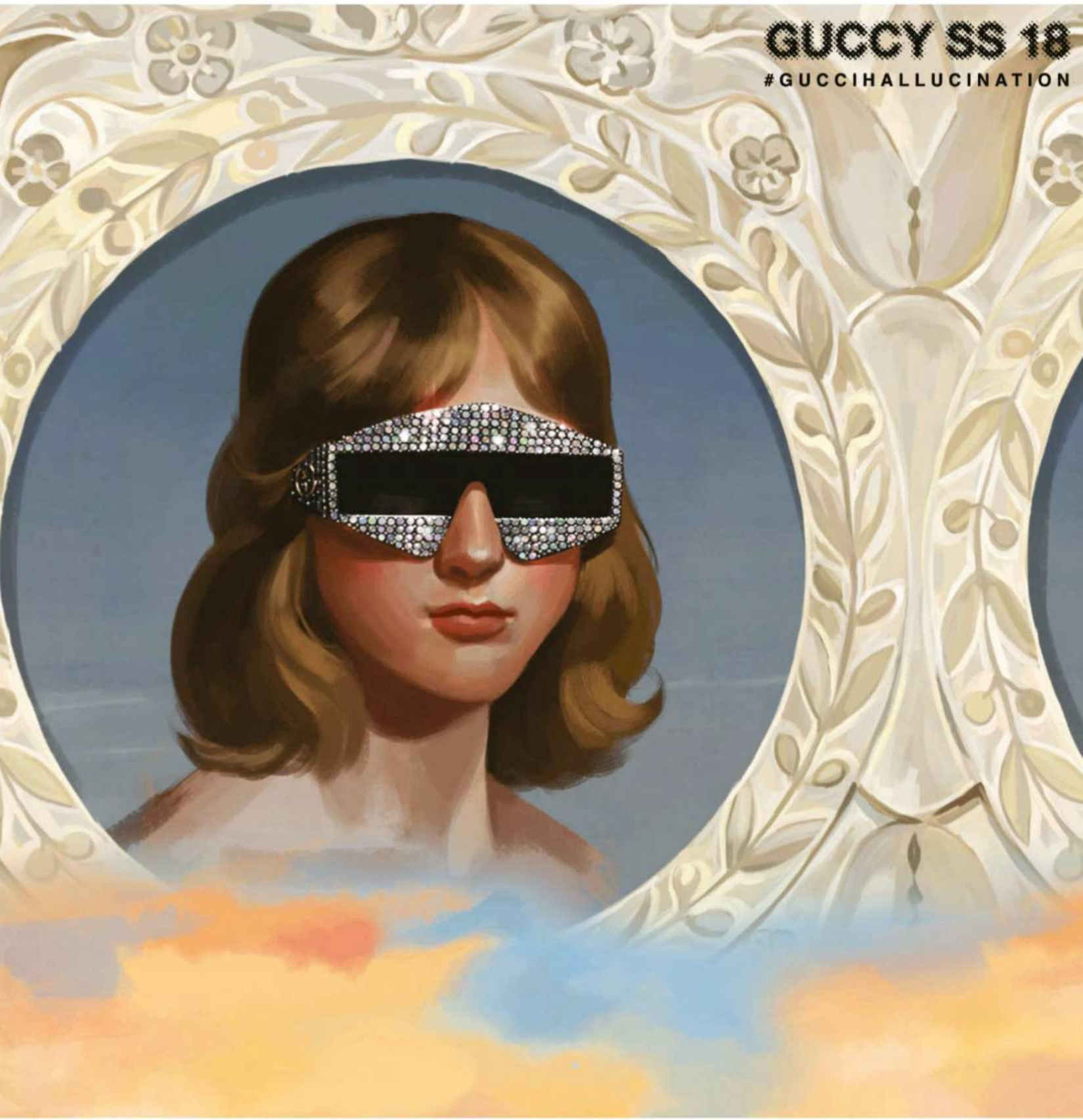
For great savings on all of our magazines, see the entire range online

myfavouritemagazines.co.uk/spring182

Order Hotline 0344 848 2852

*TERMS AND CONDITIONS: The trial offer is for new UK print subscribers paying by Direct Debit only. Savings are compared to buying full priced print issues. You can write to us or call us to cancel your subscription within 14 days of purchase. Payment is non-refundable after the 14 day cancellation period unless exceptional circumstances apply. Your statutory rights are not affected. Prices correct at point of print and subject to change. Full details of the Direct Debit guarantee are available upon request. UK calls will cost the same as other standard fixed line numbers (starting 01 or 02) or are included as part of any inclusive or free minutes allowances (if offered by your phone tariff). For full terms and conditions please visit: bit.ly/magtandc . Offer ends 30 April 2018.

ENTER THE WORLD EXPOSITION



http://springsummer.gucci.com

Designer:

LOW http://low.thebignow.it

"Italian agency LOW is credited with a digital journey through the Spring Summer 2018 Gucci Hallucination campaign illustrated by artist Ignasi Monreal"

Colours

#5D707F

#D1C2AA

#FACF89

#523A28

Tools

jQuery, HTML5, Three.js

Fonts

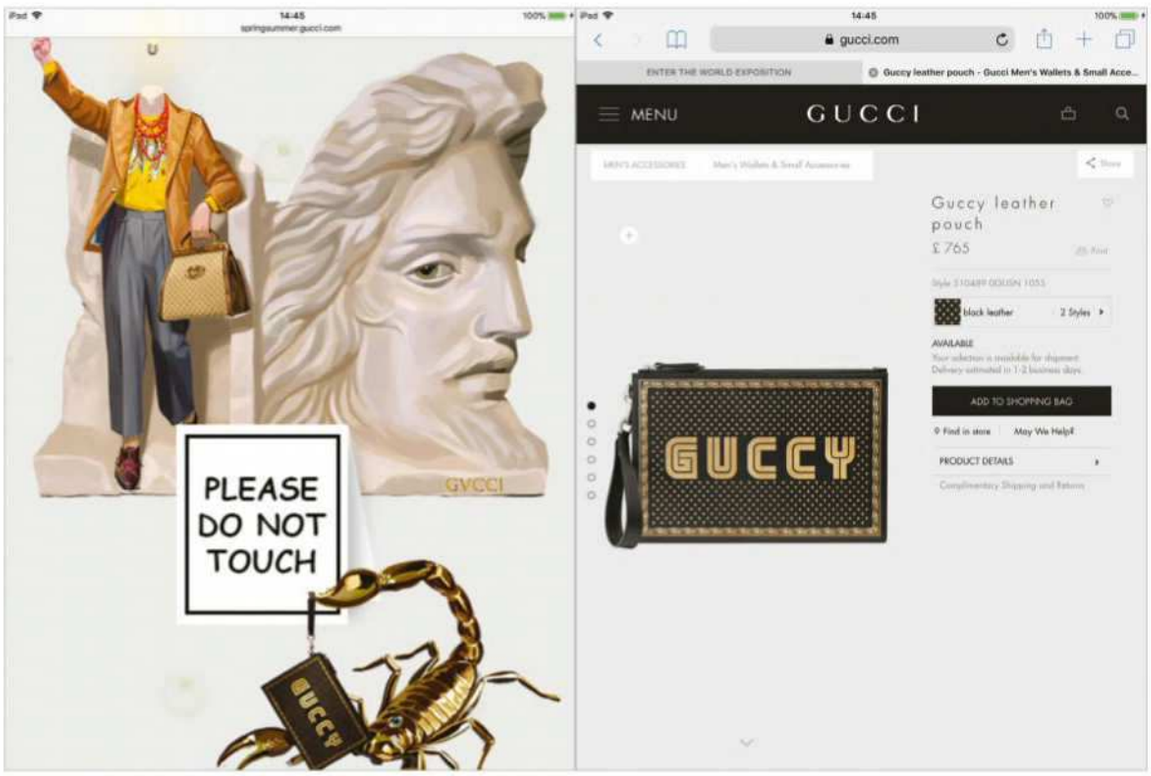
abcABC 1234567890

abcABC 1234567890

abcABC 1234567890

Using just one font on a tiny sliver of caption text, the site features Helvetica Neue in Bold, Italic and Regular

Above
A scrolling dream-like experience, the site's visuals are built solely around the quirky illustration work of artist Ignasi Monreal



Above
Ignore the warnings and click or tap through the imagery to link out to the product line and purchase a purse

Above
Even on touchscreens the site reformats to keep the flow identical

Create an animated bird box with HTML5 & CSS

An animated effect of flying birds that can be used to decorate your website content

1. Initiate document

The first step is to initiate the page document. This consists of the HTML container, responsible for storing the head and body sections. While the head section is used to load the external CSS and JavaScript resources, the body section is used to store visible content elements created in later steps.

```
<!DOCTYPE html>
<html>
<head>
<title>Birds Animation</title>
<link rel="stylesheet" type="text/css"
href="styles.css" />
<script src="code.js"></script>
</head>
<body>
*** STEP 2 HERE
</body>
</html>
```

2. Picture container

The animation is to be placed inside an article container that will use the 'data-picture' attribute. Use of a container provides better flexibility to control the animation from CSS and JavaScript. It also makes sure that animation elements can be kept separate from other web page content to avoid potential problems.

```
<article data-picture>
</article>
```

3. CSS initiation

With the HTML now complete, you should create a new file called 'styles.css'. This step sets the page document and its body to have no visible border spacing. The page will appear with a black background, while the default colour for content to inherit is set to appear as white.

```
html, body{
  background: #000;
  padding: 0;
  margin: 0;
  color: #fff;
}
```

4. Picture container and children

The container is set to a specific size using fixed positioning. The fixed positioning guarantees that the animation is always visible regardless of scroll position; change this to 'absolute' if you want it to scroll with the content. An animation called 'move' is applied to repeat with a duration of 20 seconds. All child elements are set to display as block elements using absolute positioning.

```
[data-picture]
{
  position: fixed;
  width: 10cm;
  height: 2.5cm;
  animation: move 20s infinite;
}
[data-picture] > *
{
  position: absolute;
  display: block;
}
```

5. Animation definition

The 'move' animation applied in the previous step defines attribute states to transition between at key parts of the animation. This example brings the animation with container positioned out of view, transitioning into view at 30% of the way through the animation. The final frame, at 100%, moves the picture container out of view.

```
@keyframes move {
  0%{ left: -10cm; top: 0; opacity: 0;}
  30%{ top: 50%; opacity: 1;}
  100%{ left: 100vw; top:0; opacity: 0;}
}
```

6. Bird creation window load

The bird HTML is generated by JavaScript – keeping the HTML simple and allowing an opportunity to increase the bird count without requirements for extra code or content. This can only be done when the page has fully loaded, hence the code being inside a function that is triggered in response to the window load event.

```
window.addEventListener("load", function()
{
  *** STEP 8 HERE
});
```

7. Bird HTML

The bird HTML is created from the JavaScript through a loop that counts from zero to the target amount defined by the 'birds' variable. A random size and coordinate is also selected for each bird before it is added to the HTML content string.

```
var html = "";
var birds = 30;
for(var i=0; i<birds; i++){
  var size = Math.random()*1.5;
  var x = Math.random()*100;
  var y = Math.random()*100;
  html += "<span style='font-
size:"+size+"em; left:"+x+"%;
top:"+y+"%;'>&gt;</span>" }
```

8. HTML to target

The final requirement of the JavaScript is to send the generated HTML to any page element using the 'data-picture' attribute. The querySelectorAll is used to get a list of all of these elements so that the generated HTML can be applied to them through their 'innerHTML' attribute.

```
var nodes = document.
querySelectorAll("[data-picture]");
for(var i=0; i<nodes.length; i++){
  nodes[i].innerHTML = html;
}
```

The Mads https://themads.agency

Designer:
The Mads https://themads.agency

“Russian digital creatives The Mads Agency present a portfolio site that blends clean minimalism with bold project previews”

Colours

#00759D #B8335E

#E5BFBE #564D57

Tools

jQuery, GSAP, Bitrix (CMS), AJAX Libraries API

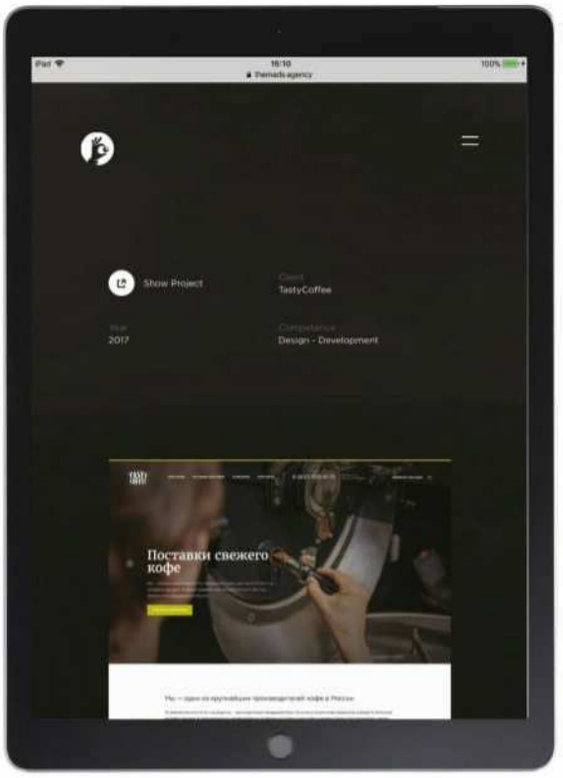
Fonts

abcABC 1234567890
abcABC 1234567890
abcABC 1234567890

The Gotham Pro family is used across project titles and links in Regular, Light, Italic and Bold typefaces.

Above
Opening to a splash screen flooded with negative space, the site adds subtle GSAP animations to distinctive red line drawings

Above
Employing a fullscreen timed carousel, three featured projects transition in sequence inviting visitors to click through and view



Above
On every device, selected projects are detailed via key information, links to view and full-length screenshots

Learn how to create an animated cursive writing effect

A visually animated text effect to present your messages in the style of traditional calligraphy

1. Define the HTML document

The first step is to define the HTML document. This consists of the HTML container, which is used to store the head and body sections. While the head section is used to load the external JavaScript, the body section is used to store page content elements created in the next step.

```
<!DOCTYPE html>
<html>
<head>
<title>Cursive Writing Effect</title>
<script src="code.js" type="text/
javascript"></script>
</head>
<body>
*** STEP 2 HERE
</body>
</html>
```

2. Drawing canvas

The writing effect is dependent on the use of pixel manipulation, meaning the best tool for the job is the canvas element. This provides access to more specific tools for drawing effects than available from regular DOM elements styled with CSS. The canvas is set to a specific width and height – allowing it to act like an inline image element.

```
<canvas width="800" height="150"></canvas>
```

3. Canvas reference after load

Create a new file called 'code.js'. This first step of JavaScript waits for the page to complete loading; then it will search for the page canvas and to get a 2D drawing referent. This is stored as the 'ctx' reference name – to be used for drawing operations in the upcoming steps.

```
window.addEventListener("load", function()
{
  var ctx = document.
querySelector("canvas").getContext("2d");
  *** STEP 4 HERE
});
```

4. Variable definitions

Several variables are defined to control the output presentation. While 'brushWidth', 'brushOffset' and speed are used for calculations relating to the drawing effect, 'txt' is used to define what text is to be drawn to the canvas. The x variable is used to control the horizontal drawing position – set to start at 30 pixels.

```
var brushWidth = 200;
var brushOffset = brushWidth;
var speed = 7;
var txt = "Opportunities multiply when
seized.";
var x = 30, i = 0;
*** STEP 5 HERE
```

5. Drawing settings

Settings for the 'ctx' drawing reference defined in step 3 are applied, defining the font to use, outline width and inner colour to apply within the text – set to black. You should take note how these properties accept the same type of values used for their CSS counterparts for font and colour definitions.

```
ctx.font = "1cm cursive";
ctx.lineWidth = 3;
ctx.fillStyle = "#000";
*** STEP 7 HERE
```

6. Drawing function

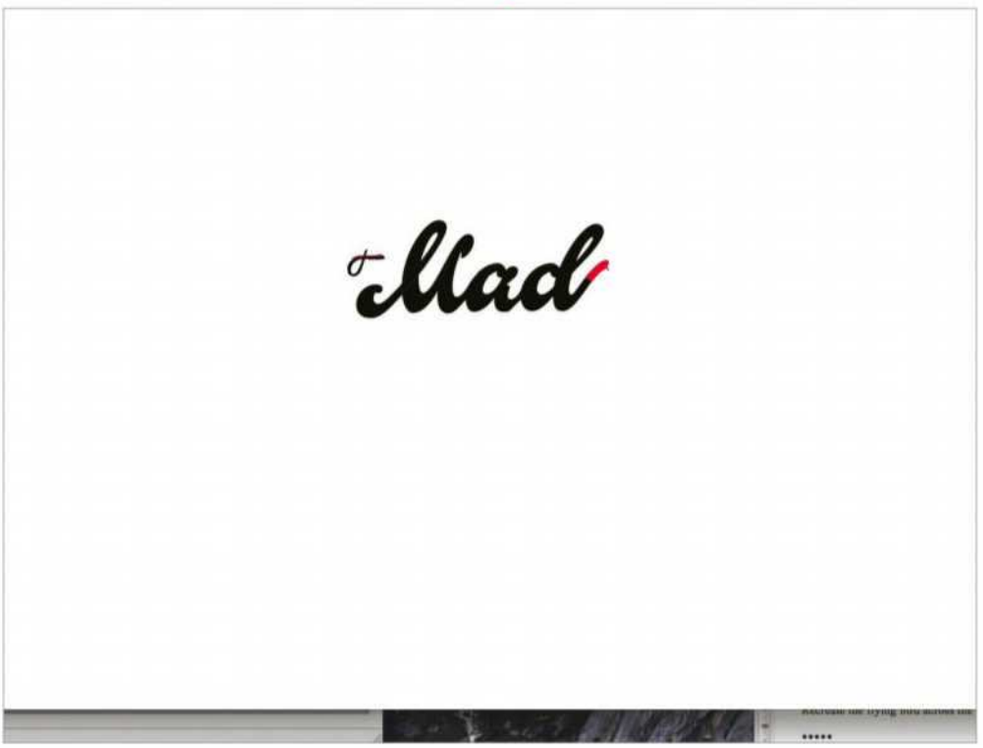
A function called 'draw' is immediately executed to initiate the drawing effect of the text. Firstly, the space is cleared for a clean drawing display. Calculations are used with the drawing reference 'draw' part of the text – like a brush stroke. The 'i' and 'x' variables refer to the current letter and its position to present as partially drawn.

```
(function draw() {
  ctx.clearRect(x, 0, 800, 150);
  ctx.setLineDash([brushWidth -
brushOffset, brushOffset - speed]);
  brushOffset -= speed;
  ctx.strokeText(txt[i], x, 70);
  *** STEP 7 HERE
})()
;
```

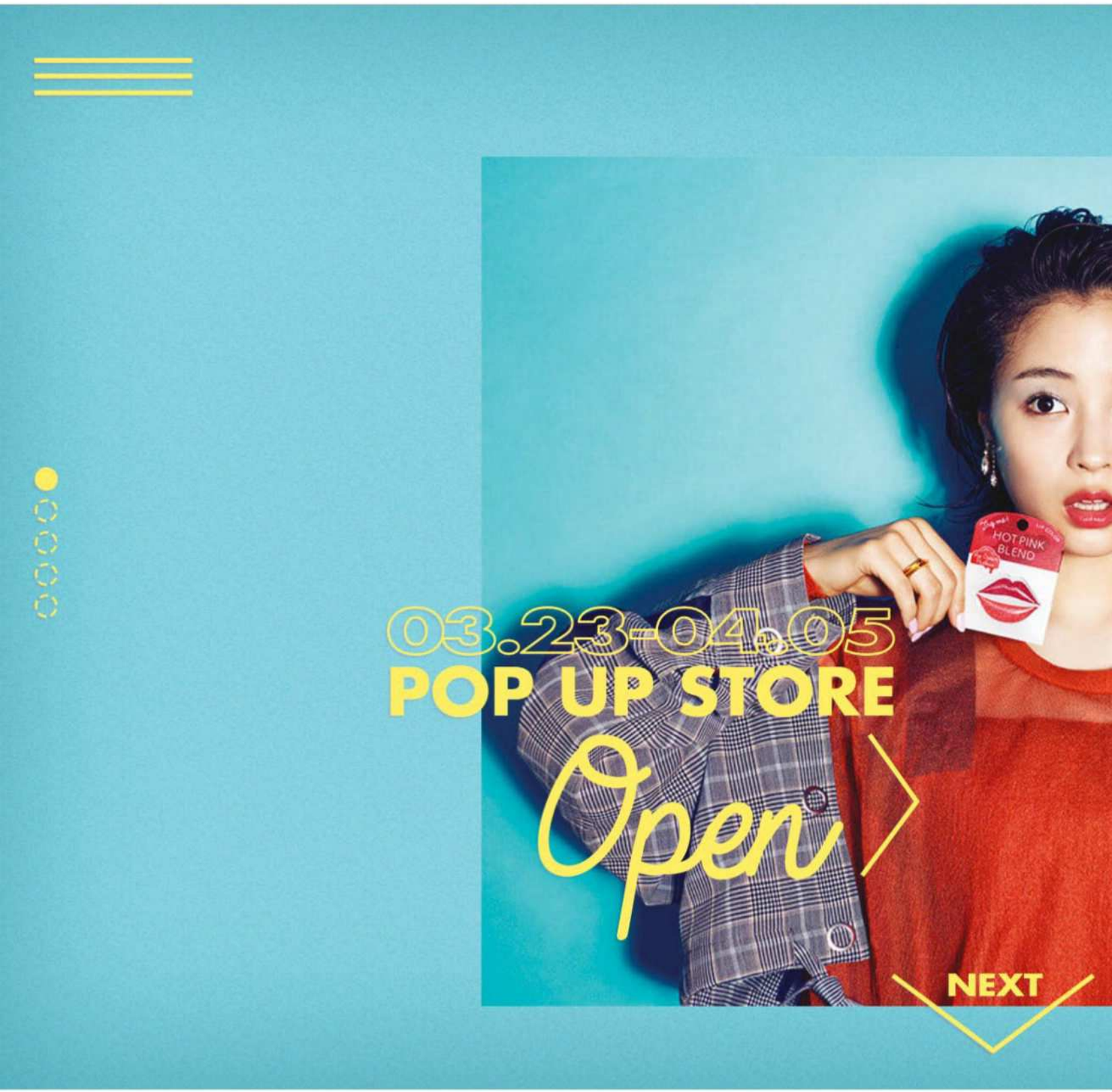
7. Drawing recall

While the current letter has not yet completed drawing, the same 'draw' animation is recalled. When the letter has been completed, some of the settings for the next letter in the text are identified before the 'draw' animation is recalled, if the end of the text has not yet been reached.

```
if (brushOffset > 0)
requestAnimationFrame(draw);
else {
  brushOffset = brushWidth;
  x += ctx.measureText(txt[i++]).width +
ctx.lineWidth * Math.random();
  if (i < txt.length)
requestAnimationFrame(draw);
}
```



Ice Cream Parlour Cosmetics



www.shiseido.co.jp/icecosme

Designer:

Shiseido Inc | Hiroka Hasegawa http://hirokahasegawa.com

"Echoing the sugary sweet ice cream tones of Shiseido's makeup products, this Japanese cosmetics catalogue bursts with cutesy colour"

Colours

#81D5E3 #FFF579 #F36144 #7B7A8C

Tools

jQuery, SVG, HTML5, CSS3

Fonts

abcABC
1234567890

Futura Pro Medium font by Paul Renner for Neufville Digital and LinoType is used on any English text snippets.

abcABC
1234567890

Hiragino Kaku Gothic Pro is employed to style the Japanese Kanji text on the footer links.

Create a slide out menu

A slide out menu allows you to make full use of screen space for user navigation

1. Document initiation

The first step is to define the page document. This consists of a HTML container representing the webpage, which contains the head and body section. While the head section is used to load the external CSS and JavaScript resources, the body section is used to store the visible page content created in step 2.

```
<!DOCTYPE html>
<html>
<head>
<title>Slide Out Menu</title>
<link rel="stylesheet" type="text/css"
media="screen" href="styles.css"/>
<script src="code.js"></script>
</head>
<body>
*** STEP 2 HERE
</body>
</html>
```

2. Page content

The page content consists of a heading title, along with a navigation container. This navigation stores a series of links and has been assigned a 'data-action' attribute. It is this attribute that will be used by the JavaScript and CSS to apply styling and functionality to the container and its elements.

```
<h1>Slide Out Menu</h1>
<nav data-action="expand">
  <span>&#9776;</span>
  <a href="#">Page 1</a>
  <a href="#">Page 2</a>
  <a href="#">Page 3</a>
</nav>
```

3. CSS initiation

The HTML is now complete, so create a new file called 'styles.css' to initiate the presentation formatting. This step sets the HTML document and its body to have no visible border spacing, along with a black background. Colour is set to white as the default colour for content text to inherit.

```
html, body{
  display: block;
  width: 100%;
  height: 100%;
  background: #000;
  color: #fff; }
```

4. Navigation setup

The navigation is to display with fixed positioning and with a z-index above everything so that it can appear to cover the full screen regardless of where the user has scrolled to. It is initially positioned out of view on the left side of the screen's visible viewport. A transition rule is applied to animate any changes over a duration of one second.

```
nav{
  display: block;
  position: fixed;
  box-sizing: border-box;
  top: 0;
  left: -100vw;
  z-index: 9999;
  padding: .5em 1em .5em 1em;;
  width: 100vw;
  height: 100vh;
  text-align: center;
  background: red;
  transition: all 1s; }
```

5. Nav open and icon

The navigation's left position is set to zero when an 'open' class has been applied, triggering the animated transition defined in the previous step. The navigation's first child is the expand icon, which uses fixed positioning to always remain visible in the centre of the left side of the screen.

```
nav.open{
  left: 0;
}
nav *:first-child{
  position: fixed;
  padding: 1em;
  cursor: pointer;
  left: 0;
  top: 50vh;
  clear: both; }
```

6. Navigation links

Each anchor link inside of the navigation container is set to be four times the size of the default text. Their colour is set to black, with a margin applied at their top to guarantee vertical spacing is visible. Setting their display as 'block' makes each link automatically appear to be stacked vertically.

```
nav a{
  display: block;
  font-size: 4em;
  color: #000;
  font-family: arial;
  text-decoration: none;
  margin-top: .2em;
}
```

7. JavaScript listener

Create a new file called 'code.js'. This step waits until the page has loaded, upon which it searches for the first child of all navigations with the 'data-action' attribute set to 'expand'. This first child, being the open icon, has a 'click' event listener applied, upon which toggles the element to have or not have an 'open' class applied.

```
window.addEventListener("load",function(){
  var nodes = document.
querySelectorAll('nav[data-action="expand"]
*:first-child');
  for(var i=0; i<nodes.length; i++){
  nodes[i].addEventListener("click",functi
on(){
  if(this.parentNode.className == "open")
this.parentNode.className = "";
  else this.parentNode.className = "open";
  });
  }
});
***
```

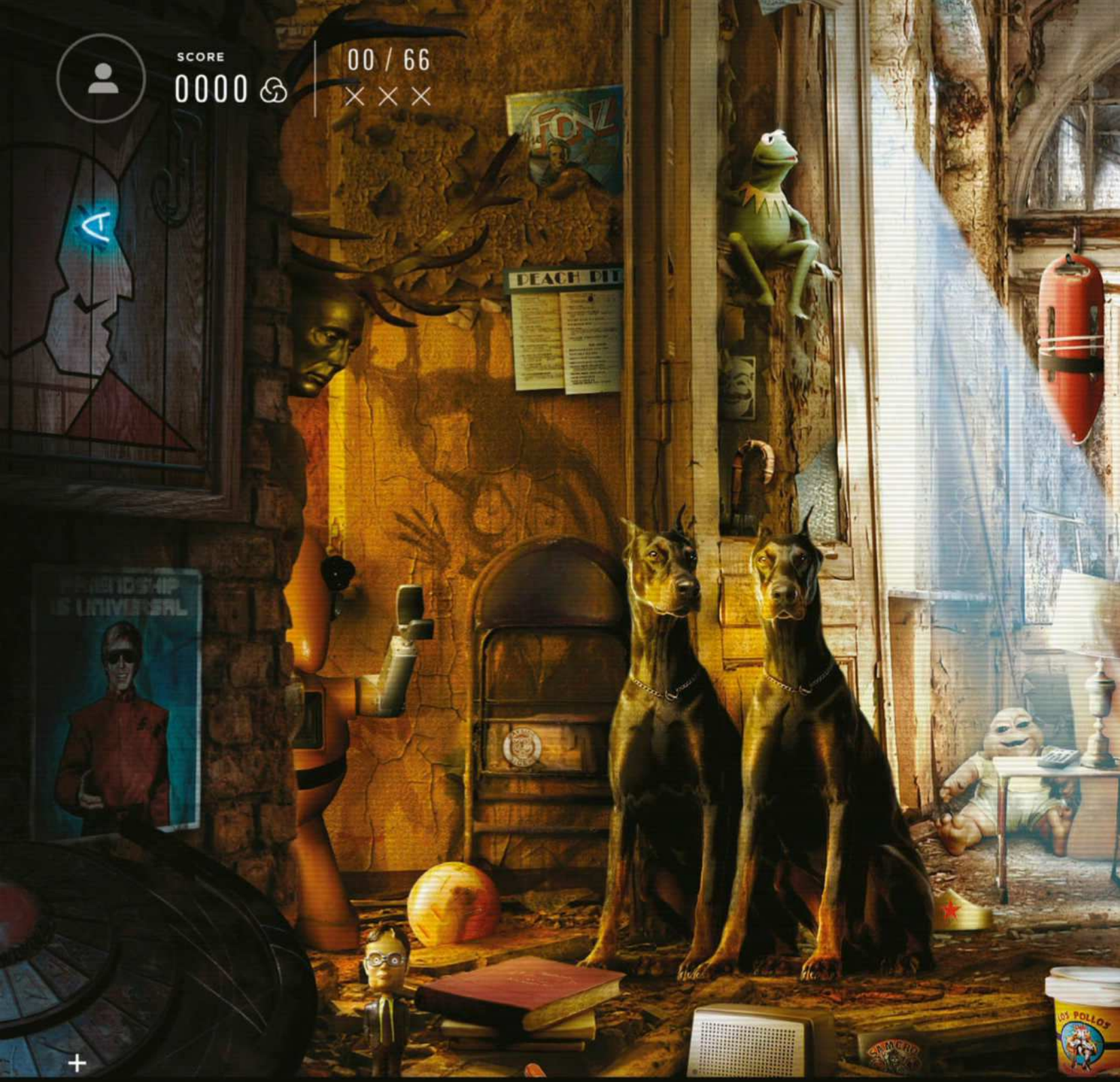
The number one destination for **web design** news, views and how-tos.

Get Creative Bloq **direct to your inbox** with our weekly web design newsletter

CB CREATIVE BLOQ

Graphic design | **Web design** | 3D | Digital art

www.creativebloq.com



Second Seasons

They say any sequel to a successful story is the toughest act to follow, so stay tuned for how PopCorn 66 re-ran with retro TV to land repeat glory online

PROJECT DURATION
18 months

NO OF PEOPLE INVOLVED
8

NAMES & POSITIONS

Eric "Elvis" Simonet
Music & Sound Designer

Vivien Ripoche
Back-end Developer

Julien Frey
French Copywriter

Cédric Borgel
English Copywriter

Claire Monica Burgess
English Copywriter

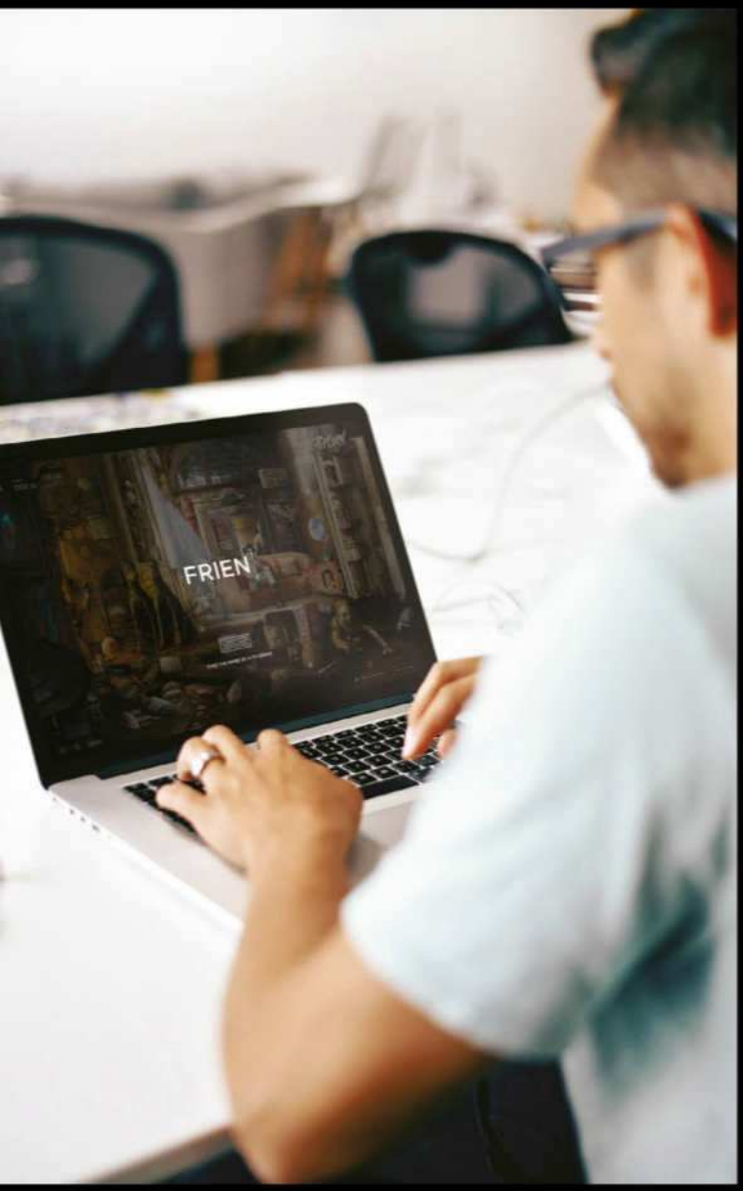
Chelsea Brown
English Copywriter

Priska
Partner & Art Director

Romain Zitouni
Partner & Creative Developer

MAKE NO BONES ABOUT IT, this age of the on-demand boxset has marked an incredible resurgence for television online. The hunger for high production, episodic and yes, blockbusting shows rivals the cinema box-office given the flexibility of consumption for viewers. Of course the beauty of this is the catchup element too, where a growing archive of old TV classics can be found and enjoyed in entirety by those who missed them first time around. To appreciate best our featured web masterpiece this month, this kind of 'homework' would indeed be recommended. This is because PopCorn TV, an in-house project for digital studio PopCorn 66, is a veritable dream for telly addicts. Based in Paris, France the agency specialises in creating rich digital experiences routed in pop culture references with PopCorn TV building on a previous outing to pack your browser full of them – literally. With nods to the 'hidden object' games popular on mobile devices, the website follows on from the movie-themed PopCorn Garage from 2015 to deliver an engaging digital quiz bursting with clues to iconic small-screen smashes. "PopCorn TV is the result of another project that we created two years ago, named PopCorn Garage (**www.popcorngarage.com**)," begins Romain Zitouni, Partner and Art Director at PopCorn 66. "This project was already challenging the internet on cinema culture with 66 hidden references in a garage. Since its release, it has been a huge success with more than 1.4 million players worldwide. Without the success of PopCorn Garage, PopCorn TV would not exist." A scary thought indeed, so after spotting this impressive follow-up for ourselves in issue 273's Lightbox we leapt from our sofas to hear from its makers on how the production came to light.

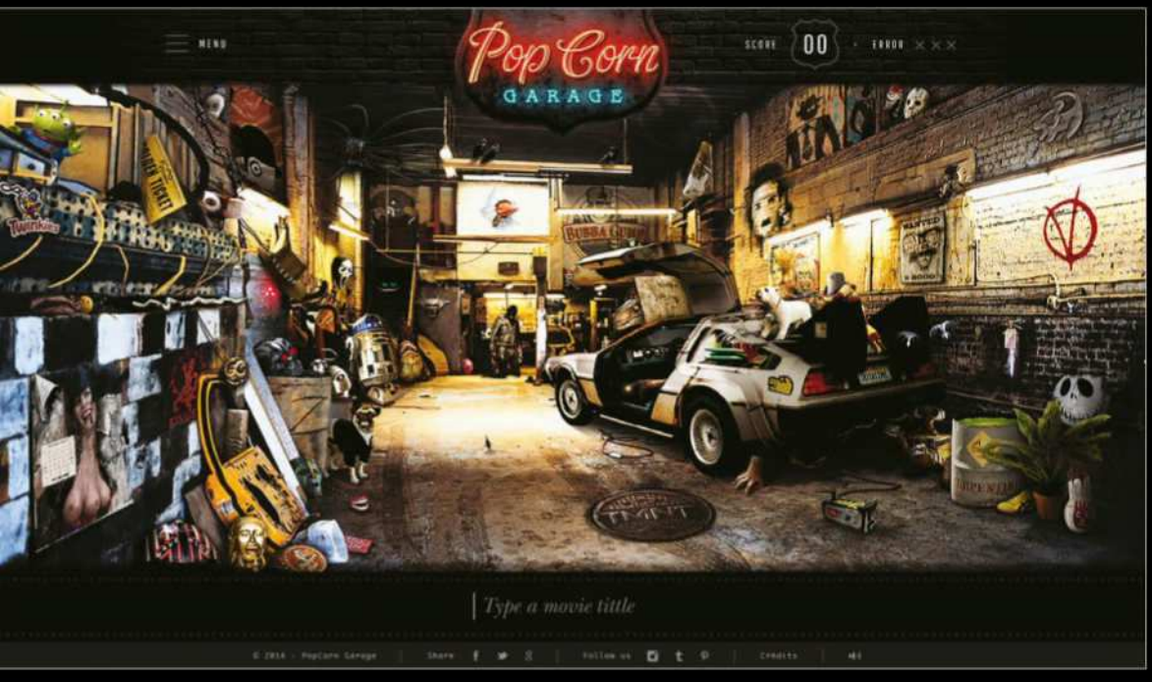
POPCORN TV
www.popcorntv.fr
by
POPCORN 66
www.popcorn66.fr
@PopCorn66_FR



DIFFERENT STROKES

"After PopCorn Garage, we received a lot of messages from players around the world asking us to do a sequel," reveals PopCorn 66 partner and Creative Developer Priska, when asked how the story started. "So we took the time to think carefully before embarking on a second version." Clearly aware of doing justice to the impact PopCorn Garage had made on web audiences, makers PopCorn 66 didn't want to simply rush any follow-up. The new project would of course also be an in-house promotional effort, not commanding any kind of direct commercial incentive or support so scheduling was an issue too.

"PopCorn Garage and TV are proactive projects without any funding so it was therefore necessary to get organised enough to unlock the time to work on it." But the passion certainly remained, with Priska confirming the studio's constant fascination and inspiration around pop culture. But this time would be a chance to "explore another universe" within this, demanding a new theme, new décor and therefore a brand-new name too. "We also wanted to surprise and challenge each other too," agrees Romain. "The word that characterizes us is 'PopCorn' and not Garage. So for this second version, we didn't want to remake a new Garage and lock ourselves into the cinema theme." Plus the guys would have complete free reign to make those judgement calls. Apart from their own creative pressure and pride, there would be no customer or client to impress. Once a suitably fresh theme was settled on, the stage would be set and green-lit to go.



THEMATIC TENSION

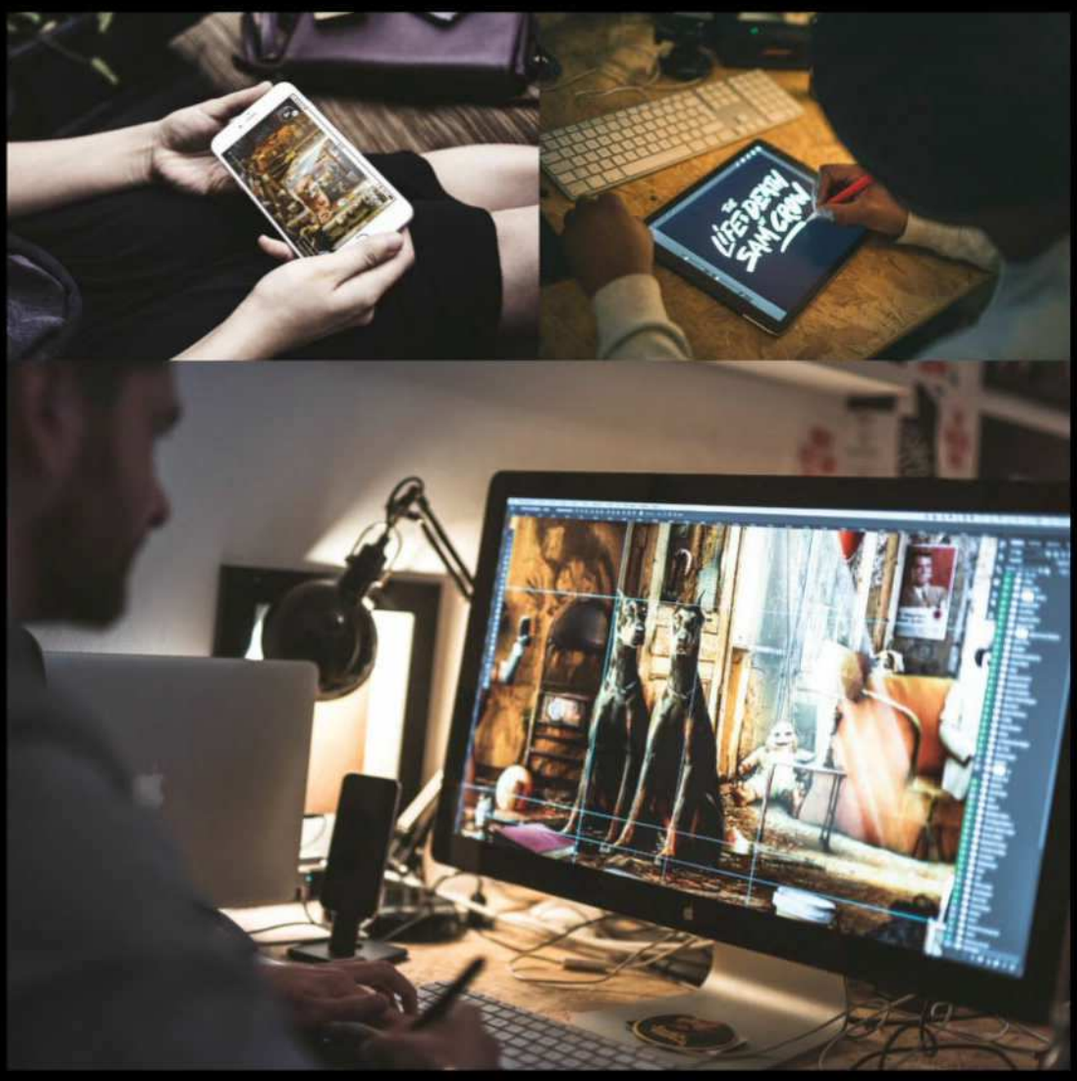
"For the original idea, we must go back to the birth of Popcorn Garage," Romain concedes when recalling how the new idea came about. "That project started from a very simple principle – the summarising of a cult film in one object. From there, we formulated a set with several references to films that had left their mark on us. The idea of turning these into an interactive game that defies all film fans then seemed a pretty natural extension. It was then necessary to find effective game mechanics for making the experience simple and addictive." This was where the team pointed towards those hidden object discovery games, a kind of *Where's Wally* for spotting the visual hints and metaphors relating to each film. So if the mechanic was already in place and proven to be popular, it would now be about arriving at a pop culture theme not a million miles away. "Our first goal was to find the right theme," Priska confirms. "So we asked our close friends and some players, about 5,000 people in total, through a small survey to acknowledge their desires. The majority answered in favour of the TV series," something PopCorn 66 themselves approved of too. "Yeah the TV series was a perfect transition from the cinema theme," adds Romain. "The universe of the TV series is extremely rich, from the 50s to nowadays. Now the TV series has become more and more creative, to the point of rivalling the cinema. Some even create a real addiction to the point of commanding their own social networks, such as *Game of Thrones* or *The Walking Dead* etc. For us, it was an incredible playground to explore." With theme choice made, the focus shifted to devising a more sophisticated gaming experience while retaining PopCorn garage's simplicity. Here the guys took advice from friends Mickael Dauphinot and Alban De Fortaleza, founders of the agency Le Singe and specialists in Agile methodology – helping to organise ideas and define priorities. "Here we thought about several improvements to satisfy all players," explains Romain. "Creating a new 4K visual that could integrate a zoom function, an increase in playing

time by offering mini-quizzes, a more immersive approach to sound design and the proposition of a world ranking to challenge friends with."

A UNITED FRONT

The visual design work would of course centre on the creation of a playing field that contained 66 TV show references. PopCorn Garage quite simply was styled as a lockup garage/auto shop, providing a sense of place that matched the theme. Romain reveals, "PopCorn TV had to have its own place that corresponds to the universe it inhabits so naturally we thought of the television metaphor. Where before the gaming experience led us to open up a Garage, for version two we turn on and off a TV." This concept would throw up front-end challenges such as recreating analog TV effects, a hurdle that would dictate technological choices as Priska confirms: "The only technology that can achieve this kind of effect is the WebGL; I opted for the very famous library Three.js." "Then the biggest creative challenge was to create the game scenery in 4K to integrate a zoom function," Romain interjects. "The idea was to pace the hallmarks of classic sitcom décor, chiefly living room, sofa and fridge etc, into a dirty atmosphere that remains our graphic signature. Once the mood was established, it was necessary to choose 66 TV series references both recognisable and relevant to the location. The gaming experience is based on how all the references are arranged in the image, requiring a balance between the immediately identifiable objects and those more complicated to find – all while keeping a visual coherence that maintains overall elegance." Similarly, although 3D technology is used within the display PopCorn TV's front-end

Below
Popcorn TV can be enjoyed across all screens



Safe in sound

A uniquely vital element of the PopCorn TV project was in its use of music, something that doesn't always feature much beyond a background loop. The creative team in this instance applied high production values, befitting the classic show references hidden in the quiz itself. "The other big creative challenge was to create an original approach to sound design that would reinforce a universe steeped very much in nostalgia," explains Creative Designer and Art Director on the project, Romain Zitouni. "So here we chose the synth wave genre, a musical trend inspired by the retro sounds and electronica of the 1980s. This lead to a total of six incredible tracks being composed especially for the game by the very talented Eric 'Elvis' Simonet." In fact, fans of PopCorn TV will appreciate Romain's coyness here when hinting at the music's importance to the live experience. Eric's original soundtrack is frankly awesome and it's posting on SoundCloud (**https://bit.ly/2Gz6vY1**) encourages enjoyment in its own right. Reminiscent of Cliff Martinez's soundtrack for the 2011 motion picture *Drive*, the synth-heavy score is even joined by PopCorn TV's own neon pink logo to lend the site a familiar 'cult' vibe from the outset. "Like every cult series, PopCorn TV had to have its own credits," beams developer Priska. "So we created a four-minute trailer from different series extracts accompanied by the track *Growing Strong*, custom created by Eric, requiring hours of editing. But the credits immediately introduce the site and do a great job of putting the player in the mood."

Manufactured pop

One of the many interesting things surrounding this unique project is the role it has played in reinforcing an artistic partnership. In some respects the real story behind building both PopCorn experiences has been about establishing PopCorn 66 as a studio. Very much a product of a creative meeting of minds between founders Romain Zitouni and Priska, the partnership has gone from strength to strength during production of both websites.

"The PopCorn Garage project really started from a common desire to collaborate together," Priska explains. " It was about creating something that belongs to us without the involvement of an agency or customer, a fan project made by fans for other fans." Indeed, the whole mission statement for where the pair wants to reside thematically with its work was very much set then and there. Following up with PopCorn TV, the approach remained different to the traditional client and agency paradigm to stay true to the original vision while crystallising the influences so close to their hearts. "The important thing was to strengthen our links with Priska and underline the point of how we complement each other," explained Romain. "So we naturally decided to form the studio PopCorn 66 out of the shared ambition to work on creative projects around pop culture by bringing our expertise together."

PRISKA
Partner and
Creative developer

ROMAIN ZITOUNI
Partner and Art Director

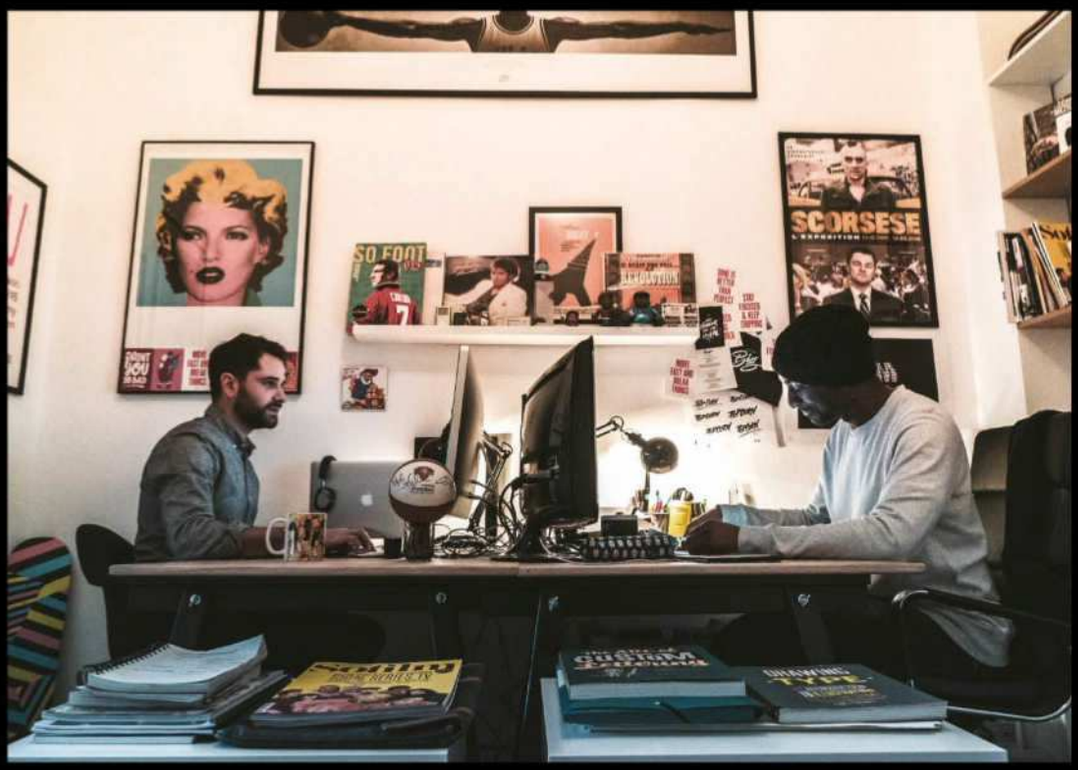
visuals (the game world) is of course 2D. What this meant was that any actions from the player on the visuals such as instigating the zoom or pan effects also proved to be puzzling in terms of development. “I had to review my trigonometry course to accomplish this task,” laughs Priska.

MOBILE MOANS

All in all, the project would end up commanding a lot of assets that would also complicate the back-end side of development. The game ended up including 99 TV references to find, 114 with bonuses, and involving over 700 quiz questions with 300 or so Blind Test sounds. This is where friend Vivien Ripoche stepped in to offer some developer expertise, assisting in the build of a PHP and MySQL backend capable of managing the game data and services being served to the front-end. Within this, Priska recalls that a particular headache was caused by the method used by players to submit their guesses. “One of the biggest coding challenges was creating an input text recognition algorithm. The gaming experience is based on typing in the correct series title, so it was therefore necessary to create an intelligent algorithm that does not penalise players for typos or spelling errors. It would also need to be smart enough not to consider an answer as valid if its spelling was similar to a correct answer within the visual, but still not technically right.” This tricky balance to strike was echoed in the conundrum faced by so many these days for creating ambitious digital experiences suitable for multiple platforms. “We also wanted a site accessible on all devices including mobile,” stresses Romain. “Another big challenge was to create a game available on the maximum number of platforms.” “Since WebGL is not compatible everywhere, we had to create two games in one,” Priska adds. “A ‘full’ version in WebGL with full-screen video and a version without WebGL used especially for tablets and phones, so it’s a lot of work.”

LIVE SCREENING

At the launch of PopCorn Garage, the site very quickly ran into a server problem due to the massive, largely unexpected, traffic. Fast-forward 18 months and PopCorn 66 were determined not to make the same mistakes. “The release of a project like PopCorn TV presents several problems,” concedes Priska. “On launch day the site must actually be able to handle the load, but there are also all the unexpected bugs to manage. So to mitigate this, we created a group on our Facebook page where players could explain any problems encountered. We spent hours after the launch to settle everything once it went live.” Beyond such a thoughtful commitment to addressing such inevitable launch niggles, you really get sense talking to the pair that they cared and continue to care immensely for what they do. After the undeniable successes of PopCorn Garage, both really felt the pressure when it came to unleashing a follow-up because of the natural keenness not to disappoint expectant players. “Four months after going online, the results are super positive,” Romain concludes. “And we’ve had a lot of media relays around the world while picking up Website of the Day on FWA, Awwwards and CSS Design Awards. After appealing to cinema lovers, we’ve managed to drive all TV series lovers around the world similarly crazy, creating a real community through both PopCorn projects. This is what we are most proud of.”



Site Highlight

WITH A FAIR FEW TO CHOOSE FROM, WE QUIZZED POPCORN 66 PARTNERS PRISKA AND ROMAIN TO LOOK CAREFULLY AND PICK OUT A SITE HIGHLIGHT NOT TO BE MISSED

“Even though players have very well received all the features of the game, the Blind Test bonuses (theme music naming) have proven to be the real success on PopCorn TV. They really brought a new dimension to the experience and were much acclaimed on social networks.”

Subscribe today and receive your free gift!

Every issue, delivered straight to your door

Never miss an issue
13 issues a year, and you'll be sure to get every single one

Delivered to your home
Free delivery of every issue, direct to your doorstep

Get the biggest savings
Get your favourite magazine for less by ordering direct

What our readers are saying about us…

"I've been a reader of Web Designer Magazine since the early days."
@sixrevisions via Twitter

"Love the latest magazine that I purchased today"
@navigation_web via Twitter

"My favourite magazine!!!"
@eduardomurillo via Twitter

Pick the subscription that's right for you

MOST FLEXIBLE

Subscribe and save 20%

✔ Automatic renewal – never miss an issue

✔ Pay by Direct Debit

Recurring payment of £34 every six months, saving 20% on the retail price

DIRECT Debit

Instruction to your Bank or Building Society to pay by Direct Debit

Originator's reference 7 6 8 1 9 5

Name of bank

Address of bank

Postcode

Account Name

Sort Code

Account no

Please pay Future Publishing Ltd Direct Debits from the account detailed in this instruction subject to the safeguards assured by the Direct Debit guarantee. I understand that this instruction may remain with Future Publishing Ltd and, if so, details will be passed on electronically to my Bank/Building Society. Banks & Building Societies may not accept Direct Debit instructions for some types of account

Signature

Date

GREAT VALUE

One year subscription

✔ Great offers, available world-wide

✔ One payment, by card or cheque

A simple one-off payment ensures you never miss an issue for one full year. That's 13 issues, direct to your doorstep

☐ **UK** £67.60 (saving 20% on the retail price)

☐ **Europe** €81.55 ☐ **USA** $103.20

☐ **Rest of the world** $103.20

Pay by card or cheque

Pay by Credit or Debit card

☐ VISA Visa ☐ Mastercard ☐ Amex

Card number

Expiry date

Pay by Cheque

I enclose a cheque for £

Made payable to *Future Publishing LTD*

Signature

Date

Your information

Name

Address

Telephone number

Mobile number

Email address

Postcode

☐ Please tick if you want to receive any communications from Future and its group companies containing news, special offers and product information.

Please post this form to

Future Publishing Ltd, 3 Queensbridge, The Lakes, Northampton, NN4 7BF, United Kingdom

Order securely online to receive your free gift

myfavouritemagazines.co.uk/WBDPS18G

Speak to one of our friendly customer service team

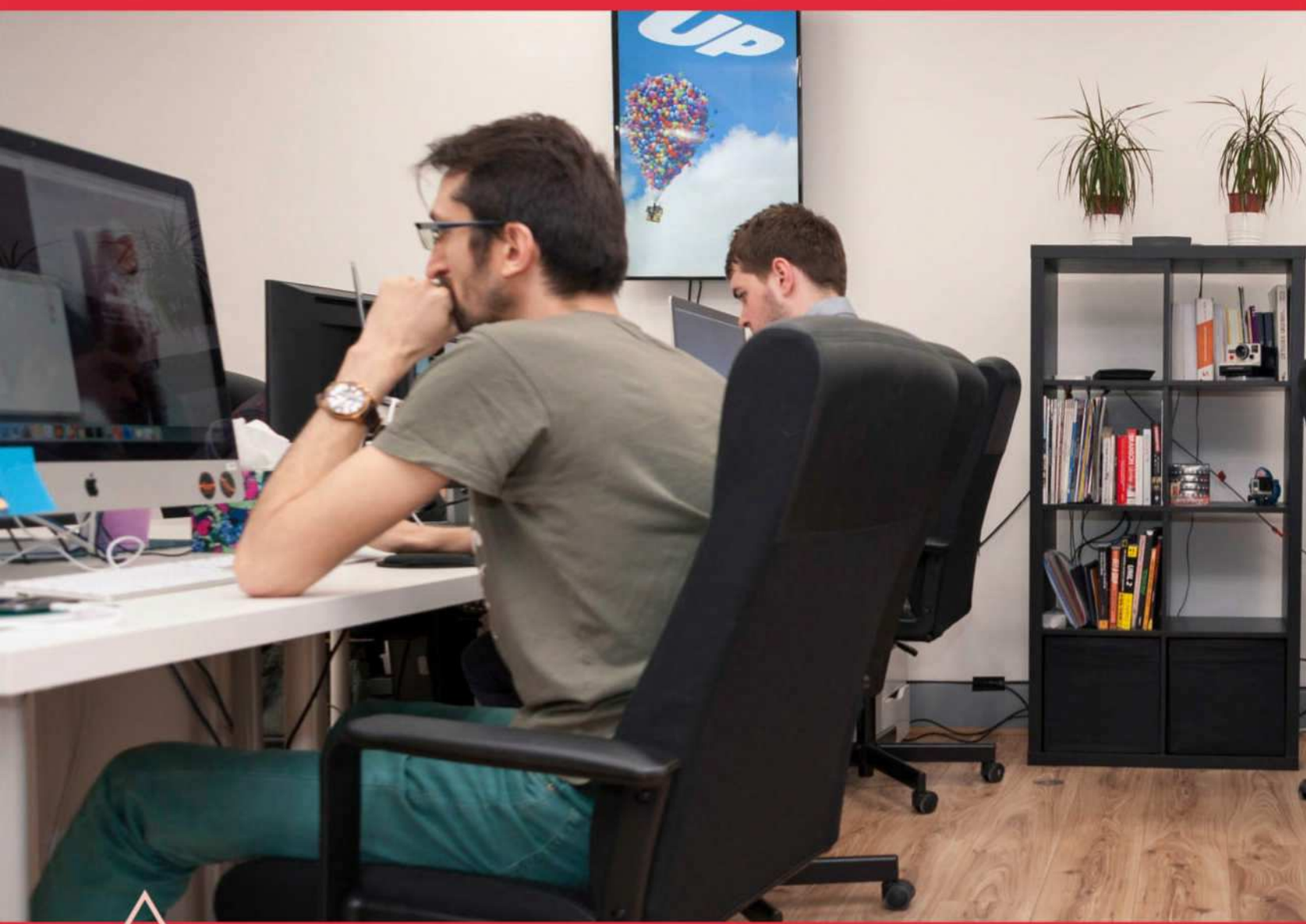
Call 0344 848 2852

This offer will expire on **10 June 2018**

*Terms and conditions: This offer entitles new UK Direct Debit subscribers to pay just £34 every 6 months plus receive a bookazine worth £12.99. Gift is only available for new UK subscribers. Gift is subject to availability. Please allow up to 60 days for the delivery of your gift. In the event of stocks being exhausted we reserve the right to replace with items of similar value. Prices and savings quoted are compared to buying full-priced print issues. You will receive 13 issues in a year. Your subscription is for the minimum term specified and will expire at the end of the current term. You can write to us or call us to cancel your subscription within 14 days of purchase. Payment is non-refundable after the 14 day cancellation period unless exceptional circumstances apply. Your statutory rights are not affected. Prices correct at point of print and subject to change. . UK calls will cost the same as other standard fixed line numbers (starting 01 or 02) or are included as part of any inclusive or free minutes allowances (if offered by your phone tariff). For full terms and conditions please visit: www.bit.ly/magterms. Offer ends 10.06.18

Discovering Design

FX Digital describe themselves as explorers. Creating new digital experiences that redefine the spaces we all inhabit, their ability to craft engaging and exciting sensations has ensured they continue to gain the attention of leading brands looking to connect with their audiences



Who **FX Digital**

What **Website Design, Graphic Design, WordPress, eCommerce, Video, Web Animation, Web Development, SEO, Photoshop, WebGL, WebVR, three.js, and A-frame**

Where **91 - 95 Brick Lane, London, E1 6QL**

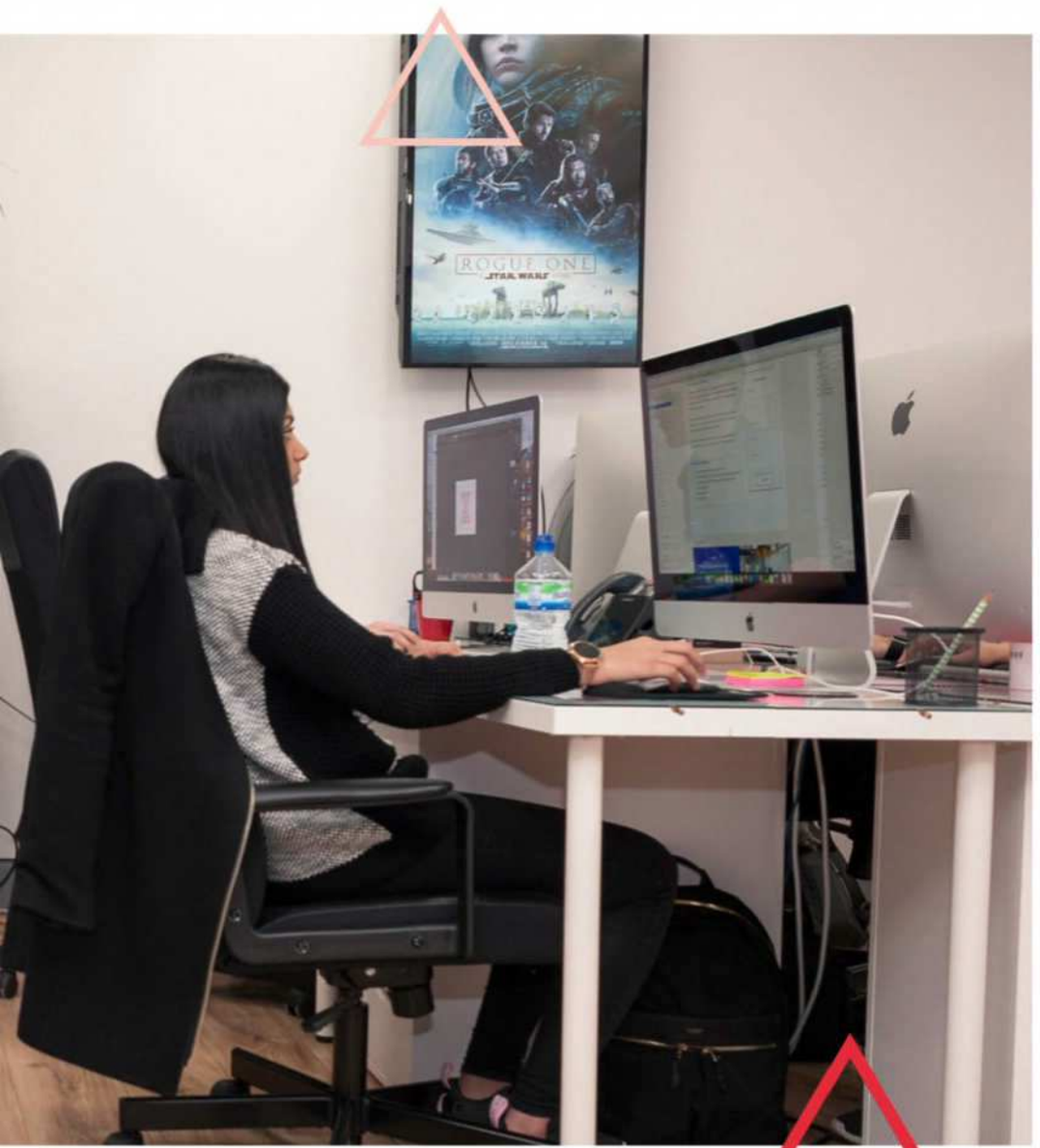
Web **fxdigital.uk**

Key Clients
Double Negative
Prime Resi
Haviland Productions
Vikarious Productions
London Siggraph

Beginning as a hobby in 2010, co-founder Matthew Duhig was at The University of Southampton when his sister asked him if he could create a website for her new bridal business. "Although confident that I could build the site my design ability was a bit limited, so I asked my school friend Tom (Smith) to design a logo and layout, Matt explains. "He was, at the time, studying Computer Animation at Portsmouth and gladly accepted the challenge."

"After completing the design and build my sister started recommending us to businesses that she would meet on her travels. It wasn't long before we had our second gig. We were delighted to be given the opportunity to develop a site for another local bridal company and were going to be paid £300! Anticipating at this early stage that we were going to rule the world, we decide to incorporate a company."

Both Matt and Tom continued to build websites as a side gig while they finished their studies at university. All this took place in local cafés and Matt's family home, kept company by their dogs.



"After graduation we were both excited about the possibility of pursuing our new-found passion on the web and wanted to continue to work together, but we agreed we could benefit from some real-world experience before starting on our agency journey," Matt explains. "Tom successfully interviewed for a job at Double Negative and started as a runner looking to follow a career as a VFX artist, while I landed a job developing email templates at a digital marketing company called eCircle.

"I had an incredible time at eCircle, as did Tom at Double Negative. We both learned a lot from a ton of inspirational people, but we just couldn't shake the urge to pursue the agency idea. So, we both left those jobs and – after a brief period in which I worked at a systems company and Tom cycled from Norway to Ghana – we started taking the business a lot more seriously. Tom and I moved in together in Reading to help concentrate our efforts.

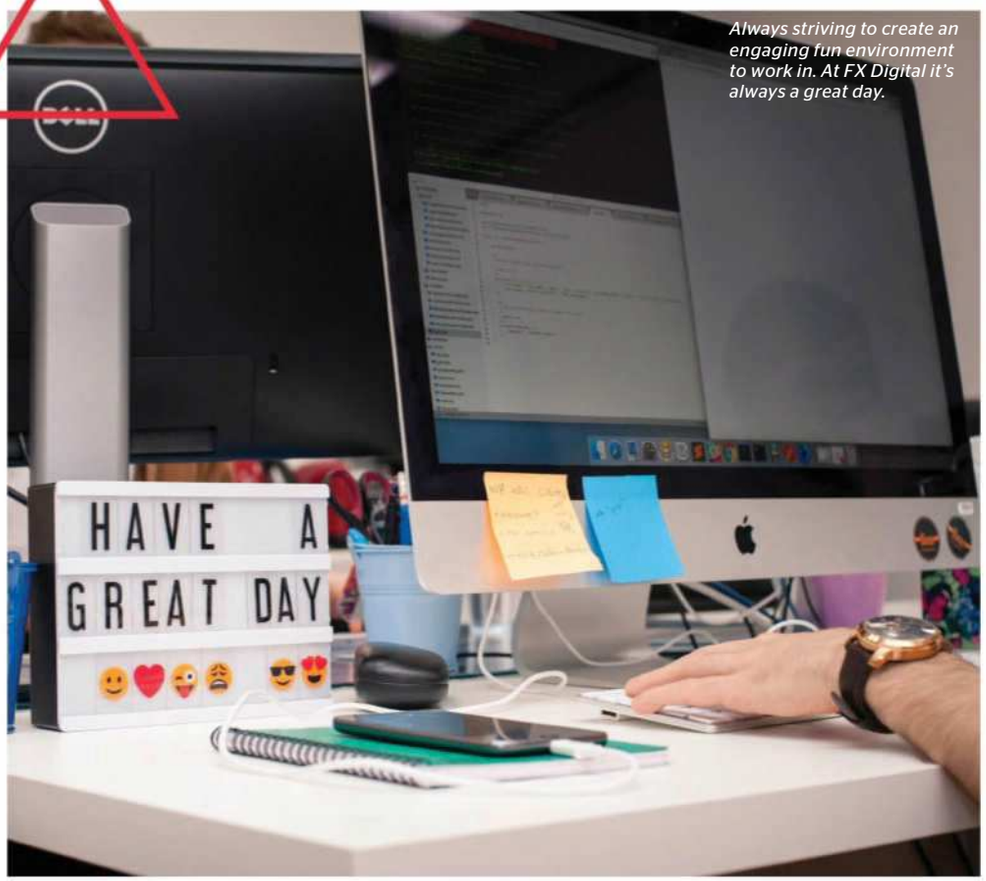
"The growth began and continued quite organically," Matt says. "From a bedroom in Reading we made our way over to East London and finally settled on the Truman Brewery on Brick Lane as a place to base the business. We're still in that same building today, just in a much larger office space than when we started."

FX Digital as a name for their company came later, initially calling themselves Juicy Webs. Matt explains: "We started things up as Juicy Webs, a name originally chosen by Tom and I as two excited youngsters over a beer in the local pub. For some unknown reason we were pretty passionate about the name, which over time had become more of a hindrance than a help.

"The name really did have to change if we wanted to be taken more seriously as an agency. Cue frantic name finding. It's perhaps one of the most difficult processes we've ever been through as a business. To make matters worse, we had been operating for three years before opting to change the name. We already knew how we wanted people to feel when they saw what we were called, and what we wanted it to represent, but we couldn't just pick a word that excited us.

"Being the creative one in the pair, Tom took the lead here. We listed hundreds of words and passed each one around some friends for feedback, but none stuck. Eventually Tom locked himself away and started coming up with ideas for a new visual identity. Before long he settled on the idea of FX Digital. It suited our vision perfectly; combining the worlds of VFX and emerging web technologies. It also had a rather elusive vibe, not limiting us to one medium in the same way Juicy Webs did. Inspired by the Penrose triangle the logo was created as an impossible shape, representing the effect we wanted to develop in our work using immersive technologies."

Matt continues: "Finding the domain was easy enough. We went 'cliché agency' with it and stuck



Always striving to create an engaging fun environment to work in. At FX Digital it's always a great day.

'weare' in front of 'fx' to give us 'wearefx.co.uk'. Before too long we were approached by the owner of 'fxdigital.uk' and opted to purchase this to use as our primary domain. We've found that it's important to include your company name in the URL. It's how people identify and recognise you."

Having spent what seemed like an eternity coming up with a new name for their agency, the next step was for the pair was their own website. Tom explains their approach: "An agency's site is invaluable. Any business's site is really. It's an opportunity to showcase your identity and your skills, but above all, it should be at the forefront of your marketing activities – driving leads and acting as a resource to help land opportunities. It's essentially your CV.

"We've been through several iterations with our site and it's in a good place at the moment. However, the quality of the work we produce and what the team here is capable of, improves so often that our site always looks dated to us.

"It takes a lot of effort to improve your own marketing material as an agency, and you have to be really disciplined to make it work. We try to treat it like any other project and commit internal resource to delivering it. However, more often than not we find ourselves making most of the improvements over the Christmas break when there's little or no client work due."

The skills that Matt and Tom brought to their business has meant a steady stream of clients. Now well established, have they changed how they attract new potential work? "Despite the importance of having a site that drives leads through the business, ours generates us a relatively small number of enquiries and always has done," says Matt. "As disappointing as this is, it's not really ever been a problem for us as each and every lead we have ever had has come through a single source: word of mouth."

Tom also comments: "Word of mouth is essential for the growth of any agency. We're firm believers that by placing the utmost importance on the delivery of work we're giving ourselves the best opportunity possible to enhance this. We can analyse our clients today and, more often than not, track them back to one or two entities or individuals that had faith in us when we first started.

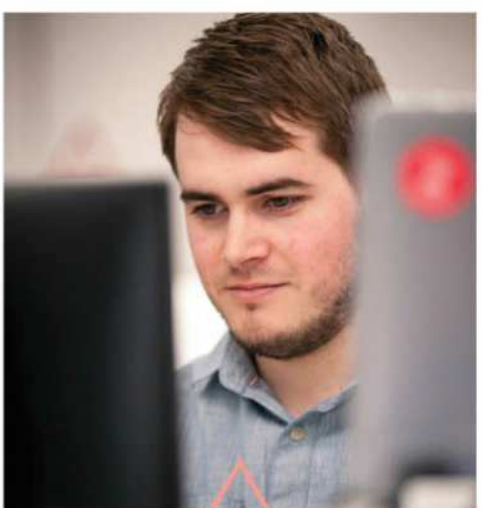
"Despite this, we still insist on pitching for each piece of work that we win and have beaten many a Goliath to a job because of our performance when pitching. Pitching not only gives an agency the opportunity to prove just how talented a group of people they are, but also provides an opportunity for them to thoroughly understand a client, their audience, and their proposition before they start to work with them."

How agencies pick the clients they work with can be manifold. For FX Digital it's always a personal connection with the client or the experiences they want creating, as Matt explains: "For the most part we select the work we do based on the people we meet. Luckily for us – because most of our business comes through word of mouth – most of the opportunities we find involve working with some remarkable and like-minded people. We like to think that we have a pretty unique culture here, and operate on a mentality of achieving more and valuing staff input. We look for clients that match that outlook."

Matt goes on to outline the relationship they always try to foster with the businesses they work with: "It's important to have a productive working relationship with your clients. You need to respect their business and they reciprocate by recognising the skills you bring to the table.

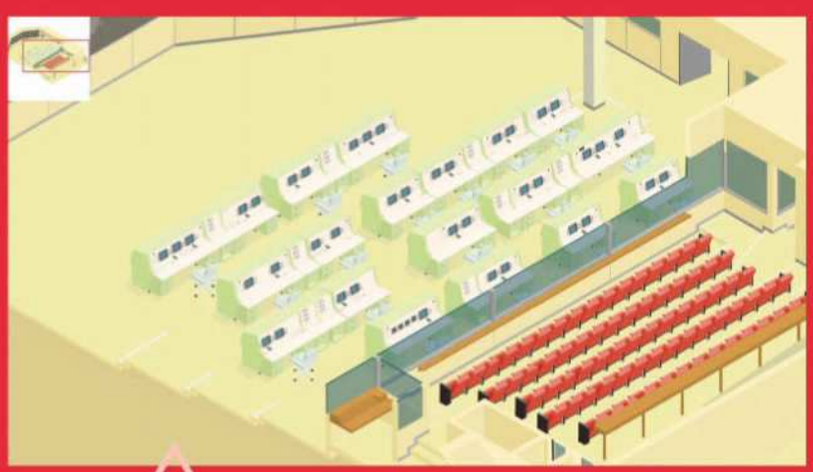
"That sort of partnership – and all our projects are partnerships – allows us to avoid those situations where members of the team are stuck in a meeting or on a call for half a day only to make zero progress. Everyone understands their input, their value, and where we're going with what's in front of us – including the client. We're in the fortunate position to have enough experience behind us to choose what we pitch for. And where we spot a gap in our knowledge, we leverage our R&D team to fill it by creating our own self-initiated work.

"Early into the development of the agency our second hire, Joy Chen (Head of R&D), is an epic creative and technical genius who became our full-time head of R&D. We wanted to experiment with immersive technologies like WebGL, but struggled to focus on this on top of fee-generating



"Not only is R&D a great collaborative process, but the projects we create through it give us a unique edge when talking with clients and allow us to demonstrate our ability to jump into new technologies and create something genuinely new"

Matthew Duhig, Director and Co-founder



Mission Control

missioncontrol.movie

An immersive website to support the documentary film release of 'Mission Control: The Unsung Heroes of Apollo', made by UK-based production company Haviland Digital.

The objectives were to grow Mission Control's audience by targeting space fanatics, build up anticipation for the film release, drive users to purchase merchandise and the film on the iTunes Store, create a rich multimedia experience and to give visitors a flavour of the film.

Traffic to the merchandise site has dramatically increased since the launch and the site continues to drive traffic to iTunes, supporting 'Mission Control' in reaching its #2 spot on the US iTunes documentary charts.

Top *Archive footage was used as background video through the site to evoke a sense of wonder and nostalgia*

Middle *The Astronauts behind the Apollo missions are presented on designed pages that reinforce the theme of the site*

Bottom *WebGL interactive content was created to enable visitors to explore mission control*

Census One

fxdigital.uk/case-study/census-one

Built as an interactive experience to celebrate the release of 'A Star Wars Story: Rogue One' the internal R&D piece had the brief to create a piece of interactive online content to celebrate the latest Star Wars film release.

The experience is a 3D model of the word 'ONE' in a space theme that evokes a real cinematic emotion from the audience. Content is aggregated from Twitter and filtered by the hashtag #RogueOne, with each particle of the model representing a tweet. The audience can rotate and zoom around the model. The experience is paired with an audio track to help to further elicit a cinematic emotion.

The result of this project is a stunning interactive visual census of what fans have to say about the film, as well as showcasing what's possible in today's browser. Not only is the final piece highly engaging and interactive, it also provides a way of visualising a large set of data in an immersive environment.

Top *A WebGL Particle effect was used with each individual component a tweet. Social media never looked so good!*

Middle *The evocative graphics used throughout this experiment in data visualisation, aggregates content from Twitter to deliver an immersive experience.*

Bottom *FX Design flexed their design muscles with this R&D experiment. Each particle of the model represents a tweet #RogueOne.*

work. As soon as we had enough capital we hired Joy to give us a full-time resource to work on internal projects.

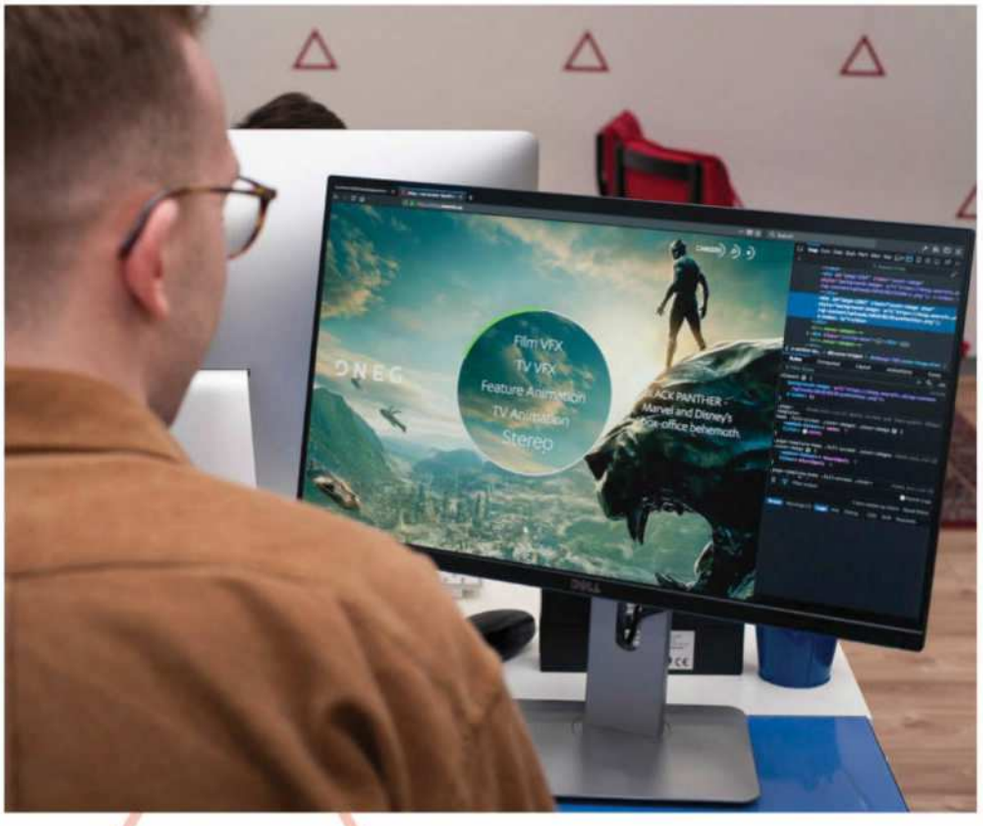
"Not only is R&D a great collaborative process, but the projects we create through it gives us a unique edge when talking with clients and allow us to demonstrate our ability to jump into new technologies and create something genuinely new. And because we make our own briefs the projects are pretty out there, from a WebVR rollercoaster space race game, 3D Globe of WWII events, through time and particle effects galore, if we want to win new work in a particular field we start with R&D as a proof of concept, which helps us choose the next round of clients."

The work that an agency produces not only meets its clients' brief, but also speaks volumes about the agency's core drivers and the ethos that is embedded in the business. Matt outlines which work best reflects who FX Digital are: "There are a lot of projects that we're proud of. If you asked each member of the FX team you'd get a different response. Some would say PrimeResi, a project that saw us produce some insanely intricate WordPress development. Others would suggest Mission Control, a 2D WebGL site that uses PixiJS to present an interactive schematic of the Mission Control room from the Apollo missions.

"For me the project that best represents what we do is Census One. Census One is a WebGL site created using the ThreeJS library. It presents a 3D particle model of the word 'ONE' and individually maps tweets with the hashtag #RogueOne to these particles by pulling them into the site using the Twitter API. The site itself was a product of R&D and something we developed internally to showcase what's possible with 3D in the browser. Not only did it require us to use our skills in design to concept an illusive space environment, it also required us to develop with modern web technologies including WebGL, JavaScript, and REST APIs, alongside optimising and exporting 3D assets into browser-friendly formats such as JSON."

> We're always experimenting, and our arsenal regularly updates to incorporate new additions

Technology is constantly on the move, none more so than in the creative environment of the web. Tom explains which tools FX favour at the moment: "We're always experimenting, and our arsenal regularly updates to incorporate new additions. At the moment, the design team uses Sketch for wireframing and designing digital products and Photoshop for image touch-up and manipulation. We use InVision for prototyping and early user testing, and we are keeping our eye on InVision Studio and excited to see how this could evolve our current design workflow.



Timeline

2011
FX Digital incorporates as Juicy Webs and begin operating from Matt's family home.
Employees: 2

2013
Outgrowing the family home, Juicy Webs moves into a small office in the Truman Brewery.
Employees: 2

2014
Juicy Webs change its name to FX Digital and welcomes Ben and Joy on as its first employees.
Employees: 4

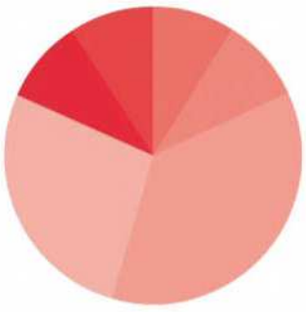
2015
FX Digital outgrows its original office and move to a bigger space in the Truman Brewery.
Employees: 5

2016
Mission Control and Census One websites launched with WebGL experiences.
Employees: 5

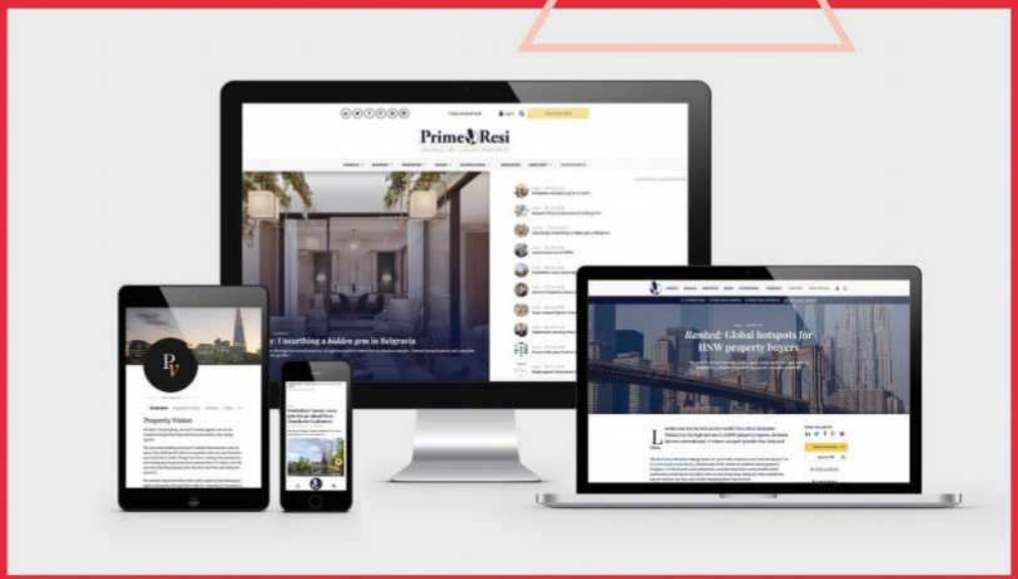
2017
FX Digital outgrows its office yet again and moves to a bigger space in the Truman Brewery. Hugely successful launch of PrimeResi membership website.
Employees: 7

2018
FX Digital moves into the fast lane winning a key account.
Employees: 11

Agency breakdown



- 1 Technical Director
- 1 Creative Director
- 1 Research and Development
- 1 Project Manager
- 3 Designers
- 4 Developers



Left *It was vital that the redesign of the Prime Resi site looked equally striking on every device it would be accessed with*

Bottom *Well-chosen complementary fonts and a colour and image palette delivers a new experience for subscribers*

PrimeResi *primeresi.com*

A subscription-based website for the UK's leading luxury property journal. Designed by the founders, the original PrimeResi website used WordPress, a theme, and a staggering number of plugins, with the subscription functionality handled by an old, outdated plugin.

Although it had served them well during the business's early stages, the PrimeResi site had now grown to over 400 subscribers, with the unfortunate consequence being that the performance and user experience of this original site were no longer delivering what a paying subscriber would expect.

FX designed and built a new site, offering a unique subscriber experience with a vast set of new features. Improvements to the site have increased gross subscriber revenue by 128.8 per cent in the month following launch. New subscribers were also up 266 per cent, while average revenue per subscriber jumped 75.3 per cent.

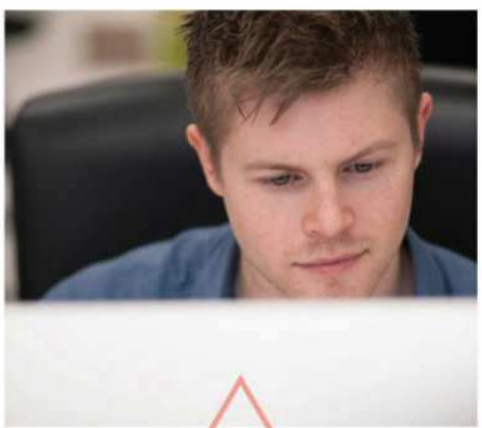
"From a development perspective we use Avocode to reference Sketch files. Avocode gives us the ability to pull assets from a file and even gives us suggestions on styling elements as per the design. The development team uses Git version control with a Git Flow approach, and we use Tower as our Git client because it supports this approach perfectly.

"We keep all our code in repositories on Bitbucket, using their pipelines feature for CI/CD. When it comes to coding, we use Sublime with a set of packages to extend it to support our workflow. Beyond this, each member of the team will theme and adjust Sublime to support their needs. We use Gulp as a task runner amongst other things to compile our SCSS into CSS, and also to lint our code to SCSS-Lint and JSHint standards.

"To bring it all together our project managers work from Asana, co-ordinating the team by creating projects, while each team member tracks time by making use of the Everhour integration. Post-delivery we use Done Done as a ticketing system to give our clients a point of contact for support requests."

Matt outlines how the creative tools FX uses have enabled more ground-breaking work to be created: "Technologies such as WebGL are enhancing JavaScript to bring improved processing capabilities. This allows us to provide experiences that were once only possible on high-end platforms to the web and even mobile. What excites us the most, however, is the improving accessibility of these technologies.

"For example, with a mobile phone and a Google Cardboard it's now possible to experience Virtual Reality through a browser (WebVR), and libraries such as A-Frame are making it even easier for developers to create content in this form. Not far behind WebVR is WebAR (Augmented Reality) and we're anticipating that to be a huge area for development over the next year or so, with the more innovative retailers creating their first WebAR experiences."



"Word of mouth is essential for the growth of any agency. We're firm believers that by placing the utmost importance on the delivery of work we're giving ourselves the best opportunity possible to enhance this. We can analyse our clients today, and more often than not track them back to one or two entities or individuals that had faith in us when we first started"

Tom Smith, Creative Director and Co-founder

FX Digital has grown slowly and organically, adding new members of staff as needed. Matt explains his approach to hiring and what it takes to get a seat at his table: "The people that work with us at FX are the single most important part of our business. We're lucky enough to have developed a friendly and spirited culture that empowers each and every one of us to deliver to the best of our ability and grow along the way. Among everything else it's important to us that the people we employ have absolute passion for the work they do. It makes it so much easier to overcome challenges and it ensures you're always seeking out the most exciting technologies and opportunities.

"We look for those self-motivated learners. In order to work in digital you need to have the willingness to constantly educate yourself using any means you can, whether that is through speaking with others or using resources such as Udemy, Code Academy, popular forums or even Google.

"Next we look for people that understand that they're part of something bigger than themselves. At FX we're a close unit of talented individuals, but the whole is definitely greater than the sum of its parts.

This is most obvious when we're working together to meet a hard deadline or pitching for work across the entire team.

"Lastly, we look for people that don't take themselves too seriously and want to have fun while working. Fun and humour are important parts of what keep us sane over here, and without it we genuinely don't believe the work we produce would be half as good as it is."

"The future is exciting for us," Matt concludes. "We're becoming recognised for the exceptional creative and development work we produce and as such the opportunities we're being exposed to are more interesting than ever.

"Over the past year we've put in place an incredible set of foundations in our creative, development, and support operations that will help ensure we can grow and achieve even greater things in 2018 and beyond. We're always looking to hire and hope to add to our team in the near future.

"We have some pretty great projects that we're looking to release soon. Alongside our first television application we're also working on virtual reality content for the Samsung Gear VR and have an incredible moon landing WebGL experience that's due for release imminently. And who knows what our R&D team will turn their sights on next."

The trajectory that FX Digital is on means a giddy ride for everyone, but is welcomed since it enables the agency to push the boundaries of what is possible, and even to invent the impossible if it doesn't exist.

FX DIGITAL

Founders
Matthew Duhig and Tom Smith

Year Founded
2011

Current Employees
11

Location
London

Services
Interactive Website Design and Development

3D (WebGL) Website Design and Development

TV Application Development

VFX

Virtual Reality

How to design and build for today's emerging experiences

Design for inclusivity

Understand the diverse range of users needs

Sometimes referred to as 'Universal Design', inclusive design considers as many people's needs and abilities as possible, instead of a 'one size fits all' approach to the experience.

As designers it can be easy to unwittingly design for those that are just like ourselves, or prioritise these considerations due to tight budgets or deadlines. As designers we should be aiming to include people with varying ranges of cognitive or physical disability, rather than exclude them. Designers should do this by removing the barriers that create extra effort and separation, enabling the end user of your product or service to have the confidence to participate equally, and without support.

Over the next year, expect to see inclusive and ethical design become an expected part of the UX Design process. Fortunately, there are plenty of other people getting involved in the digital community, with social movements such as The A11Y Project, AXSChat, Tech For Good and Tech For Good Live podcast gathering rapid momentum over the past 12 months. These groups provide a supportive space for designers to learn more about the inclusive design process and the problems that different people face when using technology.

Inclusive design shouldn't be confused with accessible design. Products and services are usually made accessible as an afterthought; for example, a watch might be retrospectively made accessible for blind people by including braille numbering on top of the watch face. This modification to a device designed for those with sight may solve one technical problem, but introduce many more issues for those that are blind. Inclusive design seeks to fundamentally redesign a product from scratch, removing barriers from the start. Inclusive design is proactive, not reactive.

When starting any new project, one of the most important questions UX designers should ask themselves at all stages of the design process is, "Who will this design exclude?".

Design for immersion

Designing user experiences for extra dimensions

Traditionally UX designers had a clear separation of realities to design for: real life, and the experience delivered on screen by the person's device. Now the lines have been well and truly blurred with Augmented Reality (AR) and Virtual Reality (VR) entering into mainstream use. It's not enough to design for screens, pages and offline touchpoints anymore, now the concept of multiple dimensions opens up a plethora of ways to enhance the experience.

A whole host of interactions can be incorporated into designs, such as picking up, pinching, pushing and pulling, facial expressions, and even air tapping for Microsoft's HoloLens. To get them to do this you must also think about the cues you will give users that are used to interacting with flat screens, how will you encourage them to look around in the space? With new immersive technologies you can now use audio to grab attention, or display elements just off screen to prompt them to move left and right. This new technology also gives you the opportunity to play around with objects in a 3D space, so it's important that designers become comfortable in how shadow and light can be used to create the illusion of depth and mass for objects in the interface.

Designers also need to be conscious of the right context to use these interactions. As a user interacting with an Augmented Reality App whilst driving would be entirely inappropriate, and it might be that a voice interaction is more suitable in this type of scenario. Thorough research and testing is required of the UX practitioner to find and understand these contexts and user goals.

Overall, expect the prevalence of AR and VR to increase rapidly over the next few years as businesses and organisations find ways for this technology to fit their business models.

"It's natural to be excited by the possibilities of immersive UX. But we must be careful not to let technology distract us from the human that needs to use it"

Damian Rees
Director and Founder of Experience UX

Tools for VR, AR and inclusivity

A-Frame
aframe.io

A-Frame is a web framework for building Virtual Reality (VR) experiences. Originally developed by Mozilla, it is an independent open source project. A-Frame is HTML, making it simple to get started.

Microsoft HoloLens
microsoft.com/en-gb/hololens

Microsoft's 'mixed-reality' product, HoloLens, is the first self-contained, holographic computer, enabling you to engage with your digital content and interact with holograms in the world around you.

Funkify Disability Simulator
funkify.org

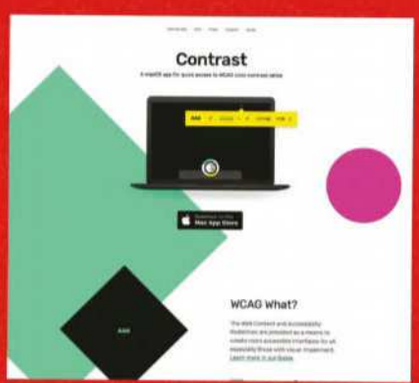
Funkify is an extension for Chrome that helps you experience the web and interfaces through the eyes of users with different abilities and disabilities. Funkify is created by a team of usability and accessibility experts in Sweden.

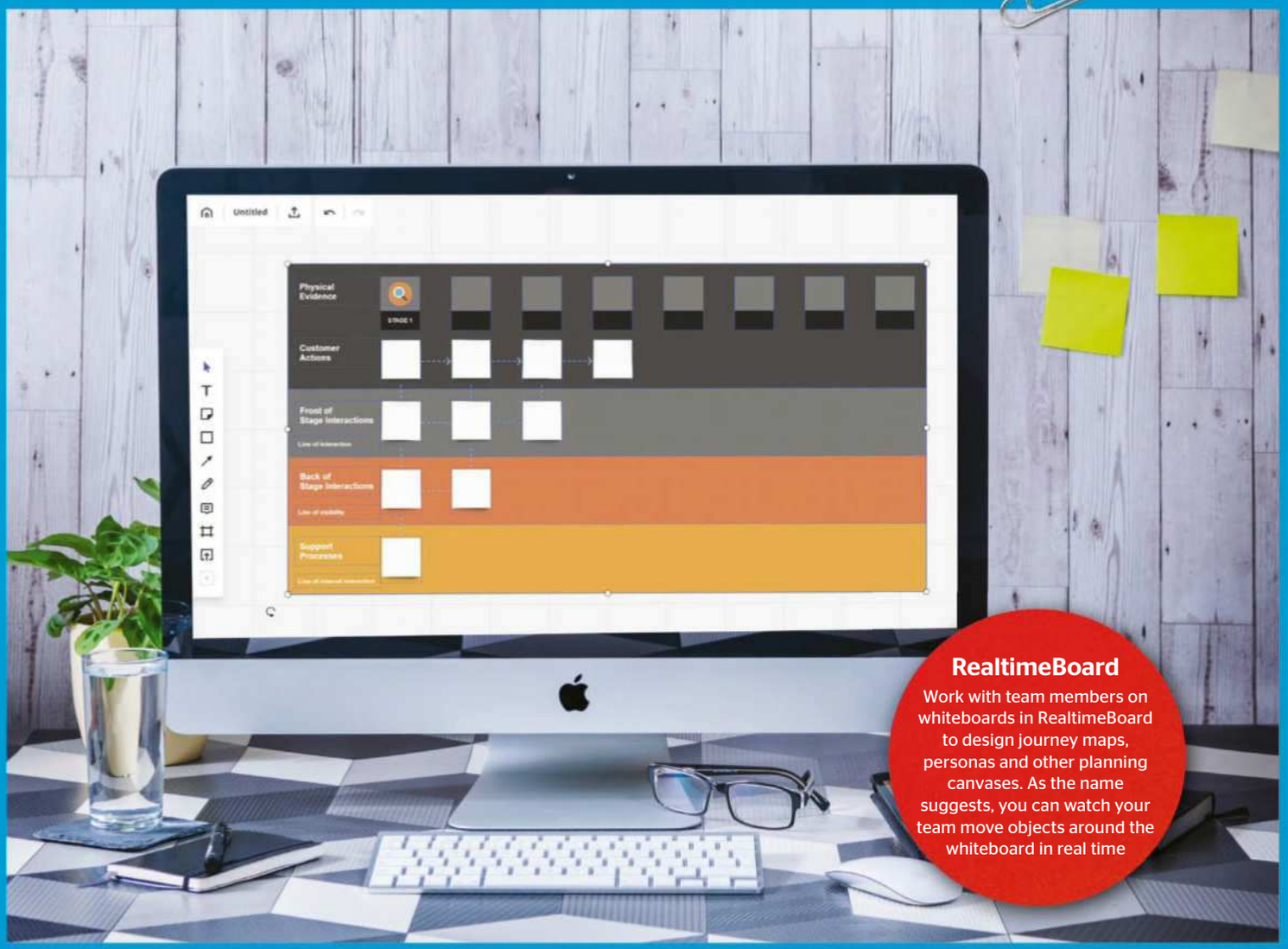
Stark
getstark.co

The colour-blind simulator and contrast checker for Sketch. Simulate the various forms of colour-blindness by quickly previewing your Sketch designs and make adjustments as needed.

UseContrast
usecontrast.com

A macOS app that provides access to WCAG colour contrast ratios. The entire app UI updates instantly when picking colours, making it easy to get the colour contrast information you need to make informed decisions about the colour of your text.





RealtimeBoard
Work with team members on whiteboards in RealtimeBoard to design journey maps, personas and other planning canvases. As the name suggests, you can watch your team move objects around the whiteboard in real time

Design for collaboration

Outstanding teamwork creates awesome products and services

As 'UX Designer' and its many permutations become more ubiquitous as a role, teams are growing and they have a bigger seat at the table. As a result, more business stakeholders are interested in knowing – or even being involved – in what you're doing.

The UX role has now matured, and there are plenty of online communities, tools, conferences, and books aimed specifically for the UX designer. To complete the perfect storm, the digital marketplace is also saturated with multiple offerings for a single type of product, and organisations are more willing to invest the time in creating unique user experiences to make them stand out in a fiercely competitive crowd. Suddenly, UX practitioners find they not only have a voice, but are influential in navigating a product or service to market.

Superior soft skills are the secret weapon behind superior UX teams. This includes communication, listening, empathy, workshop facilitation, teamwork, and storytelling. These provide the foundation that all other deliverables are based on. How do you know if the prototype you are testing will meet a user need if you have not listened properly in the research phase of the project?

These skills are not an innate talent and need to be practised just like any other skill. Not only that, but developing your soft skills as a team enables you all to communicate properly with one another, forming a common strategy so you can all aim for the same goal. There are many good UX practitioners, but great ones have exceptional soft skills to help them do their job.

Collaborating with the customer or client is also essential to a smooth-running project. There are many tools such as Marvel, InVision and Axure that will enable you to quickly prototype up your work to show them the 'Promised Land', instead of sending emails back and forth you can now make your solutions come alive. The benefit of this approach is increased buy-in from clients and customers, and frictionless collaboration.

Some of the biggest obstacles to collaboration on a project can come from other business stakeholders and departments not understanding what UX activities entail. The solution here is to be as transparent and as open as possible. As a team you can pique people's interest by creating exciting areas of wall space in high traffic areas where deliverables such as personas, journey maps and wireframes can be displayed to spark conversations between different people within the business.

Even the rise of remote working and distributed teams is a waning threat to any UX team. There's a tool for every stage of the process, and you don't even need to be in the same room as each other. Project planning and management can be organised through tools such as Slack or Flock or Asana. Visual deliverables can be taken care of using collaborative whiteboards such as RealtimeBoard. Teams can work simultaneously to create fully fledged prototypes using one of the new generation of tools like Figma or InVision.

Design for trust

Creating confident relationships with end users is key

Trust is a human emotion that can be designed for, and can make or break the user's experience, but why is it so hard? Well, there's a lot out there to put off even the most savvy digital user, with dark UX patterns, fake news and clickbait rife. Emerging technologies such as blockchain and self-driving vehicles will put the majority of UX designers' skills to the test.

In recent years trust has shifted from being controlled from the top-down by the business or organisation, to being collectively controlled by users via social media about how trustworthy (or untrustworthy) their experiences with a brand have been. It's fair to say that companies are not in control of this aspect of how they are viewed anymore, and so it's imperative that a brand's actions speak louder than its words. To gain the trust of the user, the experience must become as transparent as possible, with businesses being open about their motives, beliefs and activities. Designers can enable that relationship by not hiding away this information from the user, removing any anxieties they may have.

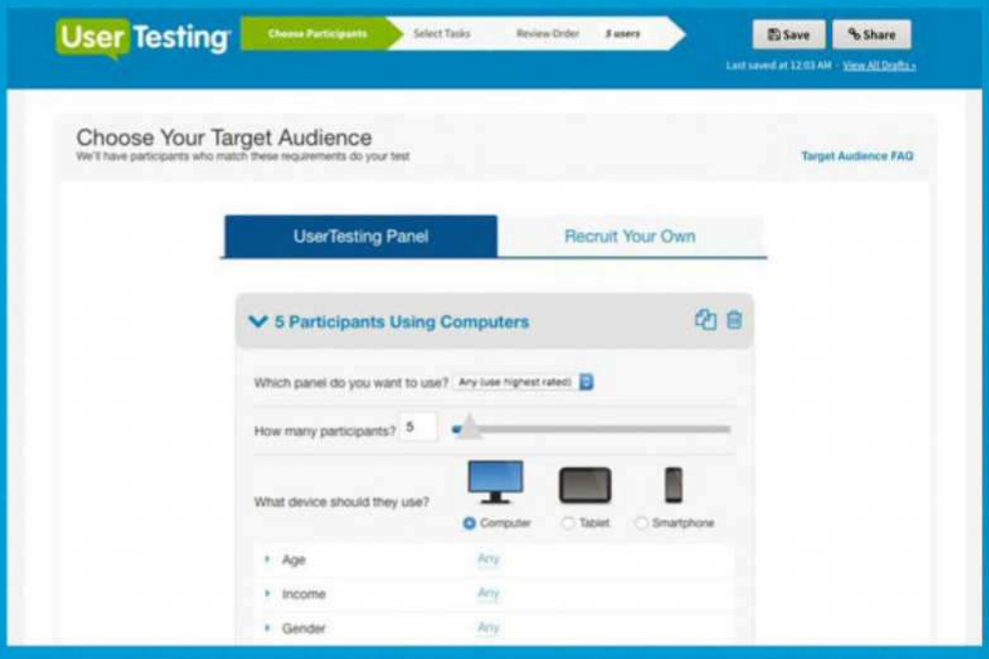
When a customer takes a leap of faith and invests their time, and possibly their money in your product or service, you suddenly have a social responsibility to make good on that relationship. So despite all that, how can trust be designed for? Thankfully there are a few techniques UX designers can use to instil confidence in the end user throughout their journey.

"Trust and transparency is the most valuable, and strongest currency in the digital industry, and it is an essential key ingredient in building successful products and services that customers love"

Mike Langton
Digital Experience Manager, McCarthy and Stone

We all judge a book by its cover, and it's also well known that a user is more likely to trust a site that is more aesthetically pleasing. This is called the aesthetic-usability effect, and is described as us perceiving beautiful things as easier to use over ugly ones (even if that is not the case). Included in the look and feel of the site aesthetic should be the tone of voice and type of imagery that are used to convey a professional, reliable impression of the business or organisation.

Of course, the ultimate indicator of trust should always be in the user testing results, along with observations of the user's reactions to sites. Subjective measures like trust can also be captured at the end of tests. Moderated user testing will always provide much greater insights, but there are tools online to run unmoderated tests such as **UserTesting.com**.



Tools for trust and collaboration

Dark Pattern Library

darkpatterns.org/

Dark Patterns are tricks used in websites and apps that make you buy or sign up for things that you didn't mean to. The purpose of the Dark Pattern Library site is to spread awareness and to document the companies that use such techniques.

Government Research Consent Guidelines

bit.ly/2uhg2fo

The UK government website contains an entire manual on service design and the consent forms you need signed to ensure you can be trusted with a person's data gathered during user research.

Figma

figma.com

Figma is a browser-based design tool that makes it easier for teams to create software. Present and prototype in the same tool as you design. Version control your team designs.

Float Team Schedule

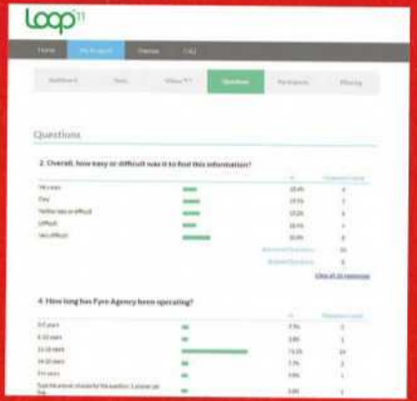
float.com

Float enables you to visualise your team's project assignments, overtime and time off in one place. Collaborate on project plans and resolve conflicts with real-time drag-and-drop scheduling.

Loop11

loop11.com

Loop11 is integrated with **JustInMind.com** and is used to create prototypes that can then be used to run online usability tests, with the results shown in detailed reports and in-test videos.



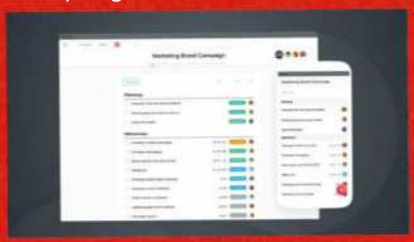
Tools For coherence

Axure
axure.com

Create simple click-through diagrams or highly functional, rich prototypes with conditional logic, dynamic content, animations, math functions, and data-driven interactions. Use Axure Share to upload content to share with your team.

Asana
asana.com

Asana is an online project management tool, designed to help teams track their work. Asana gives you everything you need to stay in sync, hit deadlines, and reach your goals.



MockFlow
mockflow.com

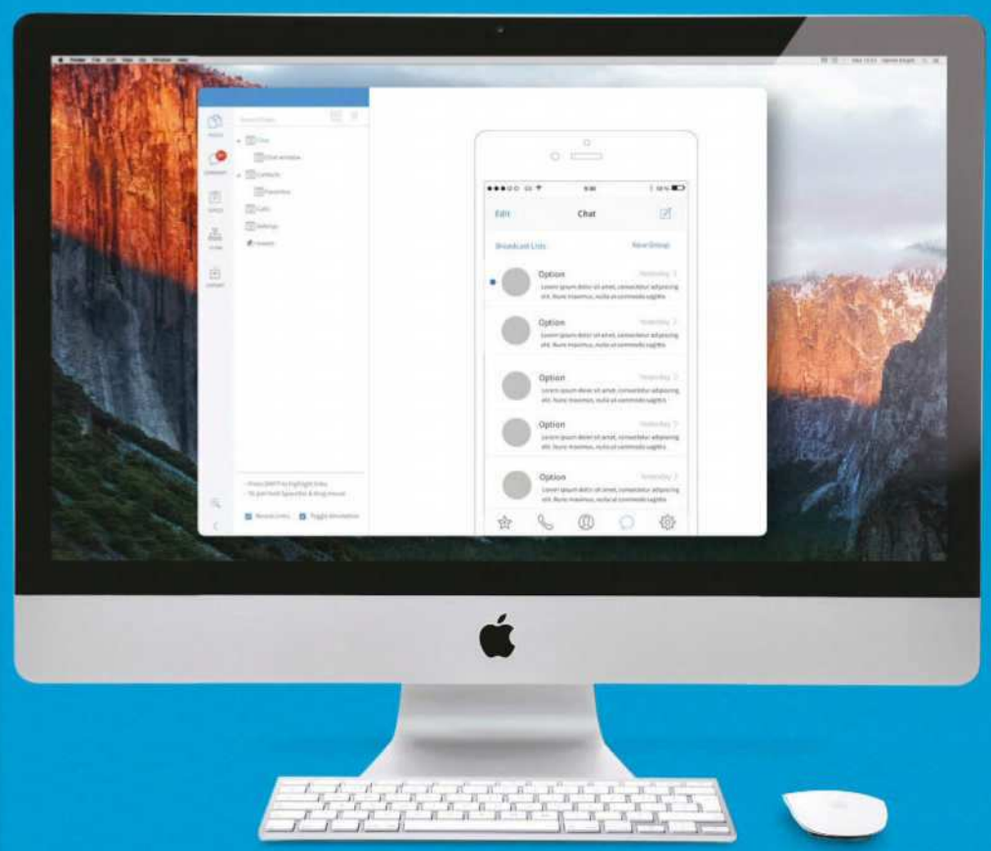
MockFlow provides a full solution for design teams, which includes wireframing, sitemaps, UI spec systems, design workflow and more. Enables you to plan and create better user interfaces together within a single suite.

Storyboard That
storyboardthat.com

Storyboards are a fun and engaging way to relay research findings and user journeys to stakeholders. Use their extensive image library and flexible templates to create storyboards of this information.

Smaply
smaply.com

This website has an online editor which enables you to integrate basic service design tools into your daily work, such as user journey maps, stakeholder maps and personas. Your designs can be downloaded as PDFs and image files.



Design for coherence

The modern user journey is increasingly fragmented

With more and more touchpoints emerging, organisations are in danger of their user's journey becoming so heavily fragmented that it could become an incoherent mess. To add to the omnichannel experience there are now chatbots and other voice interfaces to consider in the user's journey, so the experiences and conversations people have with them need carefully designing.

Planning is key, taking a 'helicopter view' of the entire user's journey with the business. This should include doing as much user research as possible to make sure the touchpoints you design align with their goals, and what they're doing in real life. Turning this research into user journey maps and personas will help guide designers on which touchpoints should be used for different audiences. Many tools exist for supporting these activities; Smaply caters for all of the above, and Xtensio can be used to create simple personas and diagrams, but there are also more traditional offline tools such as Axure that you can use to get the job done.

It's also important to consider which touchpoints *shouldn't* be designed for, especially if it is discovered during the research that it would be inappropriate to use certain methods to contact certain audiences. For example, on a digital experience dealing with a homeless person registering for support services, would it be appropriate to ask for an address?

Designing a coherent experience means not just designing for screens and apps anymore, but every means of contact the customer has with that organisation, so that a unified message can be delivered, regardless of the type of touchpoint. It's imperative that this key message is decided on from the start. The entire UX team should know from research what message to deliver. It's a common belief that the more material you present to the user, the greater chance that some of it will be remembered. It's the old adage of throwing a load of mud in the hope some will stick, but this isn't true. Your audience will end up confused about the message you are trying to deliver.

"My favourite tools are the ones that speed up workflow and take care of the menial tasks we would rather not do, enabling teams to innovate on more interesting tasks"

Laura Yarrow
UX Architect, Experience UX

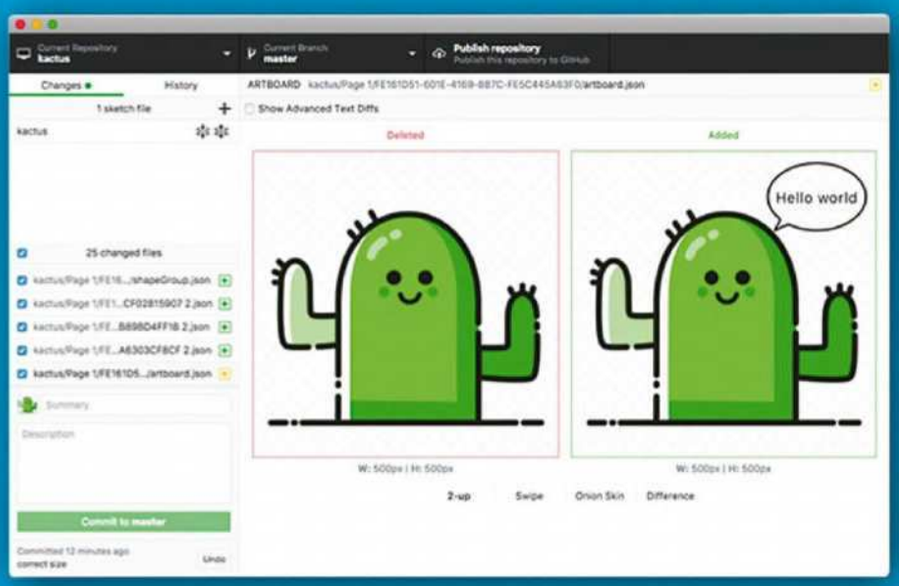
Design for efficiency

Leverage version control for enhanced UX productivity

As UX teams grow, there are smarter ways of managing the multitude of design assets created by a team. No more naming your work 'homepage_wireframe_finalFinal14.pdf', or taking it in turns to work on the same document in your team. Thankfully now there are tools aimed specifically at design teams to version control design work. The majority are based on Git, the same technology used by developers to manage their application code.

There are so many advantages to using this sort of software to manage your designs. Not only can multiple designers work on the same project at the same time, but you can roll back to a previous version if needed. Although you will only see the current version of a file, a full version history is kept and reviewing the changes made between versions of a file are even possible. These features of version control mean problems like losing work when a file is accidentally overwritten, or two people decide to make changes to the same thing are now a thing of the past.

Once changes are made, many tools let you communicate those changes to the team. This is a step forward in terms of productivity and efficiency, enabling projects to be completed as quickly as possible. Lots of the larger web-based design tools like Figma and UXPin provide this as part of the subscription, but there are standalone tools like Kaktus, Abstract and Folio for Mac.



Float tutorial

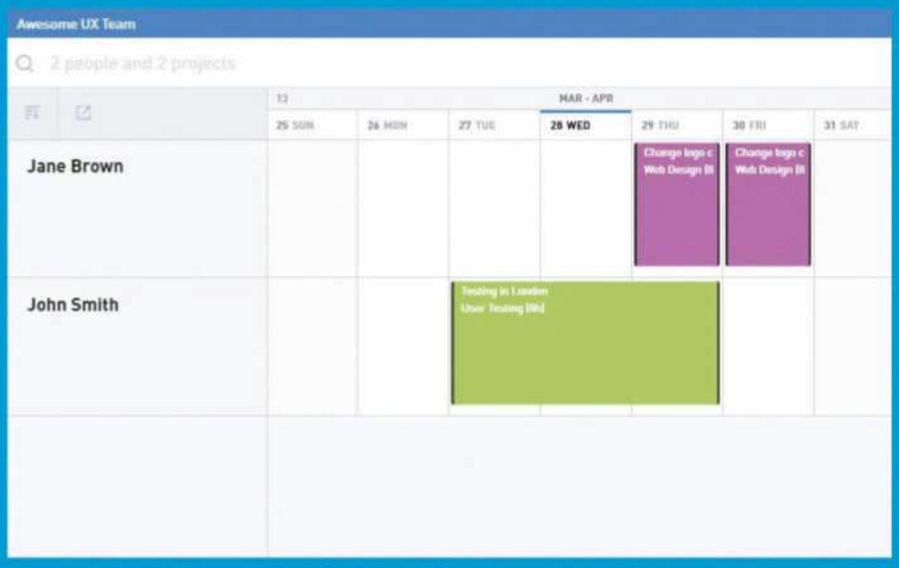
Manage UX teamwork in Float

01 Sign up and create a team

This will be the name of your design team, at the moment you will just see your name in the calendar, and a prompt to 'Add more team members', which you should click to get started adding more people.

02 Add the names of people and projects

Add as many people as you need by entering their full names, and then enter in projects when prompted. You can add extra projects and team members to the shared calendar later if required.

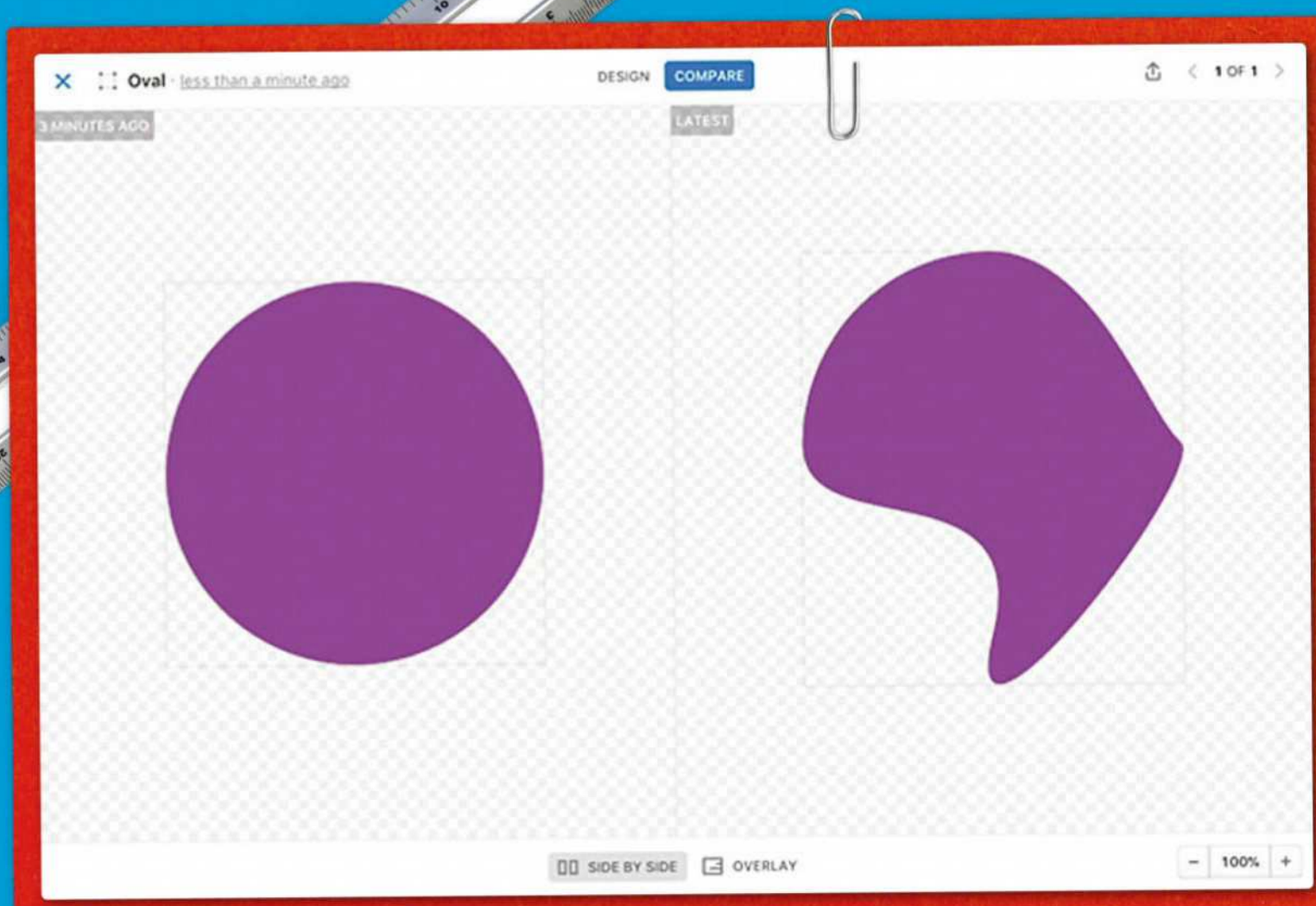


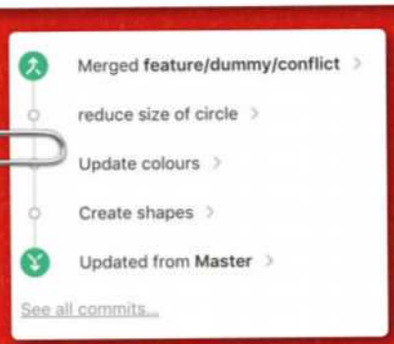
03 Schedule time

Click on a free 'block', and enter a task name and project in the dialog. You can also add a comment and other information to the event. Click Create to add to the calendar.

04 Modify your schedule

Drag to move time blocks around, or hold Ctrl+Shift then click to move a copy to another time slot. Hold Shift then click multiple blocks to select multiple time blocks that can be moved around.





"Amplify the power of Sketch by integrating it with version control tool Abstract. It's easy to see the history of changes made to a Sketch file, who made them, and when"

Abstract tutorial

Get started with version controlling your design work

01 Sign up and download the macOS app

Having a team account will enable you to all work on files related to a project without creating collisions and losing your work. Choose your account type, and name your team if applicable. Download the macOS app from the App Store.

02 Create your first project

Open the macOS app and click 'New Project'. Fill in all the details of the project, giving it a name, description, and whether the project is public or private. Click 'Create Project' to finish. Invite team members to the project.

03 Import or add a Sketch file

Drag-and-drop, or use the file browser dialog to import your first Sketch file. This can take some time, depending on the size of the file. Alternatively create a new Sketch file to version control and start working on.

04 Create a branch

Branching is a concept used in Git. Branch changes do not affect the Master version, until you are ready to merge the work into it. This enables you to roll back the changes without losing the original work in the Master.

05 Work on your Sketch file

Open the file in the branch and make changes. All changes you make can then be compared and tracked against the previous versions. If you decide these changes are no longer needed you can go back to Abstract and remove.

06 Commit your changes

A commit is like a 'super save' with context. When you commit, you write a short description of the edits you made. This way, your team can look at your commit history and know what changes were made, when and why.

07 Click 'Merge branch' to unify branches

Merging brings all the separate edits together in one place. You can merge two branches, bringing those Sketch files together, or merge to the Master. Completing a merge will archive the branch, where it can be reviewed or revived later.

Design for conversation

Understanding the current and future state of voice interfaces

The rise of chatbots and other conversational devices such as Amazon Alexa and Google Home has been all pervasive over the past few years, and many companies are still trying to work out where this new technology can be inserted into their strategy with customers. But where does traditional experience design fit in, especially when there will be no physical interface to design? This is a new frontier for service design, with endless possibilities for designing intuitive and human-centred experiences that people love.

Conversations between human beings are intricate, complex and heavily nuanced. Not to mention the cultural and semantic differences that are commonly observed in humans across the world. How do you anticipate and plan for the vast array of possible questions and reactions a human being might have? Designers will need to spend time designing all the possible flows and outcomes these conversations might take. And the more human the experience can be the better, but how can you make a machine appear human? How do you build a relationship with a machine? These are questions the UX designer must consider to create an effective outcome for the end user.

"The key to the adoption of conversational interfaces is creating services that genuinely solve a problem and add value to everyday life; or the novelty will soon wear off"

Jenny Lewis
UX Architect at Experience UX

Understanding the context that your designs will be used in is also important, so rigorous and in-depth research is essential. Would your target audience use a voice interface walking down the street? Would it be usable if it was a noisy street? All this can be answered by spending time understanding your users and capturing what their goals are.

Another essential part of the UX practitioner's role will be in planning for and testing these conversational interfaces. This will be very different to traditional testing of apps and sites, and will require much more rigorous planning of scripts and testing sessions. There are a few tools for designing the proposed chatbot conversations and also the UI, such as BotPreview and Botsociety, which then enable you to go and test these conversations out on real people before you release your chatbot or conversational UI. As a result of this frenzied focus given to this emerging technology, expect to see new roles created as offshoots of the standard UX Designer and – relatively new – UX Writer titles, such as 'Conversational Designer' (catering for research, testing, behaviours and personality of the interface) and 'Conversational Strategist' (a niche role dedicated to designing the flows and logic of the conversations).

Top tools for conversational design

BotPreview
botpreview.com

Sketch and design your own chatbot interactions using the BotPreview online editor and share them or export as static HTML or MP4/GIF video, without writing a single line of code.

Botsociety
botsociety.io

Design voice and chat interfaces using the online web editor by quickly building a high-fidelity preview of your next chatbot or voice assistant. Botsociety takes care of the appearance, the platform limitations, the preview, the export and the user testing for you.

Botmock
botmock.com

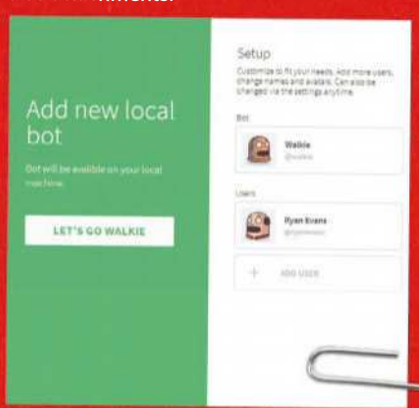
Botmock uses a drag-and-drop editor with templates to build prototypes of conversational design. Map out the customer's journey, and create a live preview that can be exported to GIF and video.

Bots UI Kit for Sketch
bots.mockuuups.com

A simple and fully customisable Sketch UI kit to help you design and showcase your Facebook Messenger Bots. All elements are turned into new branded Sketch symbols, so prototyping has never been easier.

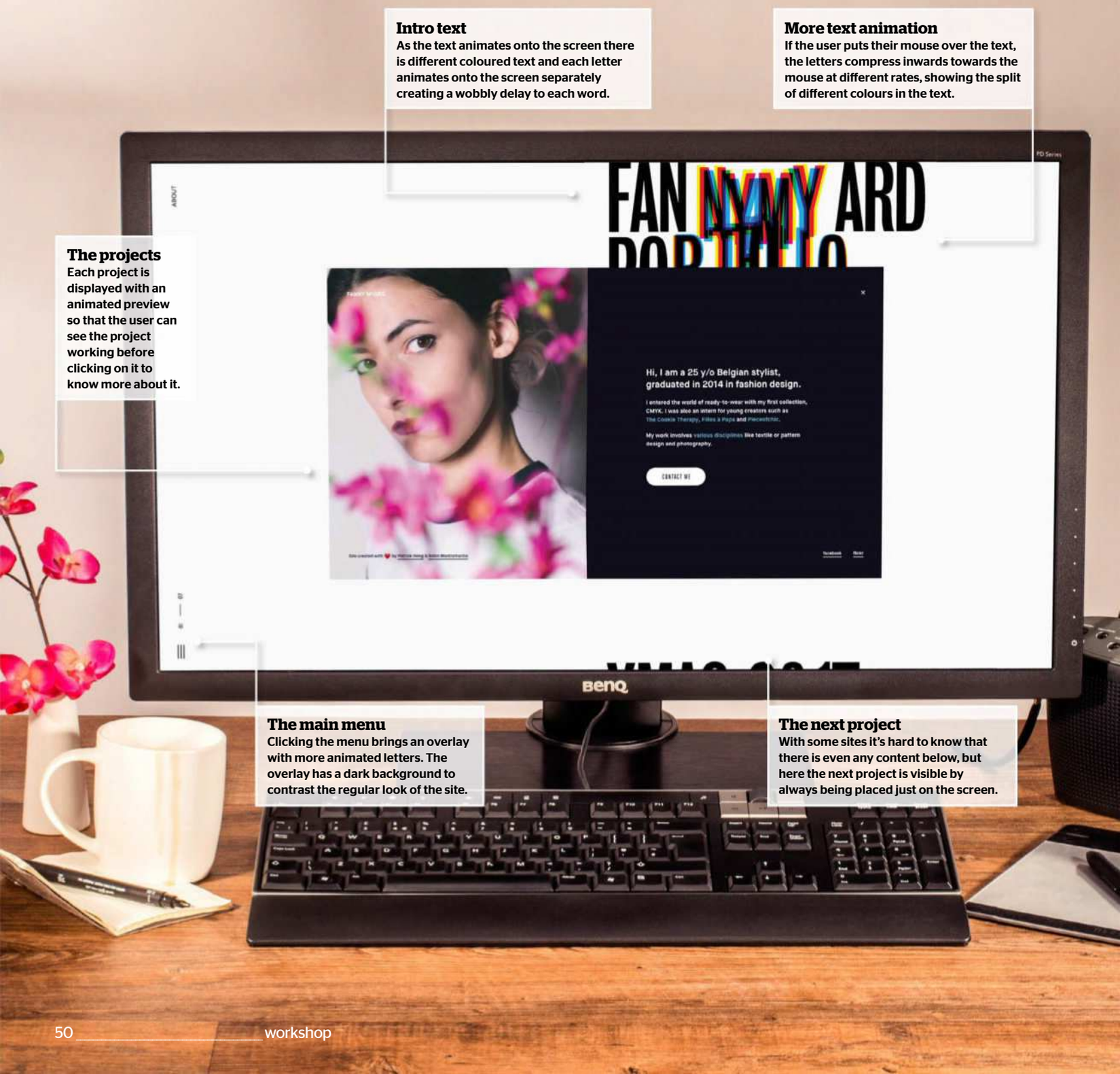
Walkie
walkiebot.co

This tool is especially for Slack users to help design slack bot dialogues. It provides an easy way to write and test bot dialogues, which include buttons and also attachments.



Create a wobbly text effect with JavaScript

Inspired by **https://patrickheng.com**



Create a wobbly text effect with JavaScript

DOWNLOAD TUTORIAL FILES www.filesilo.co.uk/webdesigner

EXPERT ADVICE
Inspiration for CMY
The inspiration for the text having different colours seems to come from the world of print. When three inks Cyan, Magenta and Yellow are overlapped they produced an almost black colour. This site uses a similar effect with colours close to those and a blending mode of 'multiply' to achieve the result.

<comment>
What our experts think of the site

Experiments in typography

For my portfolio, we wanted to animate typography in an original way. Nicolas Loureiro (the UI designer) came with the idea to separate the colour layers of the text on the site. Then we did a lot of testing with After Effects and in code to find the right effect and to find fun interactions around this.

Patrick Heng, creative developer at Merci-Michel and Gobelins Paris student

Technique

1. Creating the intro text

Open the project files and you will see that there is the velocity.js and blast.js library. Blast splits text up, while velocity is a tween engine. There are three offset variables to make each piece of text move at different intervals.

```
<script>
var offset1 = 0;
var offset2 = 0;
var offset3 = 0;
</script>
```

2. Splitting the text

Using blast, each of the text characters are split into their own span element. Then the top layer of text is looped through and the offset is increased for each letter so they animate independently.

```
$("h1").blast({
delimiter: "character"
});
function anim() {
var $spans = $("#top").find('span');
$spans.each(function() {
offset1 += 40;
```

3. Adding velocity

Now the velocity library adds the movement and opacity so that the letters move up and fade in. Each letter is offset, and the duration and easing are set for each of the letters.

```
$(this).velocity({
translateY: -100,
opacity: 1
}, {
delay: offset1,
duration: 800,
easing: "easeOutBack"
});
});
}
```

4. Calling the action

Now the 'anim' function is called and this triggers the animation to begin. A 'setTimeout' function now staggers the second block of text that will be yellow. Again, it is triggered using velocity as in the first example.

```
anim();
setTimeout(function() {
var $spans = $("#middle").find('span');
$spans.each(function() {
offset2 += 40;
$(this).velocity({
translateY: -100,
opacity: 0.8
}, {
```

5. Moving on

The delay, duration and easing are set so that the second yellow text moves correctly. Then the last piece of text which is orange is controlled in the next 'setTimeout' function to delay this moving a little longer before starting.

```
delay: offset2,
duration: 800,
easing: "easeOutBack"
});
});
}, 100);
setTimeout(function() {
var $spans = $("#bottom").find('span');
```

6. Final letters

Now the final letters are moved into place. This gives the same effect as Patrick Heng's site, which has layers of text moving. Patrick actually uses WebGL to move the text but this is a simpler way with DOM elements.

```
$spans.each(function() {
offset3 += 40;
$(this).velocity({
translateY: -100,
opacity: 0.8
}, {
delay: offset3,
duration: 800,
easing: "easeOutBack"
});
});
}, 150);
```

Create a fun face detection photo app

Combine the power of facial recognition and a web cam to build a photo booth-style app using clmtracker and the p5.js library

DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/webdesigner

With the rise of social applications, a big part of their popularity has been their built-in photo manipulation tools. Instagram has become synonymous with filters and more recently social apps have capitalised on the rise of Augmented Reality to recognise faces in images and to overlay props on various facial features. This has been hugely popular among users and a contributing factor in the mobile success of certain apps. In this tutorial, an app for creating photos with face detection will be created running in the web browser. The facial recognition is added using the clmtracker (constrained local models tracker) library and then props will be added to the face using P5js to manipulate the HTML5 canvas.

The clmtracker recognises 70 unique points on a face giving access to the x and y values of those points. Using these points it's possible to add glasses to the eyes, hats to the top of the head, false moustaches and just about anything that you can think of as a photo prop. The objects are controlled using code so that they orient to the head if it tilts from side to side by measuring the angle between the eyes and applying that to the prop. The scaling is worked out from the distance between the eyes so that the props scale to the face no matter the distance. A photo app needs to allow the user to save their image once happy, so that will also be the final feature.

1. The beginning

Open the 'start' folder in your code IDE and then open 'index.html' to edit. In the body section of the page add the logo and the instruction div tags as shown below. These will be styled up to sit over the top of other content with CSS.

```
<div id="logo"><img src="img/logo.png"
alt="Photo Boom Web Studio"></div>
<div id="inst">Use left and right arrow key
to change prop
  <br> Press S to save the screen
</div>
```

2. Defining the styles

In the head section add style tags to add this CSS to, or alternatively you could put this in its own CSS file. This code styles the body to fill the screen, removing the default browser margins and changing the typeface that will be used for the instructions.

```
html, body {
  width: 100%;
  height: 100%;
  overflow: hidden;
  padding: 0;
  margin: 0;
  font-family: "HelveticaNeue-Light",
"Helvetica Neue Light", "Helvetica Neue",
Helvetica, Arial, "Lucida Grande", sans-
serif;
  font-weight: 300;
  -webkit-font-smoothing: antialiased; }
```

3. Scaling the app

The app will run in a canvas element with a video element in the background. Here these tags are styled to take the full width and height of the screen, so that they are always viewed in the same place one over the other.

```
video {
  display: block;
  width: 100% !important;
  height: auto !important;
  vertical-align: top;
}
canvas {
  display: block;
  width: 100% !important;
  height: auto !important;
  vertical-align: top; }
```

4. Position the logo

The logo div tag will be placed in the top-left corner of the screen at a higher z-index to other content. The image inside this is then just resized to make this fit better on the display and not take up too much room within the design.

```
#logo {
  position: absolute;
  z-index: 1000; }
#logo img {
  width: 310px; }
```

5. Placing the instructions

The instructions are positioned in the bottom left of the screen, also above all other content. There is a slight padding added and the background is set to a semi-transparent black with white text to stand out over the top.

```
#inst {
  position: absolute;
  bottom: 0;
  left: 0;
  z-index: 1000;
  padding: 30px;
  background: rgba(0, 0, 0, 0.6);
  color: #fff; }
```

6. Sketching it up

Now save the HTML and switch over to the 'sketch.js' file. This will hold all of the code. At the top of the page add in some variables for the tracker, canvas and video, then other will hold various props that will be displayed on the screen.

```
var ctracker, cnv, videoInput;
var glasses, moustache, whiskers, pipe,
ears, hat, nose;
var mode, saving;
```

7. Preloading the images

Before the images can be displayed in the app they first have to be loaded in. Thankfully p5.js has a 'preload' function that will load these elements before the rest of the code runs. Here the images are loaded and stored in variables.

```
function preload() {
  glasses = loadImage("img/glasses.png");
  moustache = loadImage("img/moustache.
png");
  whiskers = loadImage("img/whiskers.png");
  pipe = loadImage("img/pipe.png");
  ears = loadImage("img/ears.png");
  hat = loadImage("img/hat.png");
  nose = loadImage("img/nose.png"); }
```

8. Setting up the app

The 'setup' function runs right after the preload has finished. This creates the video from a webcam and sets the size of the video, which is also the same size as the canvas. The pixel density of the display is set to 1, which means it ignores high density displays and this is necessary to make the image saving work.

```
function setup() {
  videoInput = createCapture(VIDEO);
  videoInput.size(800, 600);
  videoInput.position(0, 0);
  cnv = createCanvas(800, 600);
  cnv.position(0, 0);
  pixelDensity(1);
```

9. Add the tracker

The tracker is added and initialised, and it is set to work on the video. The 'mode' variable is used to change the prop that is being displayed on the image. The one here refers to the first image, which is the glasses. The 'saving' variable is used to save the frame as an image, so the user can take a snapshot.

```
  ctracker = new clm.tracker();
  ctracker.init(pModel);
  ctracker.start(videoInput.elt);
  noStroke();
  mode = 1;
  saving = false; }
```

10. Updating the screen

The 'draw' function is called every frame to update the screen. Here the screen is cleared between frames. The 'positions' variable is updated with the tracking position of the head. The 'if' statement checks the 'saving' variable, if it's set to true the video is written in as an image, otherwise the canvas has a transparent background.

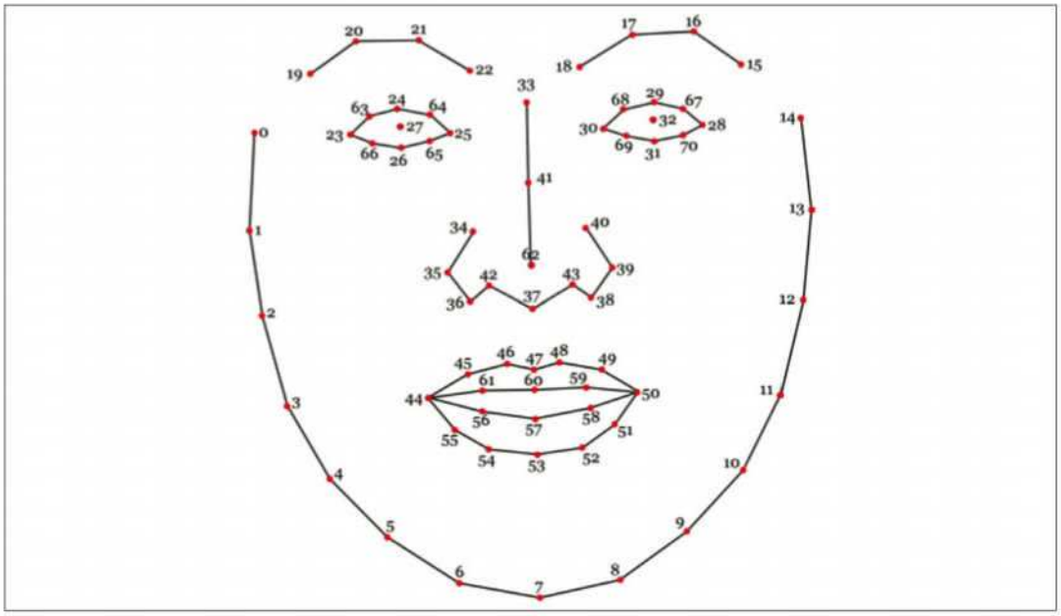
```
function draw() {
  clear();
  var positions = ctracker.
getCurrentPosition();
  if (saving) {
  image(videoInput, 0, 0, width, height);
}
```

11. Detecting a face

If a face is detected, then 70 positions of the face are stored in the 'positions' array variable. If the 'positions' is greater than one, then a face is detected. The angle is calculated using the position of the eyes and also the

Screen positioning

The video and canvas elements are created dynamically through JavaScript code so the div tags in the body have to be positioned 'absolutely' at higher z-index levels.



Mapping the points on the face

You may be wondering how the numbers for the points on the face actually relate to a face. How might something just cover the right eye, such as a pirate eye patch? The answer lies on the clmtracker GitHub page (**https://github.com/auduno/clmtrackr**). Here you will find an image with all 70 facial points shown in their relationship to the face. Using this as a guide it is possible to get hold of any point, such as the right eye using 'positions[32][0]' for the x axis position and 'positions[32][1]' for the y axis position. The tutorial uses the points between each eye to work out the angle that the head is tilted and the scale of the head as each user is likely to be positioned at different depths.

Above Here the glasses are added, and are scaling perfectly to fit the face in the scene

Above The whiskers are a lot of fun and there is no limit to what you can display as an overlay on the face

Above With the moustache working it's possible to see here the rotation to the face as well as the scaling

Above The ears are placed dynamically above the eyes using the length of the nose as the guide

Above The left eyebrow is the point at which the hat is positioned from, again using the nose as a guide

Above The last and easiest prop to be placed is the nose, which already has a position to track it to

distance between the eyes. This will enable the props to be rotated and scaled accurately.

```
if (positions.length > 1) {
  //angle between eyes gives angle of head
  var ang = atan2(positions[27][1] -
positions[32][1], positions[27][0] -
positions[32][0]);
  //distance between eyes so can work out
scale
  var d = int(dist(positions[27][0],
positions[27][1], positions[32][0],
positions[32][1]));
  var scl = map(d, 0, 290, 0, 1.8);
```

12. Position the sunglasses

The sunglasses are going to be positioned at point 33, which is between the eyes. The glasses are then rotated to the correct angle and scaled. Half the width and height is taken off the glasses so that its anchor point is the centre of the image as it's displayed in the centre of the eyes.

```
if (mode == 1) {
  push();
  //33 position between eyes
  translate(positions[33][0], positions[33]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(glasses, 0 - glasses.width / 2, 0
- glasses.height / 2);
  pop(); }
```

13. Changing the mode

The 'mode' variable will be changed with the arrow keys so when the mode is set to 2 then the moustache will be positioned at point 37, the base of the nose. You will notice that the position is set using 'positions[37][0]', which stores the x value and [1] stores the y value.

```
else if (mode == 2) {
  push();
  //37 position base of nose
  translate(positions[37][0], positions[37]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(moustache, 0 - moustache.width / 2,
-15);
  pop(); }
```

14. Placing the whiskers

The next prop for mode 3 is the whiskers image, which will be placed at the point on the mid nose. The scaling of this image is increased as well so that the whiskers stick out a little more from the side of the face and so that they start out a little from the nose on each side.

```
else if (mode == 3) {
  push();
  //41 position mid nose
  translate(positions[41][0], positions[41]
[1]);
  rotate(ang + PI);
  scl *= 1.8;
  scale(scl);
  image(whiskers, 0 - whiskers.width / 2,
0);
  pop(); }
```

15. Pipe dreams

Mode 4 will be the pipe image and this is going to be placed at the left-hand side of the user's lip. If the lighting is good, it tracks pretty well to the mouth opening and closing, which adds that realism for the prop working with the user.

```
else if (mode == 4) {
  push();
  //56 position left mouth
  translate(positions[56][0], positions[56]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(pipe, 0 - pipe.width, 0);
  pop(); }
```

16. Animal ears

The next prop is animal ears and a new distance is calculated for the length of the nose. The ears are going to be positioned between the length of the nose above the middle of the user's eyes, which is just above the hairline on most people.

```
else if (mode == 5) {
  //33 to 62 length of nose place this
distance above 33 for ears
  var d2 = int(dist(positions[33][0],
positions[33][1], positions[37][0],
positions[37][1]));
  push();
  //33 mid eyes
  translate(positions[33][0], positions[33]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(ears, 0 - ears.width / 2, 0 -
(ears.height + d2));
  pop(); }
```

17. I tip my hat to you

The hat prop is slightly tilted to one side, so this will be placed above the left eyebrow, and once again the length of nose is calculated to position this above the eyebrow. We have to use a dynamic position as there is no other way to know how close the user is to the camera.

```
else if (mode == 6) {
  //33 to 62 length of nose place this
distance above 33 for ears
  var d2 = int(dist(positions[33][0],
positions[33][1], positions[37][0],
positions[37][1]));
  push();
  //20 position left eyebrow
  translate(positions[20][0], positions[20]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(hat, 0 - hat.width / 2, 0 - (hat.
height + d2));
  pop(); }
```

18. Last prop

The last prop to be displayed is a clown's nose. This is easy to position as we have a point that is the nose. All of the props are now ready to go, but the user still needs a way to control them shifting from one prop to another with the keyboard.

```
else if (mode == 7) {
  push();
  //20 position left eyebrow
  translate(positions[62][0], positions[62]
[1]);
  rotate(ang + PI);
  scale(scl);
  image(nose, 0 - nose.width / 2, 0 - nose.
height / 2);
  pop();  } }
```

19. Finishing the draw

The end of the draw function is now completed and the image is saved out using the file name 'me.png' and the 'saving' variable is set to false so that hundreds of images aren't created! The user will control the saving by pressing 's' on the keyboard.

```
if (saving) {
  saveCanvas(cnv, 'me', 'png');
  saving = false; } }
```

20. Detecting key presses

The last part of the functionality requires key presses so this code detects if the right cursor key is pressed and cycles up through the modes. This in turn changes the props on display for the photo app on the screen.

```
function keyPressed() {
  if (keyCode === RIGHT_ARROW) {
  if (mode == 7) {
  mode = 1;
  } else {
  mode++; } }
```

21. Final step

The final part of the code does the same as the previous step for the left arrow key to cycle down. The 's' key is mapped to save the image. Save all your files and test them on a server to see this working. Make sure you have reasonable lighting on your face for best results.

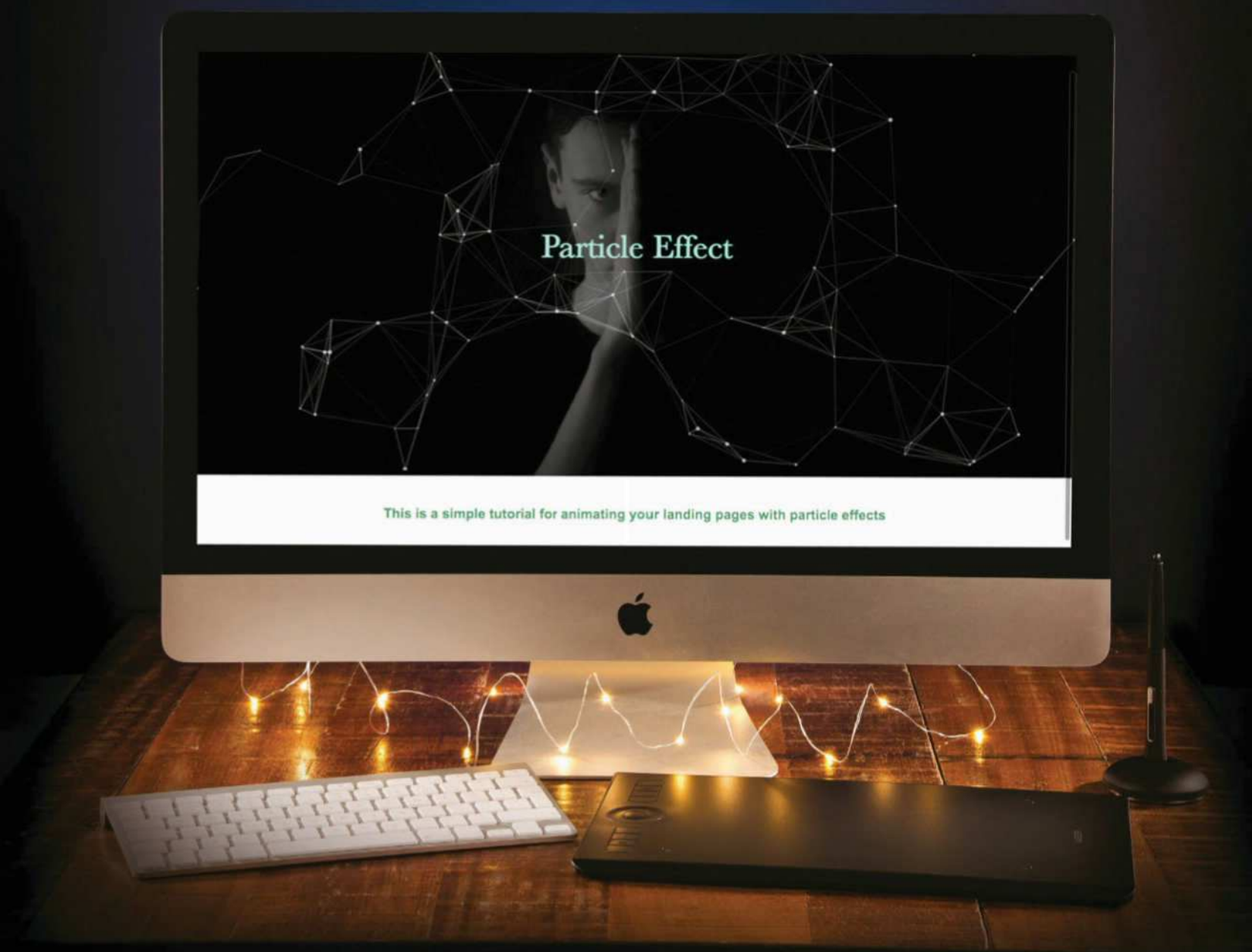
```
if (keyCode === LEFT_ARROW) {
  if (mode == 1) {
  mode = 7;
  } else {
  mode--; } }
  if (keyCode === 83) {
  saving = true; } }
```

Push and pop

Push and pop are often used with translate and rotate, because it moves the drawing position to a new place on screen and uses this as the origin to rotate that new drawing.

Add animated particles to pages

In this tutorial, the basics of particle.js library will be covered to help you animate your landing pages



DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/webdesigner

The first page that your visitors land on in your website has a significant influence on their subsequent actions in the website. On the one hand, it may influence them to navigate to other pages or undertake specific actions such as clicking buttons; while on the other, it may put them off, causing them to leave the site. As such, it is important to ensure your landing page appeals to your visitors in order to influence their latter decisions.

One way to achieve this feat is to implement interactivity using a particle system. Such a system comprises of a lot of tiny particles moving about in the page. The particles interact with one another in addition to responding to user actions such as mouse movements and mouse clicks.

This tutorial will begin by covering the basics of creating a fullscreen landing page using HTML5 and CSS3 and then proceed to animate it by implementing the library.

We'll also share some advanced tweaks of the library that will enable you to make your landing pages even more appealing. We'll also share some tips on frontend development and landing page design, the aim being that by the end of this tutorial you will have all the skills and knowedge you need to be able to the use the library to create amazing landing page effects.

1. Get started

Begin by creating a folder, particles, on your desktop to store the tutorial files. Create three additional folders within it: 'img' to store images, 'css' to store the styling files, and 'js' to store JavaScript files.

2. Create the landing page structure

Open your code editor and create an 'index.html' document to contain markup for the landing page. Begin by creating the basic structure and give a suitable title to the page.

```
<! DOCTYPE html>
<html lang=”en”>
  <head>
  <meta charset=”utf-8”>
  <meta name=”viewport” content=”width=device-width”, initial-scale=1.0>
  <meta http-equiv=”x-ua-compatible” content=”ie=edge”>
  <title>Particle effect </title>
  </head>
<body>
</body>
</html>
```

3. Add some content to the landing page

Create a header tag with the id 'fullscreen' and a section with the id 'text' in the body section. Within the header section, add an <h1> tag with the text 'Particle Effect' and an <h3> tag in the text section.

```
<body>
  <header id = “fullscreen”>
<h1> Particle Effect </h1>
</header>
<section id = “text”>
<h3> This is a simple tutorial for animating your landing pages with particle effects </h3>
</section>
</body>
********
```

Render the page in your browser. It should appear as shown in the first of the two images below.

4. Styling the landing page – link the CSS file

The landing page currently looks very basic. The aim is to add an image background and style the text so that it is centred on the page. Open your favourite code editor and then create a 'styles.css' file inside the 'css' folder. Now, create a link to this file in your HTML document by adding the following code in the head section:

```
<link rel=”stylesheet” href=”css/styles.css” >
```

5. Styling the landing page style the body text

Begin by setting padding and margin to '0' for all elements in CSS. This ensures that the design remains consistent throughout the page. Note that at this point, you should not observe any noticeable change in your HTML document.

```
*{
  margin:0;
  padding:0; }
```

Style the body text by adding the font family, font size, color and line height information. This styles all paragraphs and text added to your html document.

```
body {
font-family: Garamond, Baskerville, “Baskerville Old Face”, “Hoefler Text”, “Times New Roman”, serif;
  font-size: 17px;
  color: coral;
  line-height: 1.6;
}
*******
```

At this point, the landing page is still not appealing.

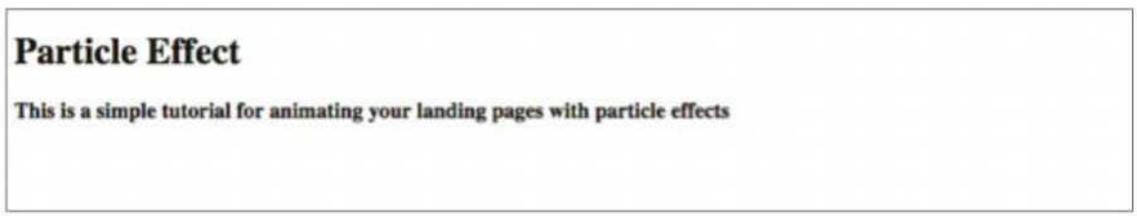
6. Style the landing page background

Begin by adding an image background. Use the free image of a man downloaded from **pexels.com**. Add the following styling code in the CSS file:

```
#fullscreen
{
  background-image: linear-gradient(rgba(0,0,0,0.7), rgba(0,0,0,0.7)), url(‘../img/man.jpeg’);
  background-size: cover;
```

Alternative centring

You can adopt a table layout and specify a middle alignment for the heading. You can choose to work with padding and absolute positioning or adopt a Flexbox layout.



Top
To get started create a simple landing page with a title and the necessary HTML tags

Bottom
With all of the HTML elements in place, the body text can be styled on the landing page

```
  background-position: center;
  height: 100vh;
}
```

The first line adds the image to the background in addition to a linear gradient set to black with an opacity of 70 per cent. The second and third lines detail the background size and position properties, while the last line informs the browser that we would like the viewport height to be at '100' at all times, thereby ensuring it is completely visible.

7. Style the main heading

The heading is currently aligned at the top mid section. In order to style it to appear at the centre of the page, simply add the following code:

```
#fullscreen h1
{
  font-size: 50px;
  position: absolute;
  top: 270px;
  left: 500px;
  font-weight: 200;
  color: aquamarine;
}
```

The code assigns absolute positioning to the heading and aligns it appropriately at the top and left sections. Add 'font-weight' and 'color' to make it look appealing.

8. Style the section text

To style up the <h3> tag in the text section, add the following code:

```
#text h3
{
  font-size: 25px;
  padding: 40px;
  color: #408159;
  font-size: 25px;
  font-family: Arial, "Helvetica Neue",
Helvetica, sans-serif;
  text-align: center;
}
```

Tweaking the system

It can be challenging to compare the effects of different parameter configurations. To ease the process, head over to **vincentgarreau.com/particles.js/#default** where controls are provided.

9. Download the particles.js library

Now that the simple landing page is complete, download the 'particle.js library' from Github. Follow this link: **github.com/VincentGarreau/particles.js/**. Click the 'Clone or download' option from where you can download the library.

10. Unpack the particles.js library

Once downloaded, unpack the files and copy three files to the 'js' folder created in Step 1. These files are" 'app.js', 'particles.js', and 'particles.min.js'. Note that while the last two are located in the main 'particle.js' master folder, 'app.js' is located in the 'demo / js' folder.

11. Implement the particles.js library – HTML code

To implement the library, create a div with the id 'particles-js' and add links to the three 'js' files in the body section. However, it is important to note that the 'particles' div and 'fullscreen' div are both performing the same function since they act as the container of the landing page. This implies that we comment out the current 'fullscreen' div and copy its contents to the 'particles' div. The HTML page code will now appear as follows:

```
<body>
  <div id =”particles-js”>
<h1> Particle Effect </h1>
<script type=”text/javascript” src =”js/
particles.js”></script>
  <script type=”text/javascript” src =”js/
particles.min.js”></script>
  <script type=”text/javascript” src =”js/
app.js”></script>
  </div>
  <section id = “text”>
  <h3> This is a simple tutorial for animating
  your landing pages with particle effects </
  h3>
  </section>
  </body>
```

12. Implementing the particles.js library – CSS code

In the same way, we comment out the the styling assigned to the 'fullscreen' div and paste it to the 'particles-js' div. The code should now appear as below for the main container:

```
#particles-js {
  background-image: linear-gradient (rgba
(0,0,0,0.7), rgba(0,0,0,0.7)), url('../img/
man.jpeg');
  background-size: cover;
  background-position: center;
  height: 100vh;   }
```

13. Implementing the particles.js library – more CSS code

Similarly, comment out styling code previously assigned to the <h1> and copy it to the 'particles-js' <h1> div.

```
#particles-js h1
{
font-size: 50px;
position: absolute;
top: 270px;
left: 500px;
font-weight: 200;
color: aquamarine; }
```

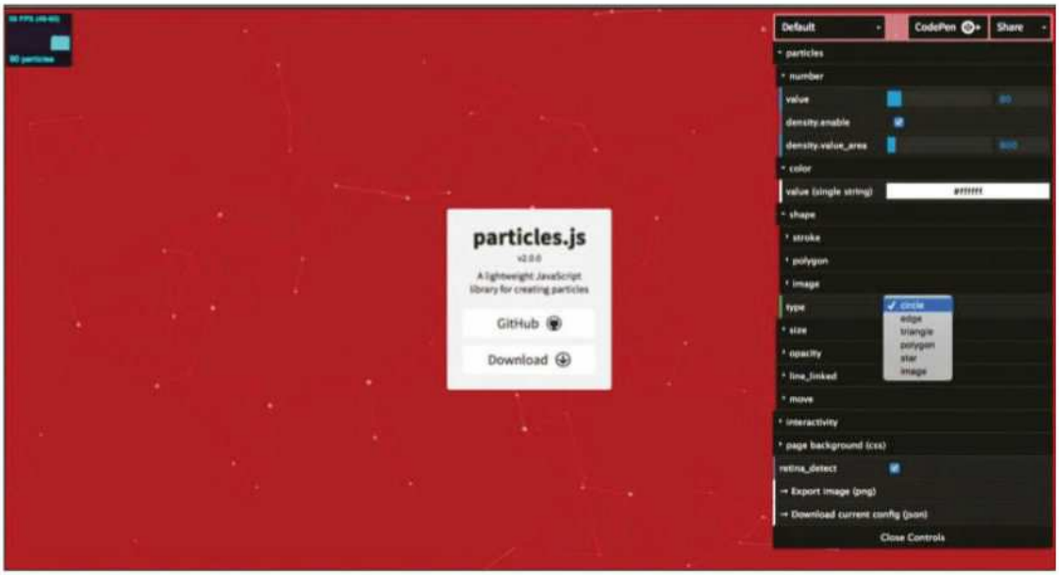
The landing page should now appear as shown in the main image at the bottom of this page, with particles appearing in the background.

14. Tweak the particle system

Implementing the particle system in your web page is only the tip of the iceberg. To maximise its capability, we need to tweak its different parameters. Two important aspects are tweaked: the particles and the interactivity.

Top
To finish up the landing page used in this tutorial a background image has been added and set to fill the screen. The text is styled to contrast with the background colour

Right
The finished landing page with particle effects added



Creating a custom particle configuration

While the default particle system configuration works, you are not limited to using it in your website as provided. **vincentgarreau.com/particles.js/#default** provides controls to visually test out different parameter values and download them to your offline location. The controls enable you to configure the particle options, interactivity and page background CSS. Additionally, the visual controls provide interactivity through the use of sliders to increase or decrease parameter values; clickable options to enable or disable parameters; and text boxes where options such as colour codes can be manually added. Once configured, you can download the configuration as a JSON file to implement in your website. The figure (left) shows the visual system.

To begin this process, open the 'app.js' file in the 'js' folder using your code editor.

15. Particle number and density

To begin with, the particle system provides the option to change the number of particles that are rendered, and the density value area within which they move about. In the 'app.js' file that you already have open, locate the number and density parameters. Amend the value and value area respectively.

```
"particles": {
  "number": {
  "value": 100,
  "density": {
  "enable": true,
  "value_area": 800
```

Note that increasing the two parameters causes a higher number of particles to be rendered, while reducing the two parameters lowers their number.

16. Colour and shape

To change the colour, simply change the 'color' code to your preferred option. Similarly, increase or decrease the shape stroke size; the type of shape (circles, edges, triangles, polygons, stars, or images); the number of polygon sides; and the image source that the particle system uses. For instance, changing the shape to 'star' and the colour to 'green' produces different results.

17. Size and opacity

Under the 'size' property, it is possible to enable or disable particle animation; tweak the animation speed (higher values means higher speed); increase or decrease the particle sizes; and disable the random movement of the particles. Similarly, increasing opacity makes the particle colour sharper.

18. Line linked

The 'line linked' parameter, when enabled, creates lines connecting the different particles. You can disable the parameter, if you wish, so that you particles move around without any having links within them. Similarly, it is also possible to increase the distance between the different shapes, as well as change the colour, opacity and width of the connecting lines – try experimenting with each of these options until you achieve the look that you're after for your site's landing page.

19. Movement

The movement of the particles can be disabled, causing them to remain static on your landing page. The direction of the movement of the particles can also be tweaked (none, top, top-right, right, bottom-right, bottom-left, left, left, top-left); as can the particles' speed (higher means faster). You can also adjust the 'out' mode (out or bounce). For instance, changing the direction of the particles to top-right leads to them moving towards the top-right section of the screen.

20. Interactivity – mouse actions

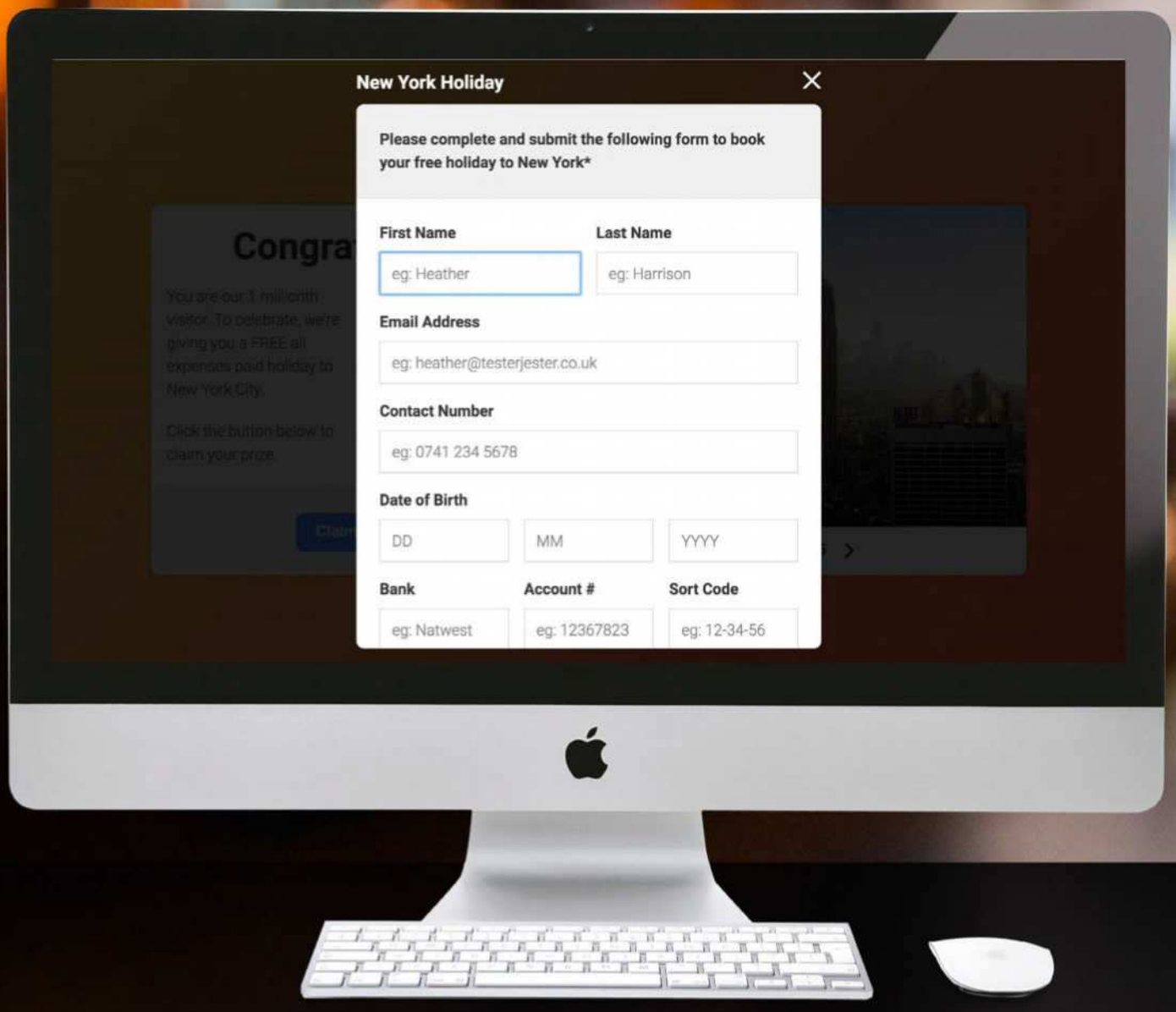
The interactivity of the particle system can be tweaked to respond to mouse actions. 'On hover' actions can be amended to either grab, bubble or repulse; while 'on click' actions can be tweaked to bubble, push, remove or repulse. Both actions can also be disabled, if you wish.

21. Modes

The four modes of interactivity can be tweaked. For the grab and repulse modes distance can be increased while with the bubble mode, distance, size, duration, opacity and speed can be tweaked. Finally, push and remove modes can have their particle number increased.

Build a modal component in React

Learn how to create a simple, usable modal component leveraging the power of React's portal functionality



DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/webdesigner

React is one of the most popular JavaScript frameworks in use today. Created by Facebook to help it build user interfaces, React brings simplicity, speed, and scalability to web projects.

In this tutorial, we're going to use React to create a simple modal component, which is optimised for both desktop and mobile devices. To enhance the modal, we'll be using React's relatively new and very useful portal functionality, which enables us to render an element outside of the DOM hierarchy of its parent component, whilst still behaving like a normal React child.

To begin, download the tutorial files from Filesilo, then open the 'website-template' directory in your preferred text editor. In Terminal, 'CD' into 'website-template' and then run 'npm install' to install all of the project's dependencies. We can then start a development server by running the 'npm run start' command, enabling us to view the website at 'localhost:3000'. As we make changes, this page should automatically refresh, so helping us to ensure that we're always looking at the most up-to-date version.

1. Create component structure

In 'src/components', create a new file called 'Modal.js'. All the CSS for the modal is already present in '_Modal.scss', enabling us to focus purely on building the React component. Within the new file, we'll create a new React component called 'Modal' by importing and then extending React's component class.

```
import React from 'react';
import Close from '../assets/img/close.svg';
export default class Modal extends React.
Component {
  constructor(props) {
  super(props);
  }
}
```

2. Validate props

Next, we'll set some validation rules for properties, which our Modal will accept when used in parent components. Below your import React statement, add "import PropTypes from 'prop-types'" and then below the constructor function in the component itself, add the following snippet:

```
static propTypes = {
  active: PropTypes.bool.isRequired,
  focusElem: PropTypes.string,
  handleClose: PropTypes.func.isRequired,
  returnFocusElem: PropTypes.object,
  title: PropTypes.string }
```

3. Create modal JSX

Create a new function within the component called 'modalContent'. We'll use this to store the JSX (AKA: the HTML) of our modal. The use of 'this.props.children' enables us to pass the modal its content like a regular HTML element without us having to modify the component directly.

```
modalContent = () => { return (
  <div className='c-modal'>
  <div className='c-modal__wrapper'>
  <div className='c-modal__title-wrapper'>
  {this.props.title ? <h1 className
  ='t-beta c-modal__title'>{this.props.
title}</h1> : ''}
  <img src={Close} className='c-modal__
close'
  tabIndex='0' alt='Close Modal' />
  </div>
  <div className='c-modal__box'>
  <div className='c-modal__content'>
  {this.props.children}
  </div>
  </div>
  </div>
  </div>
)}
```

4. Render modal – part 1

For now, let's render the modal to the screen so we can see what we're working with. In the modal component, create a new function called 'render' and pass it our 'modalContent' function. Add the below as the last function of the React component:

```
render() {
  return this.modalContent();
}
```

5. Render modal – part 2

Open 'CallToAction.js'. This is the parent component, which will store our new modal. First off, let's import the modal to this component by adding "import Modal from './Modal';" to its list of imports. Then within this components render function, just before the closing 'div' element, add the following snippet:

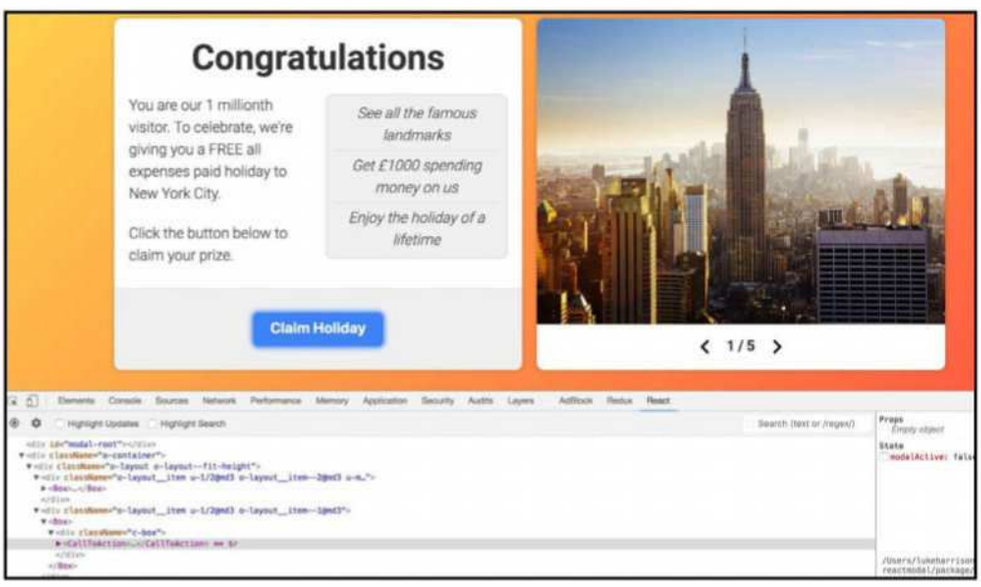
```
<Modal title='New York Holiday'>
  <AppForm />
</Modal>
```

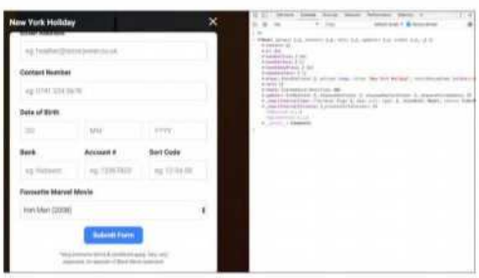
6. Add portal mount point

You should now be able to see the modal. Since it's still located within the DOM tree of its parent component, the parent styles are interfering with its display. One way to fix this is to render the modal right at the top of the DOM tree, just after the opening 'body' element. This can be done using React Portals, which provide a way to render children into a DOM node that exists outside the hierarchy of the parent component. Let's start by adding a new element to 'App.js', the top-level React component, to hold the modal. Add the following snippet just after this

Errors

One of the good things about developing whilst the dev server is running (AKA: 'npm run start') is that if you make any coding errors, React will display these right at the forefront of your app so you know what has gone wrong immediately.





Left
Within this tutorial's app, you can use React dev tools to examine each individual component, see what props they're being passed, as well as any internal state that they are holding

Top
With a component selected in the dev tools, if you then type '$r' into the standard dev tools console, you can view the React component as a pure javascript object, which can come in very handy during development

component's opening 'div' attribute:

```
<div id='modal-root' />
```

7. Create the portal

Next, back in 'Modal.js', add "import { createPortal } from 'react-dom';" just after the React import. Then in the 'constructor' function, just after the 'super' function, add "this.el = document.createElement('div');" This is the wrapper element, which the 'createPortal' function requires to hold our modal content. Finally, replace the existing 'render' function with the snippet below to render the modal through the context of a React portal:

```
render() {
  return createPortal(
  this.modalContent(),
  this.el
  ); }
```

8. Mount the portal

You will notice the modal has vanished. This is because we still haven't mounted the modal portal into the 'div' element we created at the top of the DOM tree. Add the below snippet to the 'modal' component to make it appear again:

```
componentDidMount() {
  document.getElementById('modal-root').
appendChild(this.el);
}
```

9. Tracking modal state

At the moment our modal is always showing. Obviously this isn't ideal, so let's build in some toggle functionality. Open 'CallToAction.js', and add the following snippet just before its 'render' function. This adds two things: a state object containing a Boolean to track if the modal should be showing, and a function to set this Boolean to 'true', which should also trigger the modal.

```
state = {
  modalActive: false
}
showModal = () => {
  this.setState({
  modalActive: true
  })
}
```

10. Pass state to modal

We now need to pass the state object to the modal so it knows when it should be showing, or not. To do so, add 'active={this.state.modalActive}' to the opening <Modal> tag. Then, still within 'CallToAction.js', replace the 'Claim Holiday' button component instance with the version in the snippet below. What this does is pass our 'showModal' function to the button, so when it's clicked, the 'modalActive' variable in the state object is set to 'true'. We wrap our 'showModal' function in another function to stop it from immediately firing on render.

```
<Button handleClick={() => this.
showModal()}>Claim Holiday</Button>
```

11. Conditional rendering

In 'Modal.js', let's now change the 'render' function so the modal only ever appears in the DOM if its 'active' prop is true.

```
render() {
  if(this.props.active) {
  return createPortal(
  this.modalContent(),
  this.el
  );
  }
  else {
  return false;
  }
}
```

12. Create close modal function

Now we'll look at closing the modal when its 'X' icon is clicked. In the 'CallToAction.js' component, duplicate the 'showModal' function, rename it 'closeModal', then make it set the 'modalActive' variable to 'false'. Next, we should pass this function to the modal so it can be called from within it. To do so, pass it as the 'handleClose' prop as in the snippet below:

```
<Modal
  handleClose={() => this.closeModal()}
  active={this.state.modalActive}
  title='New York Holiday'
>
```

13. Attach the close function

Next, within 'Modal.js', add 'onClick={this.props.handleClose}' to the 'img' element, so that our passed function will trigger on clicking the 'X' icon. We should also consider people who navigate using keyboards. Add the below function to the modal component, and then add 'onKeyPress={(e) => this.handleKeyPress(e)}' to the 'X' icon 'img' element. This will add functionality so that when the 'X' icon is in focus, and the user presses Enter, the modal closes.

```
handleKeyPress = (e) => {
  if(e.which === 13) {
  this.props.handleClose();
  }
}
```

14. Add more ways to close modal

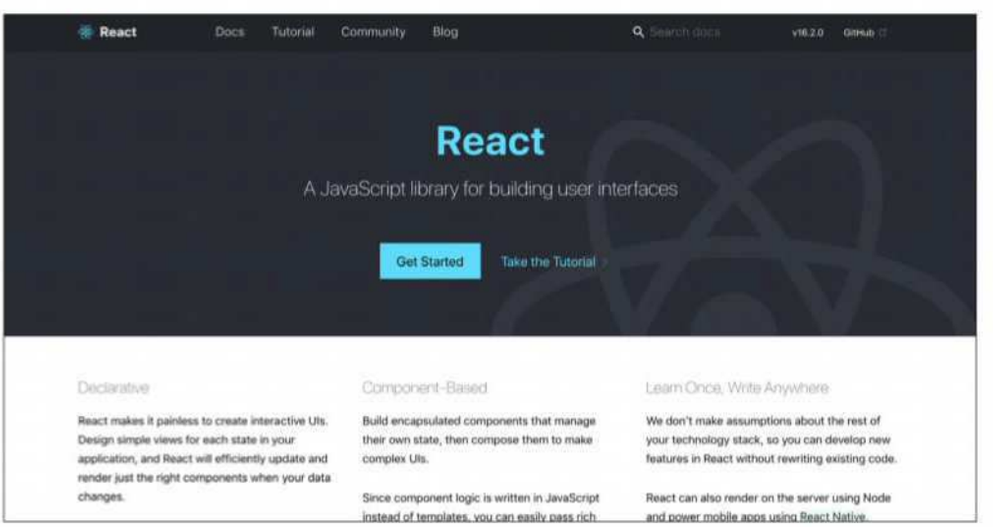
We should also add additional functionality so the modal closes if the user clicks the darkened modal background, or presses the Esc key. Let's cover the former first. Add the snippet below to the modal component and then add 'onClick={(e) => this.handleClick(e)}' to the 'c-modal' element. What this does is listen for clicks on the modal element and then figures out if it was on the modal box, or the background, closing the component if it's the latter. We have to use a 'try/catch' here for Internet Explorer compatibility purposes.

```
handleClick = (e) => {
```

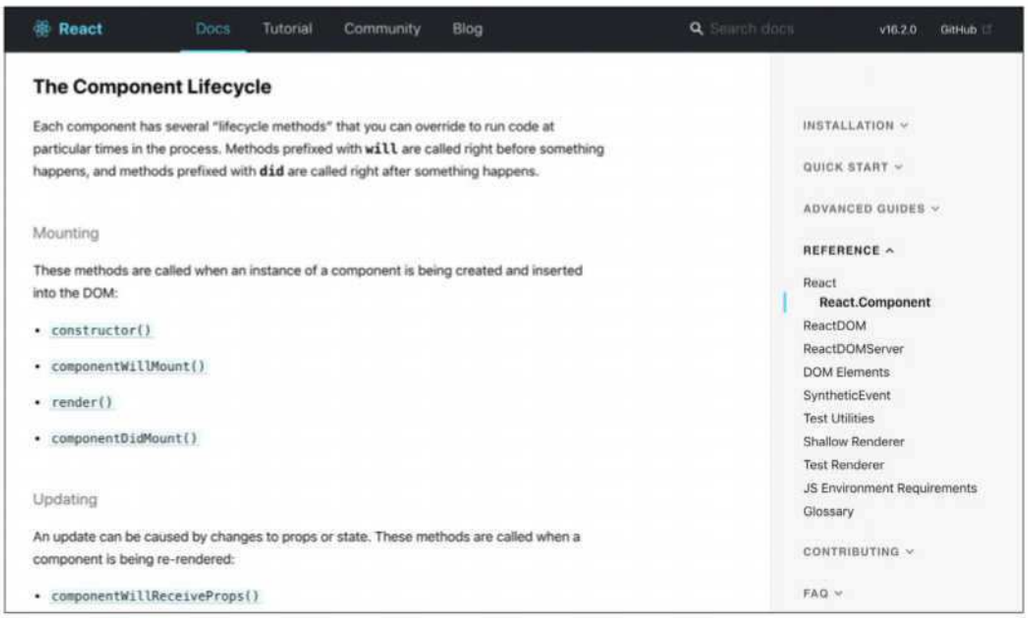
Learning from others

With React dev tools installed, you can examine the contents of any other public React website and learn how it's structured. Facebook and Instagram are two good examples to check out.

Top
The template for this tutorial was originally generated using Create React App. This is a handy tool created by Facebook that lets you quickly generate a React app with all the nitty gritty setup and build configuration taken care of



Right
All of the app build configuration is hidden behind a dependency called 'react-scripts', which means as you update this package, you will always have the most up-to-date configurations in place



React lifecycle methods

In this tutorial, you'll be using lifecycle methods in React components. These are functions which are run at particular times in the process and can be overridden with your own code. Some lifecycle functions run before the component is mounted to the page, and some run afterwards. In this tutorial specifically, we'll be using:
'constructor': This lifecycle function is run before the component is mounted.
'componentDidMount': This one is run immediately after the component is mounted.
'componentWillReceiveProps': This is run once a mounted component receives new props from a parent component.
For information on all of the different lifecycle methods (there's a lot), you can find extensive documentation at **reactjs.org/docs/react-component.html.**

```
  try {
  if(e.target.classList.contains('c-modal'))
{
  this.props.handleClose();
  }
  }
  catch(error) {
  console.log(error)
  }
}
```

15. Add Esc key functionality

Now we'll add the Esc key functionality. In the modal's 'componentDidMount' function, add the following snippet to create an event listener on the document. This listens for any presses of the Esc key and then closes the modal in response.

```
document.body.addEventListener('keydown',
(e) => {
  if(e.which === 27) {
  this.props.handleClose();
  }
});
```

16. Create object for scroll position

One thing you will notice is that on smaller screens, when you scroll to the end of the modal content, the page content underneath the modal then starts to scroll. Ideally, we'd want the page content to freeze in place once the modal is activated, with the user's scroll position being restored on modal close. To do this, we need to track the user's scroll position on modal activate. Create a new state object in the modal, underneath its 'propTypes' validation, like below:

```
state = {
  cachedScrollPosition: null }
```

17. Cache scroll position & freeze

Next, we'll add a new 'life cycle' function to our modal component, which triggers when new props are received. In the function, we run an 'if' statement to see if the modal needs to be shown. If this is the case, we record the user's current scroll position in our state object, and then in the 'setState' functions callback, add a special 'is-fixed' class to the body, which prevents scrolling.

```
componentWillReceiveProps(nextProps) {
  if(nextProps.active) {
  this.setState({
  cachedScrollPosition: window.pageYOffset,
  }, () => {
  document.body.style.top = '-${this.state.
cachedScrollPosition}px';
  document.body.classList.add('is-fixed');
  });
  } }
```

18. Restore scroll on modal close

We also need to account for when the modal is closed. If we don't, the 'is-fixed' class will be stuck on the body and the user won't be able to scroll. Add the below 'else if' statement just after the 'if' statement. This restores the page to its previous state, making sure the user retains their original scroll position.

```
else if(this.props.active && !nextProps.
active) {
   document.body.style.
removeProperty('top');
   document.body.classList.remove('is-
fixed');
   window.scrollTo(0, this.state.
cachedScrollPosition); }
```

19. Store ref for the button DOM element

One helpful improvement we could make to the modal is so when it's closed, the focus is taken back to the button used to trigger it. That way, if the user is navigating the website via tabbing, they don't have to re-tab back to their previous position prior to opening the modal. In 'CallToAction.js', replace the 'Button' component with the new one below. What we're doing here is storing a reference to the button's DOM element.

```
<Button
  handleClick={() => this.showModal()}
  refName={el => this.claimHolidayBtn = el}
>
  Claim Holiday
</Button>
```

20. Return focus to trigger button

Pass the reference to the Modal component via its 'returnFocusElem' prop, like so: 'returnFocusElem={this.claimHolidayBtn}'. Then within 'Modal.js', we need to build a 'bridging' function so we can perform actions after the user has signalled that the modal should close, but before it actually closes. Add the below function to the modal component, and then replace all references of 'this.props.handleClose' to 'this.handleClose' (Except for within the 'bridging' function we've created).

```
handleClose = () => {
  this.props.handleClose(false);
  if(this.props.returnFocusElem) {
  this.props.returnFocusElem.focus();
  }
}
```

21. Success! Modal created!

We've created an easy to use, flexible modal, which you can control via mouse or keyboard commands. Moving forward, you could improve this even further by adding things like animation via the 'ReactTransitionGroup' package. You can see a working example of this in the bonus-package directory.

Create an animated running water effect

Inspired by **en.oollee.com**

DOWNLOAD TUTORIAL FILES www.filesilo.co.uk/webdesigner

EXPERT ADVICE
Simplifying effects
The original effect relied heavily on the ability to perform pixel manipulation on the target image. The most suitable option for this is the use of the canvas element, which enables JavaScript to perform pixel operations on an image space. While the canvas approach is the most ideal option for the effect, it's not as easy to use and understand as regular DOM/CSS styling. This type of effect would also require more than the nine steps of this tutorial.

So what is the option to recreate the water drip effect used in the original website? After a bit of thinking and experimentation, it appears that pixel manipulation can be 'faked' with CSS. While the technique appears to show manipulation of the pixels to produce the dripping effect, the method uses multiple elements sharing the same background image – showing only a specific part for each animation frame.

The use of CSS formatting provides an opportunity to further adapt the technique, such as by defining a shape for the outer container. Similarly, the technique could make use of SVG elements for better shape definition. All achievable without adding unnecessary the complexity through JavaScript.

<comment>
What our experts think of the site

Avoiding the complications

There are many advantages to keeping your code simple. Smaller file size and lower processing requirements are just two advantages that are relevant to your web projects. Where laptops and mobile devices are concerned, these enable your website to load faster and help preserve battery life

Leon Brown, freelance web developer

Technique

1. Initiate page document

The first step is to define the page document. A container called HTML is used to store the head and body sections. While the head section is used to load the external JavaScript and CSS resources, the body section is used to store the visible content elements created in Steps 2-5.

2. Body content definition

The initial content consists of an article container that has a 'drip' class applied to it. This class is used as the reference name for JavaScript and CSS find where to apply the effect on the web page.

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

3. JavaScript initiation

Create a new file called 'code.js'. The first step is to set an event listener to wait for the page to load – allowing for web page elements to be referenced. A loop counting up to the number identified by the 'count' variable is also initiated within this event.

```
window.addEventListener("load", function(){
var count = 10;
for(var i=0; i<count; i++){
*** STEP 4 HERE
}
});
```

4. Effect node creation

Now 'span' nodes are created within the loop defined in Step 3. These span nodes are set to have a unique top margin and opacity based on their position. These unique attributes are important for the CSS rules to provide the illusion of pixel manipulation.

```
var node = document.createElement("span");
node.style.marginTop = (i*2)+"mm";
node.style.opacity = (i/100)+.5;
```

5. Apply node to drip

Each node generated within the 'for' loop of Step 3 is required to be added to the drip. A query selector is performed on the 'document' to search for the element using the drip class, upon which 'appendChild' is used to attach the node created in Step 4.

```
document.querySelector(".drip").
appendChild(node);
```

6. CSS: HTML body

Create a new file called 'styles.css'. This step sets the HTML document and its body section to cover the full screen without any visible border spacing. The body has its background set to have a picture sized to cover the full available space.

7. Drip container

The drip container uses absolute positioning so that it can be moved by the applied 'outAnim' animation. Its height is set to 50 per cent of the view height (vh), with the animation duration set to loop infinitely over a duration of five seconds.

```
.drip{
position: absolute;
display: block;
height: 50vh;
animation: outerAnim 5s ease-in-out infinite;
}
```

8. Drip: droplets

The effect is made by applying the same background to the child elements inside the drip container. These are set to cover the full width, but only at a height of 5mm. A slight blur is applied as part of the distortion effect. The 'innerAnim' animation is also applied to adapt these items during the animation.

```
.drip > *{
position: absolute;
display: block;
width: 100vw;
height: 5mm;
filter: blur(2px);
background: url(background.jpg)
rgba(0,0,0,.5);
background-size: cover;
animation: innerAnim 5s ease-in-out infinite;
}
```

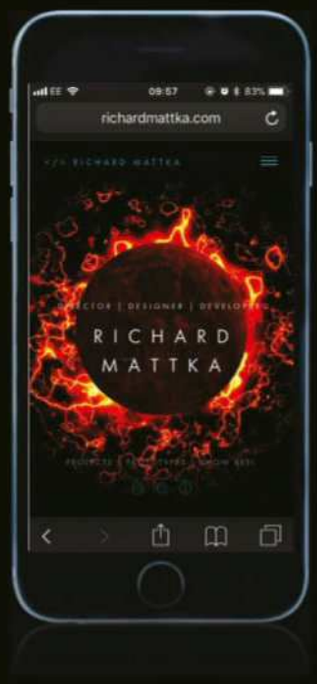
9. Drip/droplet animation

Two animations are defined for the animation. While the 'outerAnim' animation is responsible for moving the drip container in and out of view, the 'innerAnim' adapts the portion of the background shown by the inner child elements. These combine to produce the changing 'distortion' effect.

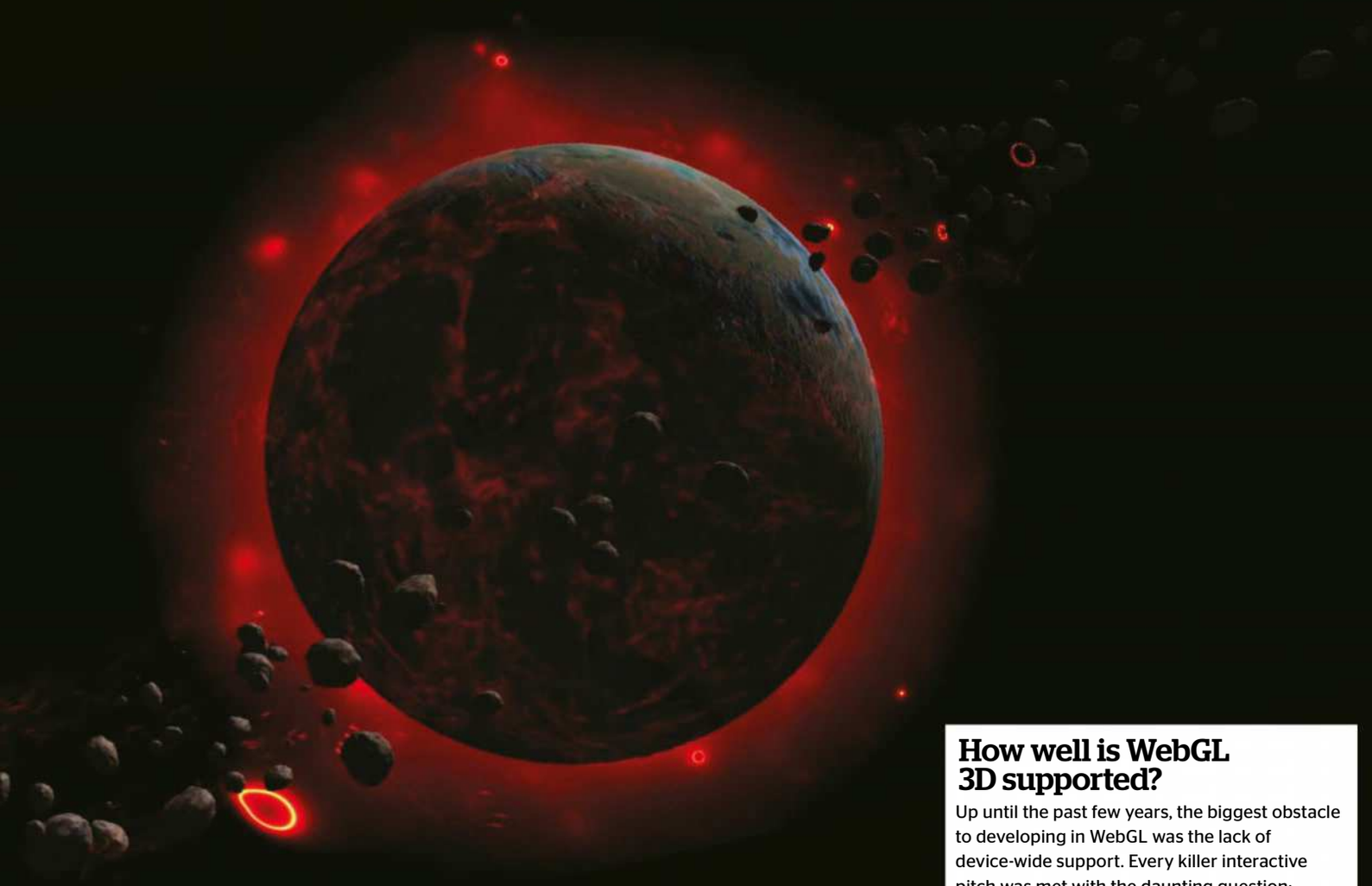
```
@keyframes outerAnim {
0% { opacity: 0; top:0; }
20%{ opacity: 1; }
100% { top: 100vh; opacity: 0; }
}

@keyframes innerAnim {
from { background-position:0 -100%;}
to { background-position: 0 0;}
}
```

UNLEASH THE POWER OF WEBGL WITH THREE.JS

Learn how to create realistic 3D scenes on the Web



Explore alien landscapes, underwater worlds or ancient gauntlets. Interact with life-like models that speak and react as you engage with them. Build beautiful cinematic sequences to launch into a new game or website. Learn the latest features of a new computer by viewing it from every possible angle, or dive into an exploded view. Imagine a site for a new car, where you could examine each part in as much detail as you wish, then watch it reassemble before your eyes. Turn on your VR headset, get behind the wheel and take your customised car for a spin. All of this is possible and so much more using WebGL, all right in the browser without plugins. 3D models are at the heart of these amazing experiences. This article is going to explore using 3D models in WebGL, and create beautiful results. We'll show you data formats, performance considerations, resources and how to get started. Using the popular Three.js library, you'll learn about importing and using models, lighting and textures to enhance surface detail, the importance of environmental maps and optimisation tips to help your project run smoothly. You'll be a able to follow along, making your own 3D scene. You'll load primitives to start, then exploring model loading and techniques to make your scenes more photo-realistic. More than enough to get started and plenty to fuel your next projects, with a few tips as well. Many of the 3D principles that you will learn here are relevant in any other 3D library or environment.



How well is WebGL 3D supported?

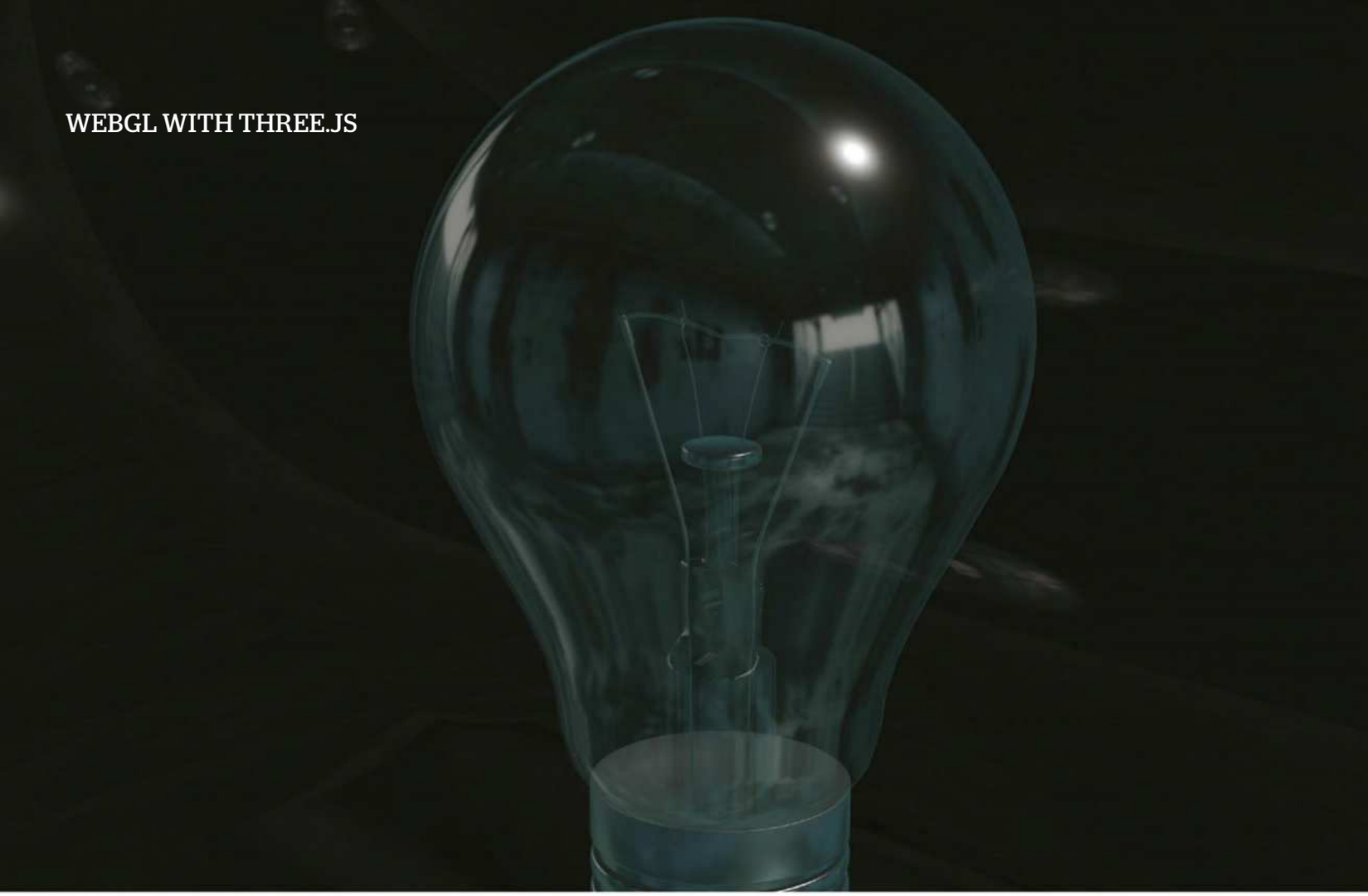
Up until the past few years, the biggest obstacle to developing in WebGL was the lack of device-wide support. Every killer interactive pitch was met with the daunting question: "Great, but will it run on mobile, or do we have to serve up some second-rate fall-back experience?" Desktop support was getting there for some browsers, but not all, and mobile was a deal-breaker for most professional projects.

Fast-forward to 2018, and with advances in device hardware and processing, we now have gorgeous displays with incredibly powerful processors. Browsers support WebGL natively with overall support closing in on 95%, virtually 100% on all major devices and browsers (**https://caniuse.com/#search=webgl**).

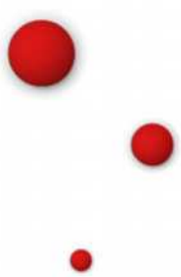
WebGL is now integrated into web standards, allowing physics, image processing and effects as part of the webpage canvas, all GPU-accelerated! That means amazing real-time 3D graphics, powerful effects via Graphic Shaders and immersive video are all possible. Complex models with high levels of detail, reflections, environmental maps, shadows – all possible. This requires no special plugin or downloads. All users now have instant access to beautiful experiences right in their browser and right in the palm of their hand.

"Every image, every scene, can tell a story. Advanced WebGL now enables photorealistic, interactive rendering that can reach mass audience through the web without plugins. It's an exciting time to be in web development and VFX"

Richard Mattka
Award-winning interactive director, author, VFX artist



"JavaScript libraries abstract away the complexity of WebGL and allow you to create real-time 3D content in a much easier manner"



What 3D are we talking about?

There is more than one way to get 3D on the web

WebGL is a JavaScript API based on the OpenGL 3D graphics standard, allowing JavaScript plugin-free access to the graphics hardware, via the HTML5 <canvas> element. WebGL enables real-time 3D graphics in webpages.

WebGL

A Web Graphics Library is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plugins. WebGL programs consist of code written in JavaScript and shader code that is written in OpenGL Shading Language (GLSL), a language similar to C or C++, and is executed on a computer's graphics processing unit (GPU).

WebGL can display complex images, animations and physics simulations all super fast inside a canvas element. It is a web standard, designed and maintained by the non-profit Khronos Group.

CPU vs GPU?

Why would you want to take advantage of GPU over CPU for rendering your 3D scenes? In a word, power. Even for your 2D work, you want this power.

The Graphics Processing Unit (GPU) is designed specifically for rendering graphics, where the CPU is a generalised work horse. The GPU is really good at compute-intensive operations, such as video renders and physics simulations.

3D Libraries vs native WebGL

Since WebGL is a low-level API, it can be challenging to use. You need to write hundreds of lines of code to perform basic tasks. JavaScript libraries abstract away the complexity of WebGL and allow you to create real-time 3D content much more easily. For 2D work there is PixiJS, Phaser, EaselJS and many others. For 3D, Three.js and BabylonJS are great options. A-Frame uses Three.js and WebGL elements allowing you to create webVR experiences.

Three.js

Three.js is an open source 3D library created by Ricardo Cabello AKA Mr Doob. It's focused solidly on 3D with an excellent library of components, classes and tools. It also boasts a very active community and extensive documentation. It is feature-rich, flexible and has excellent performance, making it a good choice for development.

Real-time rendering

Real-time rendering offers some pretty significant advantages over pre-rendered scenes. Games require constant interaction between player input and what is rendered. Web experiences that allow users to direct the action, move the camera and objects all leverage rep-time rendering. WebGL allows real-time rendering right in the browser, without plugins.

Anatomy of a 3D scene

3D scenes are comprised of a common set of elements

The Scene

A scene is a container object that encapsulates all the other objects and elements that we wish to show. Think of it like an empty stage we are going to film.

Camera

Like filming a movie, we need a camera to capture the action. It can move, tilt, pan and zoom. Three.js has a few different camera classes, including a Perspective Camera, which we will use for 3D, and an Orthographic camera used for rendering 2D scenes or UI elements. For this article we are most interested in the Perspective Camera.

You can define the Field of View (FOV), which is the extent of the scene that is seen on the display at any given moment. The aspect ratio is typically the width of the canvas element divided by the height. You can also define the near and far clipping planes, which define what is rendered by the camera.

Renderer

The Renderer handles the display of the 3D scene using WebGL. It targets a HTML <canvas> element to draw into. You can define the size at which you want it to render your scene as well. Typically this will be set to include the width and height of the browser window.

3D Objects

3D Objects are comprised of geometry, materials and can be transformed in 3D space.

Geometry

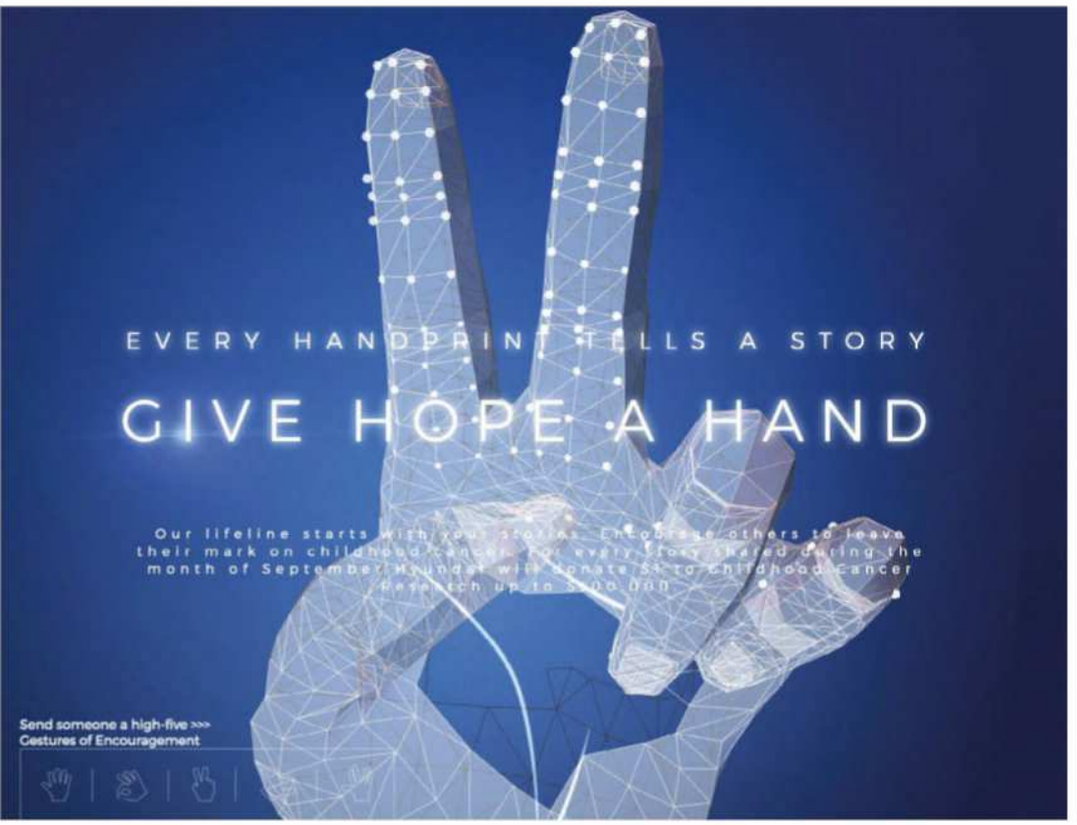
A model's geometry defines the vertices or points of an object. The faces created by these vertices are also defined, which will be filled by a material. Geometry can be primitives such as spheres, cylinders and cubes. It can be defined manually through code or as custom shapes. It can also be loaded from external files exported from 3D modelling software.

Materials

Materials are used to texture the faces of an object's geometry. Three.js comes with several materials, such as Normal, Basic, Lambert, Phong and Shader. Attributes include things such as textures maps, colour, opacity, roughness and enabling morphTargets for animations.

Mesh

A mesh is an object that combines geometry, and applies a material to it. Once created this can then be inserted into a scene and moved around. This object is what we would typically think of as a physical object because it now has enough information to be seen.



TIPS
3D realism

1
Use layers

Adding foreground, mid and background objects will add depth and enhance the composition of your scene. Camera motion with these layers of depth will really make your animation stand out.

2
Use Depth of Field

In film and photography lenses are used to great effect. By narrowing your focus to your area of interest you can capture this look. You can use post-processing shaders to help, based on distance form camera. Or you can simply blur out textures and objects manually at depths that are close or far away, depending on your scene

3
Use ambient effects

Although they can come at a cost of processing power, ambient effects are a great way to add depth and interest to your scene. Dust particles, film grain or noise, a thin layer of smoke or steam rising are all great details that make the scene really come to life.

4
Use environmental maps

Seeing the world around the objects through reflections is another way to add realism to metallic and reflective surfaces. A wet street would reflect street lights or passing cars; bright energy beams reflect off even the most matte surfaces. You can set environment maps right in the properties of most 3D materials.

Your first 3D scene The best way to learn is by seeing. Get the code from filesilo.co.uk

1. Get HTML ready

To begin, you need somewhere to see your 3D scene. Set up a basic HTML file and include a link to Three.js, either hosted externally or add it from the Three.js repository here: **https://github.com/mrdoob/Three.js**.
Note: This code has been tested on the latest release of Three.js v91.

2. Add a scene, camera and renderer

To start add a scene, which will contain all the objects. Then add the camera. The first attribute is the field of view. The second is the aspect ratio. You can also define the near and far clipping planes in the third and fourth attributes. Last, add the renderer, which handles drawing the scene to the canvas.

3. Adding objects

Three.js includes a number of primitive geometries to get you started. Instead of the typical cube or sphere, we'll try out a Torus Knot. To create one, we need a TorusKnotBufferGeometry. The first two attributes are torus radius and tube radius. The next two are the tube segments and radial segments, which determine the amount of divisions or faces will be made along the surface. The more segments a curved surface has the smoother the curve will appear. The last two values are for the times the geometry winds around the axis of symmetry and interior torus circle. Playing with these last variables can generate some very cool results.

In addition to the geometry, we need a material. Let's use a MeshNormalMaterial first. This way we can see the object without needing to light it.

The third thing we need is a Mesh created using the geometry and material we defined.
Note: By default, when we call scene.add(), the object is placed at (0,0,0). To avoid the camera and the object being in the same location we can simply move the camera back a little.

4. Add animation render loop

The animation loop, sometimes called a render loop, is called ideally 60 times per second. Film runs at 24 frames per second (FPS) which is fast enough to trick the eye into seeing constant motion without interruption. In computer animation we aim for at least 30 FPS but ideally 60 FPS to achieve the same result. This ensures very smooth visual performance even if frames are dropped periodically.

We bind this animation loop to the requestAnimationFrame function, which does two things. First, it ensures the browser is ready to render the next frame. It also means animations can pause rendering when the user is no longer viewing that browser tab.

Add the code you find on FileSilo to render the scene in an animation loop. Now when you run your code you should see a nice torus knot object rotating in the centre of the scene.

5. Apply light reactive materials

We used normal materials initially so you could see the object's shape without the need for lights. You will rarely use that material, except for testing purposes, so let's add some real materials next. Let's use a standard material. Tweak the size and position of the object as well to give us some room for next steps.

Update your object code block to look like the code on **FileSilo.co.uk**.

6. Add lights

We can't see the objects anymore if we rerun the code now, because there are no lights in our scene. Let's add some lights. An ambient and two spots. Now you have something that looks much better. A smooth metallic object with highlights and shadows along its curves.

7. Apply textures

Textures create the surface look that skins the mesh. Let's try adding an image texture. We use the built-in TextureLoader class to load the external image file. We can set the X and Y (width, height) wrapping style of the texture as well as the number of times it repeats as it wraps the mesh.

Note: We added in some new attributes for 'map' (which is the main image texture), as well as roughness and metalness values.

You should end up with a nice detailed texture surface now that could pass for a metallic or cool polished marble style look, depending on your image texture.

There is much more you can do to continue adding to this, including specular environmental maps, and bump maps. You can also load more complex models created in external 3D applications.

The different types of light for 3D scenes

Lights are what allow us to see our objects in 3D. Light rays, like in the real-world, emit from a source and interact with objects, absorbing, passing through and reflecting, depending on the surface it interacts with. We can see texture details, shading, intensity and colour.

Spot light

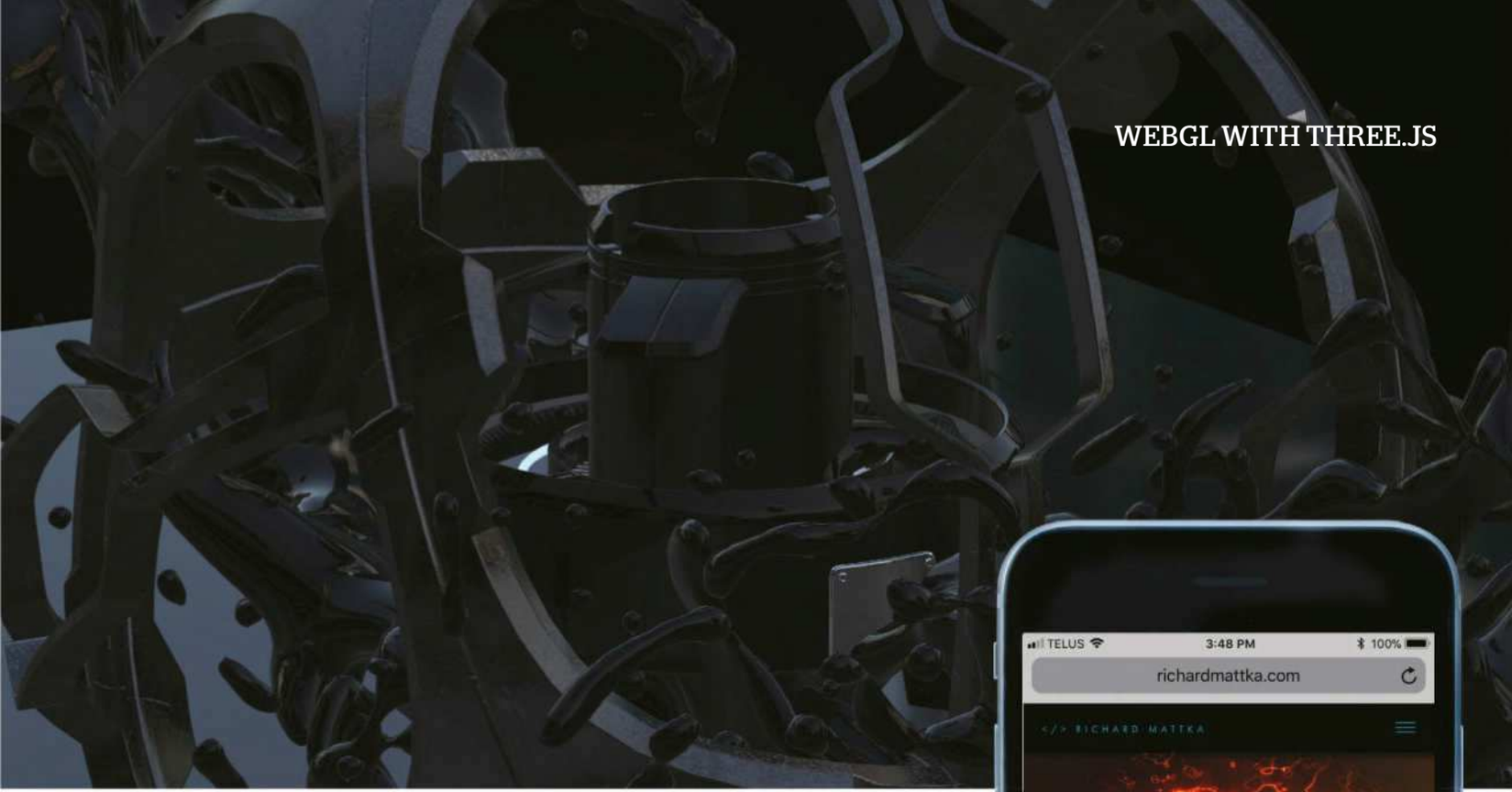
Spot lights emit their light from a single source point in one direction, following a cone shape that spreads the further it gets from the source. So if you like that effect, you can position this light close enough to see the circular pattern it creates on a surface, much like a spot light at the theatre.

Point light

Point lights emit from a single point in all directions. Cool for little fireflies or magical lights that ambiently move in a scene. Or more practically, a bar light bulb.

Directional light

Directional lights emit in a specific direction as though it is infinitely far away. The light rays are all parallel instead of a cone shape. This is much like sunlight in a scene. The sun is far enough away that its rays are essentially infinite and parallel in a scene. Use this light if you want to emulate sunlight.



How to create realistic details

Use some of these techniques to add more realism to scenes

Shadows

One key element for realistic scenes in 3D are shadows. To add shadows to your scene we need to tell the renderer to use shadows. Add this code just after declaring the renderer:

```
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.
PCFSoftShadowMap; // default THREE.
PCFShadowMap
```

Next, update the spot lights. Add this after declaring each light:

```
light.shadow.mapSize.width = 2048;
light.shadow.mapSize.height = 2048;
```

Next, add in the object's ability to cast a shadow after you declare it:

```
object.castShadow = true;
```

Last, if you don't have one add in a ground plane or something to cast shadows onto. Note: We set its property to receive shadows and we use a physical material (StandardMaterial) for optimal realism with shadows and shading.

```
var planeGeometry = new THREE.
PlaneBufferGeometry( 50, 50, 32, 32 );
var planeMaterial = new THREE.
MeshStandardMaterial( { color: 0xaaffaa }
);
var plane = new THREE.Mesh( planeGeometry,
planeMaterial );
plane.rotation.x=Math.PI/180*-90;
plane.position.y=-2;
plane.receiveShadow = true;
scene.add( plane );
```

If you rerun your tutorial code you should see your new shadows.

Bump maps

Bumps maps add amazing surface detail. They are best suited to fine details, like scratches, ridges and fine granular detail. You can use them to create complexity that would require thousands of vertices and faces otherwise. To add bump maps, simply export a bump map from your 3D modelling tool as an image and assign it like any other map, targeting the property called bumpMap.

```
var sphereMaterial = new THREE.
MeshStandardMaterial( { map:texture,
bumpMap:textureBump,  color: 0xeeddff,
roughness:.9, metalness:.5 }  );
```

One thing to note is that bump maps only simulate the changes in surface definition. The underlying geometry is unchanged. So things like shadows cast by the object would not reflect that detail. They also are not specifically mapped to the model, so they may not look great at all angles, depending on the effect you want.

Normal maps

Normal maps are better than bump maps in that they orient to the model. Like bump maps they fake the geometry changes through altering the normals that define how light is reflected from the material. Because they are not tillable textures like bump maps typically are, they are tricker to edit and are usually created in your 3D app when you export. They are applied like bump maps, using the normalMap.

Displacement maps

These maps are the route to go for surface details that need to be altered in a way that affects the underlying geometry. So terrain maps work like this, rising sand dunes or mountains that can be applied to an otherwise flat plane. If you have heavy surface lifting to be done, try to use displacement maps. Use the dispalcementMap attribute of the material when assigning.

"Normal maps are better than bump maps in that they orient to the model. Like bump maps they fake the geometry changes through altering the normals that define how light is reflected from the material"

TIPS
3D models

1
Optimise your model

Reducing your model to the lowest number or vertices and faces is always going to improve the performance and give more cycles to work with. Remove interior geometry if it's not visible, remove anything the camera will never see. Use optimisation tools like MeshLab (**http://www.meshlab.net**) for best results.

2
Bake in detail

Incredible amounts of detail can be simply baked into the texture that covers the mesh. The earth from space looks like a sphere, and the textures can be highly detailed, giving the illusion of terrain and depth, where the surface is actually smooth. Multiple textures, or reusing geometry at slightly different scales and transparency can yield beautiful results.

3
Bump maps for surface details

Bump maps provide incredible detail by using a texture instead of a model. A ridged lid with hundreds of grooves for gripping can be easily baked into a bump map, and applied to a simple cylinder. Jagged alien world surfaces can be defined in the bump map texture in 3D apps and save you thousands of vertices and faces.

4
Level of detail and primitives

If your model is only ever seen far away from the camera, you don't need thousands of vertices to define it. Consider a simplified model for far away objects and more complex models for up close work. Use built-in primitives like cubes, spheres and cylinders or even planes with flat textures on them when objects are far away.

Loading models
The easy way to export models to Three.js

For more complex models, characters and scenes, you need to use a 3D modelling application and then import your models into Three.js...

Exporting models

There are many great 3D modelling applications including Maya, Cinema 4D, CLARA.IO and Blender. You can use any that works best for you. Blender is free, open-sourced and Three.js comes bundled with a plugin to export directly from Blender into a JSON format which can be loaded into Three.js. You can download Blender at **blender.org**. The version we are going to use is 2.79.

The exporter can be found in the utils directory on GitHub here: **https://github.com/mrdoob/three.js/tree/dev/utils/exporters/blender**. Copy the io_three folder into the scripts/addons folder of your Blender installation. There are more details if you need them on the Three.js plugin page.

Inside Blender, go into the File menu to User Preferences then select the Add-ons tab. Then check Import-Export>Three.js Format. You can then use it anytime via File>Export>Three.js to generate the JSON format.

Loading models in Three.js

You have your JSON file and now you need to load it into your Three.js app. You can do this using the JSONLoader class. Just like the TextureLoader, you call load() on the object, passing in the URL to the file as well as a callback method.

```
var loader = new THREE.JSONLoader();
loader.load('lightsaber.json', loaded);
```

The callback returns the loaded geometry and any materials that go with it. The materials are an array of materials and you can use them by passing them into the MultiMaterial class. Then you can proceed to create your mesh with the geometry and your material as usual.

```
function loaded(geometry, materials) {
  var material = new THREE.
MultiMaterial(materials);
  mesh = new THREE.Mesh(geometry,
material);
  scene.add(mesh);
}
```

"Blender is free, open-sourced and Three.js comes bundled with a plugin to export directly from Blender into a JSON format which can be loaded into Three.js"

Resources

- **Three.js**
  https://threejs.org
  Loads of examples, code and extensive documentation to get you going.
- **Blender**
  https://www.blender.org
  Blender is a free, open-source 3D application with a Three.js exporter.
- **Turbo Squid**
  https://www.turbosquid.com
  Professional 3D model library with wide range of excellent models.

Tools

- **Clara.io**
  https://clara.io
  3D modelling, animation and rendering app that runs in browser and exports to Three.js.
- **Three.js Editor**
  https://threejs.org/editor
  Browser-based editor built by Three.js, with rendering, import and export tools.
- **MeshLab**
  http://www.meshlab.net
  Optimise models, reduce vertices and faces, edit, clean and heal your models.

Environment maps

Environmental maps take your 3D scene to the next level

As highly reflective surfaces, metals and plastics reveal the world around them. To map that world onto the mesh, you need to use environmental maps. Even rough, rocky textures often bounce back some of the ambient light around them even if its diffused. To load an environmental map, you need to create a special box for the map, using a classic skybox model. The textures for these can be tricky to setup, but there is plenty of places to source these, and you can make your own. Even standing out in the general environment you want to place your object and photographing the 360 image yourself is great. Once you have the image set you'll load them in like this:

```
var envMap = THREE.ImageUtils.
loadTextureCube( [ 'assets/landing/env1.
jpg', 'assets/landing/env2.jpg', 'assets/
landing/env3.jpg', 'assets/landing/env4.jpg',
'assets/landing/env5.jpg', 'assets/landing/
env6.jpg' ] );
envMap.format = THREE.RGBFormat;
envMap.mapping = THREE.
CubeReflectionMapping;
```

Then simply add to your material properly like other maps like this: `{envMap: envMap}`

Animation

Bring scenes to life with eye-catching animation

Flickering and moving lights, moving the camera, and transforming the objects in 3D space all add life to a scene. Watching lights and shadows dance off objects, and seeing environments reflect in their surface add depth and realism.

Rigged models

In 3D modelling applications such as Blender, you are able to rig your models. This means joints and a system of bones can be created to underlay a mesh. With good design, complex models can be created that move realistically as joints are rotated and moved.

Animation sequences

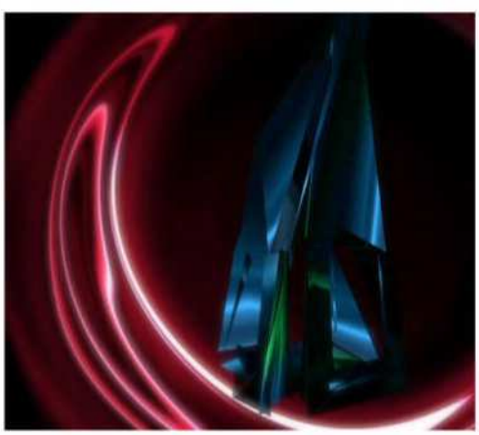
You can create animation sequences as well, that is pre-defined clips or coordinated motions, such as wave, walk, or jump. In your export process you can also export these animations. Three.js is able to use what are called Morph Targets when loading models '{"MorphTargets":true}' and expose these through its built-in animation class. You can then run these animations using a play/pause approach, even adjusting speed as needed to tweak out the animation look.

Graphic shaders

Post-processing effects and animations with shaders

Cool colour effects, blurs, glitches and complete animations. These are highly performant and can be applied as a custom material called a ShaderMaterial instead of image textures. They can also be used as post-processing effects applied on top of your entire scene. Vertex shaders get in a step ahead of Frag shaders and can be used to manipulate the position data of vertices. This allows you to manipulate the model itself on a granular level via code in a highly optimised way. Particle effects, swarms of objects or points, clouds and volumetric simulations are just a few possibilities.

Combining conventional models and techniques with graphic shaders can open a world of options to enhance your scene.



TIPS

Performance boosting

1

Use performance stats tools

Three.js and other WebGL libraries include performance measuring tools (stats.js). You want to aim for 60FPS as often as possible. Smooth performance depends on minimising per frame computations. Keep an eye on your performance and the end result will be silky smooth.

2

Simplify geometry

Use built-in geometries where you can. If it's a planet try using a sphere and bake in surface details into the bump map and textures. See how much you can reduce the segments or polygons as well and maintain a good look. Also try to use buffer geometry and reuse geometry whenever possible – this will reduce draw calls and boost performance.

3

Reduce lights

Light computations are expensive. Shadows as well. Try to reduce the calculations required to cast rays from sources to objects, by using the minimum lights required to light the scene. Think like a lighting designer. Start in a dark room and turn on a light one at a time until you get it.

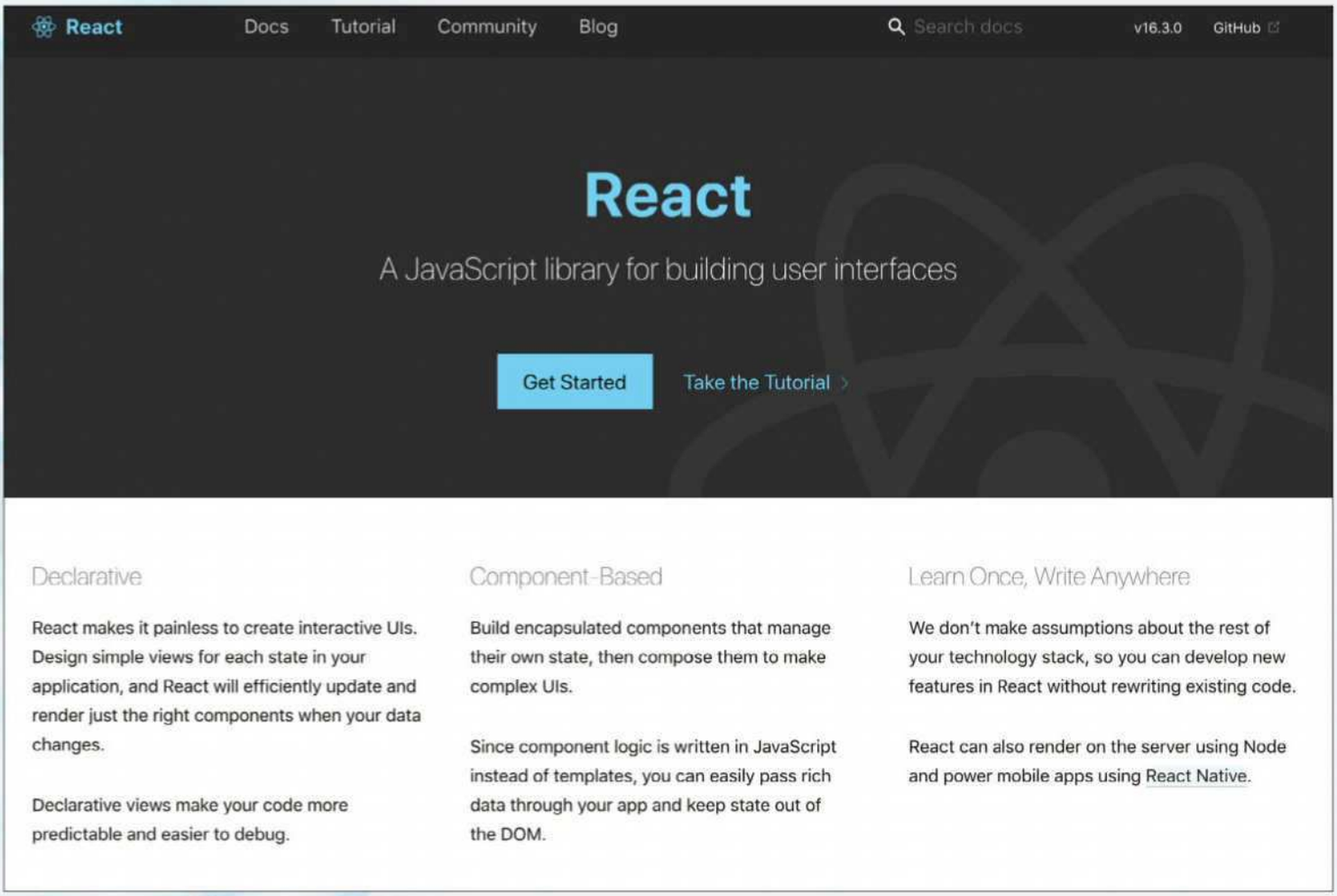
4

Reuse and simplify materials

Reuse your materials as much as possible. Don't reload the same material over and assign it to new variables, try to make the best reuse possible. Composite materials ahead of time in a canvas if it needs to be dynamic, then use that as a single texture instead of multiple ones.

BUILD COMPONENTS WITH REACT AND JSX

REACT IS ONE OF THE MOST POPULAR FRONT-END FRAMEWORKS OUT THERE, AND ITS JSX SYNTAX OFFERS A POWERFUL WAY TO AUTHOR REUSABLE COMPONENTS WITH COMPLEX FUNCTIONALITY WHICH REMAINS EASILY READABLE...



MEET REACT

Unless you've been marooned on a desert island for the last few years, you've probably heard of React. Created by Facebook in 2013, it remains to date arguably the most popular front-end framework. Like most other current frameworks, React enables you to create modular web apps using 'components;, which makes it much easier to build and maintain large scale apps. Unlike some other popular frameworks, like Angular, React is focused only on the user interface (the 'view' in an MVC framework).

In years gone by, one of the major challenges with web development was managing the complexity as projects grew larger. The lack of a straightforward module system, and difficulty with scoping, meant that it was very difficult to create and distribute reusable code. As a result, enabling the creation of components is a cornerstone of modern web frameworks. The key purpose of developing in this way is to let you split your app into reusable pieces that you can think about independently. In practice, the end goal of these custom components is to let you encapsulate complex behaviour within your own custom elements that can be embedded within a page. Think of how <video> hides significant underlying implementation, and now imaging creating <my-element> with your own functionality.

These components need to include presentation (HTML/CSS) and functionality (JavaScript) in a single, simple package.

React provides a way of creating components like this, but also provides various other features to help build maintainable, functional and fast user interfaces. Notably, it offers its own approach to rendering that is very efficient, meaning the user interfaces you create should be highly performant. It's also typical that alongside React itself, you'll utilise an extension of JavaScript called JSX, which provides additional syntax to make building components much more straightforward.

Many consider React to have a bit of a learning curve. The difficulty for most developers comes with understanding its component state and props model, which requires you to design and think about data flow in an app slightly differently to how you might have done traditionally. The good news is that, in terms of language syntax, JSX itself is a very straightforward extension of JavaScript. For the most part, what you write will feel very familiar, and you don't have to learn some of the syntactic specifics you'd need to get to grips with if you used Angular. You'll more or less get your head around how JSX works on the first reading.

"THE GOOD NEWS IS THAT, IN TERMS OF LANGUAGE SYNTAX, JSX ITSELF IS A VERY STRAIGHTFORWARD EXTENSION OF JAVASCRIPT"

REACT COMPONENTS

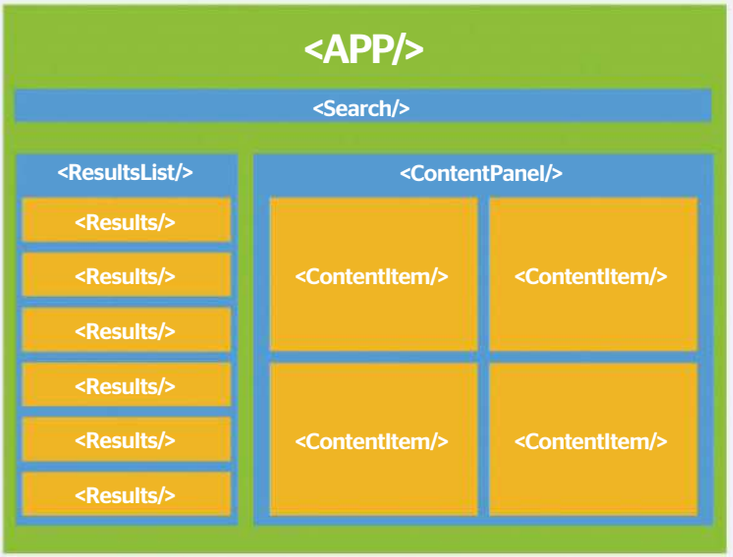
React components are essentially JavaScript classes with a render() function that returns content to display. The library ships with a React.Component class that you extend to create your own custom components.

```
class MyComponent extends
React.Component {
render() {
return (
// ...
);
} }
export default MyComponent;
```

The code above will create a new component that can be instantiated using the element <MyComponent>.

Note that where vanilla elements like are always lowercase, React elements are capitalised.

There are a couple of interesting things going on here. Firstly, the entire presentation of the component is provided by the render() method. This means that your markup and scripting will be combined into a single JavaScript file, rather than keeping them in separate HTML and JS files. It also means that it's a lot easier to define complex behaviour of a component, because you can define exactly what the component displays, conditionally if you like, within the render() function.

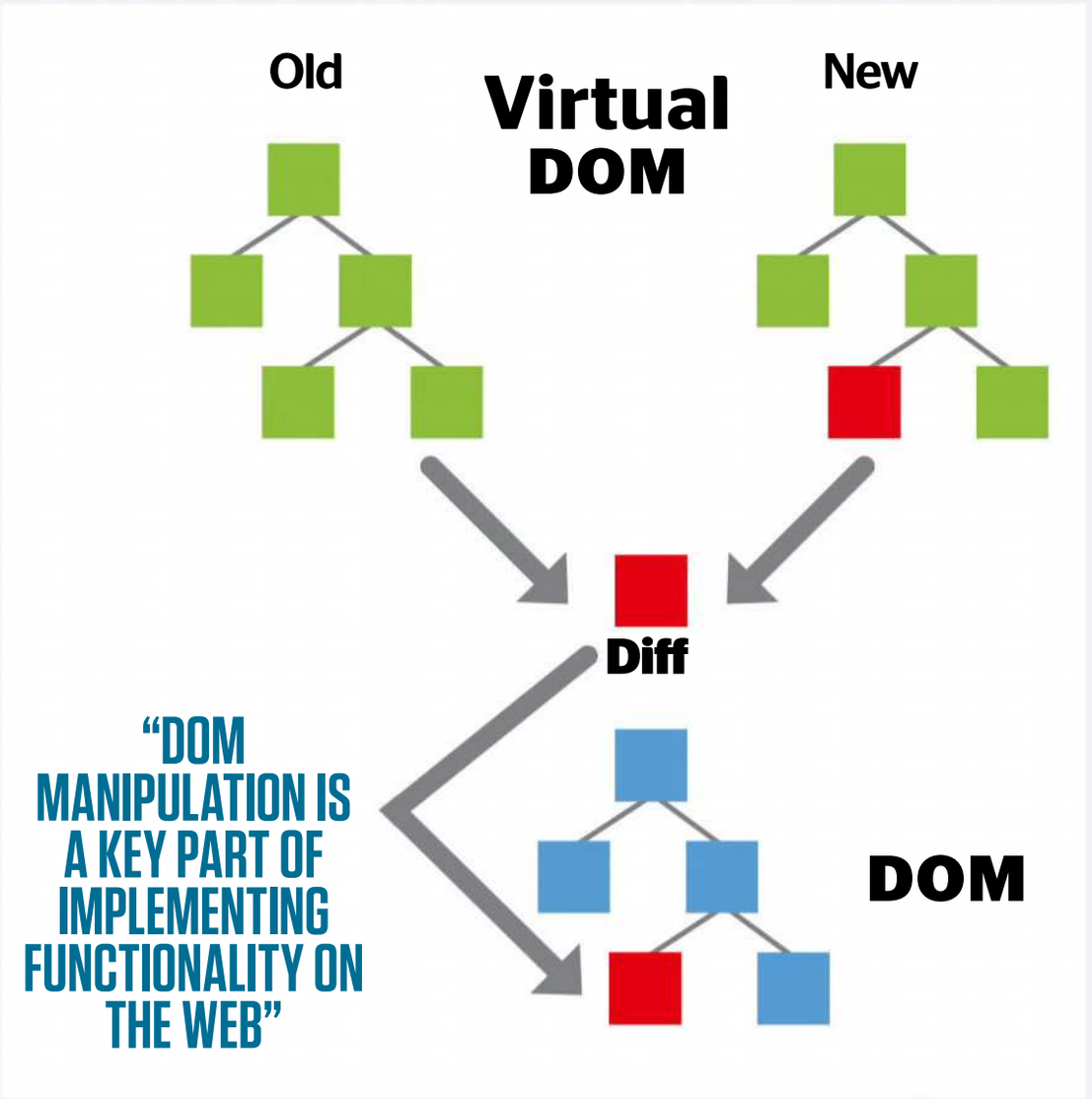


VIRTUAL DOM

DOM manipulation is a key part of implementing functionality on the web, as it's required to change what the user sees on the page. Unfortunately, it's a fairly slow operation to perform. Some front-end frameworks will rebuild the entire DOM when a change is made to an element on the page, which is very inefficient.

React offers a very different solution to this problem. It maintains something called 'virtual DOM', which is a representation of the app's actual DOM. The virtual DOM has all the same objects and properties as the actual DOM, but isn't drawn on the screen. When a change is made to a React element, it rebuilds the entire virtual DOM – but this is much quicker than updating the actual DOM since nothing is actually drawn on the page.

What happens next is the clever part. React compares the new virtual DOM with a snapshot taken before it changed, to determine which objects changed ('diffing'). It then updates these objects, and only these objects, on the actual page. Critically, React will do this for you automatically if certain attributes of a component are changed, so you don't have to manually tell it to re-render components. This process enables React to draw page updates very quickly and efficiently. However, it increases the memory consumption of pages due to the maintenance of a separate virtual DOM tree.



"DOM MANIPULATION IS A KEY PART OF IMPLEMENTING FUNCTIONALITY ON THE WEB"

JSX

Let's think about the render() function in our React component. Using vanilla JavaScript syntax, it's a bit cumbersome to have this return presentation for the component, since we'd need to manually construct a DOM tree.

Therefore, in almost all cases, React components are built using an extension of JavaScript called JSX. JSX is best thought of as a templating language that allows you to include markup in your JavaScript. Let's take a look at a simple component:

```
class MyComponent extends
React.Component {
render() {
return (
<div className="my-div">
This is my React component!
</div>
);
} }
```

When we build our app, the HTML syntax within the render() function is actually compiled to:

```
React.createElement(
div,
{className: 'my-div'},
'This is my React component!'
)
```

When it was first unveiled, some developers felt this JSX syntax represented an anti-pattern since markup and scripts were traditionally kept separated. However, since the component itself is reusable without duplicating code, it doesn't introduce the problems you might expect. Furthermore, it lends itself to a very readable style, since you can quickly see the functionality of a component described clearly in a single file. Also remember that React is still only dealing with presentation, so the scripting you write here is only to support the user interface and won't include any application logic. It's possible to use React without JSX if you want, but it's uncommon to do so.

TIP

You'll notice React uses className instead of class for DOM elements. This helps avoid confusion between JSX's extended syntax and JavaScript's native class keyword.

As well as using built-in elements, we can also use our own custom elements within JSX markup to 'nest' components:

```
return (
  <div className="my-div">
    <MySecondComponent/>
  </div>
);
```

Now consider how this can help us more easily define behaviour for a component. JSX offers a few features that allow us to do this. We can evaluate JavaScript expressions within JSX markup by embedding them in curly brackets. For example:

```
<div>
    {1 + 1}
</div>
```

More practically, this allows us to display variable content held by the component. React components hold data either as 'state' or 'props' (more on this later), which we can evaluate in this way.

For conditional functionality, we can't use if/then clauses within curly brackets, but we can use conditional expressions (x ? y : z). Alternatively, we could place the logic outside the markup and have multiple return clauses for the render() function.

We can also embed function responses within our JSX markup. For example:

```
class MyComponent extends
React.Component {
  renderContent() {
    return
<MySecondComponent/>;    }
  render() {
    return (
      <div className="my-div">
        {this.renderContent()}
      </div>
    );
  } }
```

This enables you to start to break down complex rendering functionality into manageable units. The ability to embed functions also means it's easy to iteratively render components, which is typically done using the map() function.

GETTING STARTED WITH REACT

To get started with React, you'll probably want to use create-react-app, an npm package which sets up a new React app with all the tools you'll need to get started, such as support for JSX.

Firstly, install the latest versions of Node.js and npm from **https://nodejs.org**. Then run at the command line:

```
npx create-react-app
new-react-app
cd new-react-app
npm start
```

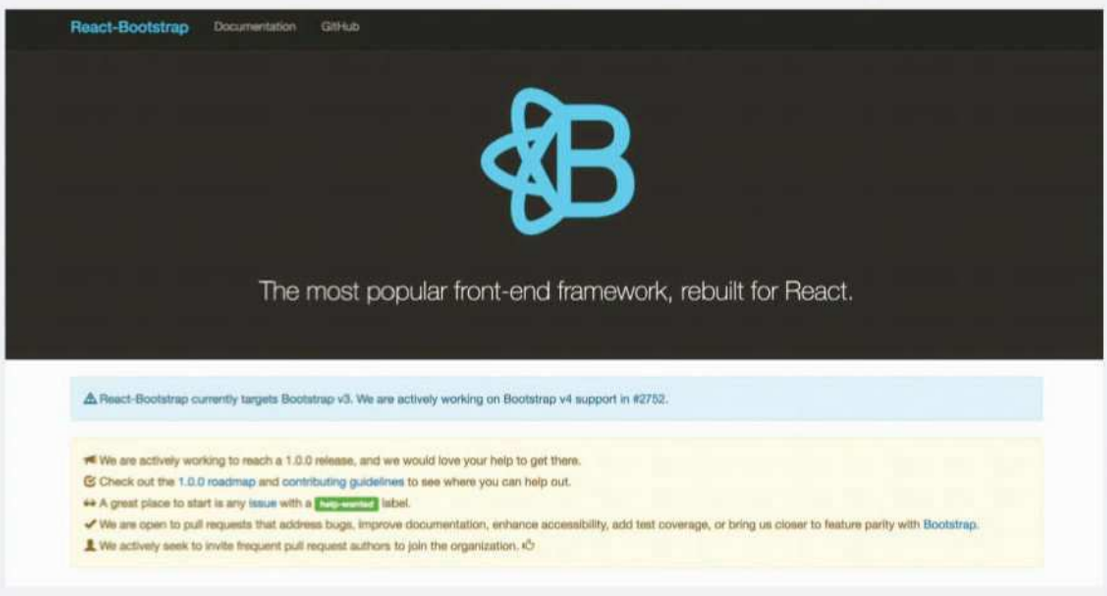
This will kick off a local web server running a new React app. You can modify from there. If you navigate to the src folder, you'll see some of the basics of how a React app is set up. In index.js, which is what the web server runs when started, you'll find an instruction for the React DOM to render an <App/> component:

```
ReactDOM.render(<App />,
document.
getElementById('root'));
```

If you look in App.js, you see the contents of this component, including the render() function with JSX syntax describing what the component should display. This corresponds with what you can see in the browser. Notice how styling is imported as a dependency for the component at the top: `import './App.css';`

STYLING AND JSX

There are a few different ways you can incorporate styling into JSX components, each with their own advantages and disadvantages. The most straightforward is to simply import a CSS file into your component and use className to attach styles to elements in the JSX. You could also use inline styling. You might think this is bad for readability and maintainability, but those issues can be solved by declaring the styles as const objects, which JSX can resolve as variables within the markup. Or you could use styled-jsx (**www.npmjs.com/package/styled-jsx**) which introduces JSX support for a <style> tag within the render() function.



REACT NATIVE

React Native is another offering from Facebook which enables the component model and JSX syntactic style of React to be used for native mobile development. React Native projects create mobile apps which are indistinguishable from those written in Swift, Objective-C or Java.

This can reduce the learning curve for developers who are familiar with React and JSX to begin creating native mobile apps. Many companies have separate departments for web and native mobile build since the skillsets were traditionally different, but React Native offers the possibility for a single skillset to be used for all front-end development.

This shouldn't be confused with a single code base for web and native development – that isn't the objective of the framework, and React Native apps will differ from React apps. Rather, Facebook aims to allow developers to learn one style of development and use it across every platform. It is, however, possible to share certain portions of code and application logic between React and React Native projects.

React Native also takes care of the differentiation between iOS and Android for you, so you don't need to maintain separate code bases for each mobile platform.

And if you're feeling adventurous, as well as React Native, Facebook have also launched React VR which allows you to use similar syntax for VR development. Exciting times.

PROPS, STATE AND EVENTS

As we've alluded to, the real power of React components come from React's model of props and state. Props are a means of passing information into a component when it is created. Let's refresh our example component:

```
class MyComponent extends
React.Component {
render() {
return (
<div>
<h1>{this.props.title}</h1>
{this.props.content}
</div>
);
}
}
```

Here, we're using JSX's JavaScript expression evaluation to use the this.props.* variables. But where do they come from? This is very straightforward. We can pass the props as attributes when we use our custom element. For example:

```
<MyComponent title="Hello
World!" content="This is my
first React component."/>
```

What this means is that in a hierarchical tree of components (which is how React apps are ultimately composed), the parent component can specify content or behaviour of the child. The child component cannot modify its own props, but if the parent component makes a change to them, it will re-render the child. React does this automatically, so you can update how the component is displayed to the user just by modifying its props. This means you can build reusable components which support data flow downward through a component hierarchy.

This is useful, but it isn't all we need. Most components will also want to encapsulate some internal implementation details, which in React terms we call 'state'. State can be set and modified within a component: class MyComponent extends React.

```
Component {
constructor(props) {
super(props);
this.state = {
title: "Hello World!",
content: "This is my first
React component."
}
}
render() {
return (
<div>
<h1>{this.state.title}</h1>
{this.state.content}
</div>
);
}
}
```

In this example, we've set the components state initially in its constructor (the function called when it's created). We can modify it later on with the this.setState() function. Changes in state also re-render the component.

So now we have two ways to handle data in React – one to store it within a component, (state) and one to pass it down into a new child component (props).

Generally you should use state sparingly and aim for most of your components to be stateless.

State is internal to the component and can't be accessed elsewhere, including in a component's parent. This means that for most apps we need something else to handle interaction between child and parent components. Let's say we have a child component with a button, and we want the button to trigger some action in the parent component. What we need is an event which the parent component can receive and respond to.

With JSX, we can pass functions as expressions to an event handler, like this code:

```
<button>
onClick={handleClick}>Click me!
</button>
```

We can then define a handleClick() function which is called when the button is clicked. But what if we want our custom component to respond to onClick events?

```
class ParentComponent extends
React.Component
{
  render() {
    return (
      <ChildComponent
onClick={this.handleChildClick}
label="Click me!/>
    );
  }
  handleChildClick() {
    // …
  }
}
```

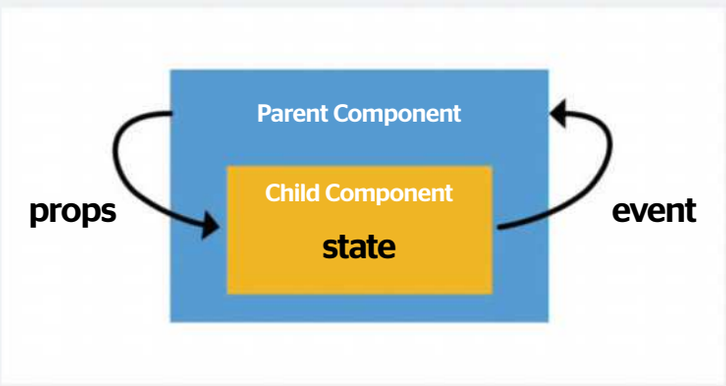
What we're actually doing here is passing the parent component's handleChildClick() function to the child component as a prop called onClick. The child component can then call the function using this prop:

```
class ChildComponent extends
React.Component {
  render() {
    return (
      <div>
        Here is a button:
        <button onClick={this.
props.onClick}>{this.props.
label}</button>
      </div>
    );
  }
}
```

Between the ability to pass props to a component, have it maintain its own state, and allowing events to be passed to parent components, you should have most of what you need to begin developing user interfaces in React, taking advantage of JSX syntax. To understand how to build more complex applications with this model, it's worth looking into the Flux design pattern which Facebook have developed for use with React.

TIP

You might have heard some controversy around React's licensing agreement. Prior to 2017, React had a unique license which some felt granted inappropriate empowerment to Facebook. However, this is no longer an issue as React now uses the MIT license.



CREATE THE IMPOSSIBLE

www.photoshopcreative.co.uk

EVERY ISSUE • DOWNLOAD FREE RESOURCES

Photoshop creative

Available from all good newsagents and supermarkets

ON SALE NOW

• Striking imagery • Step-by-step guides • Essential tutorials

BUY YOUR ISSUE TODAY

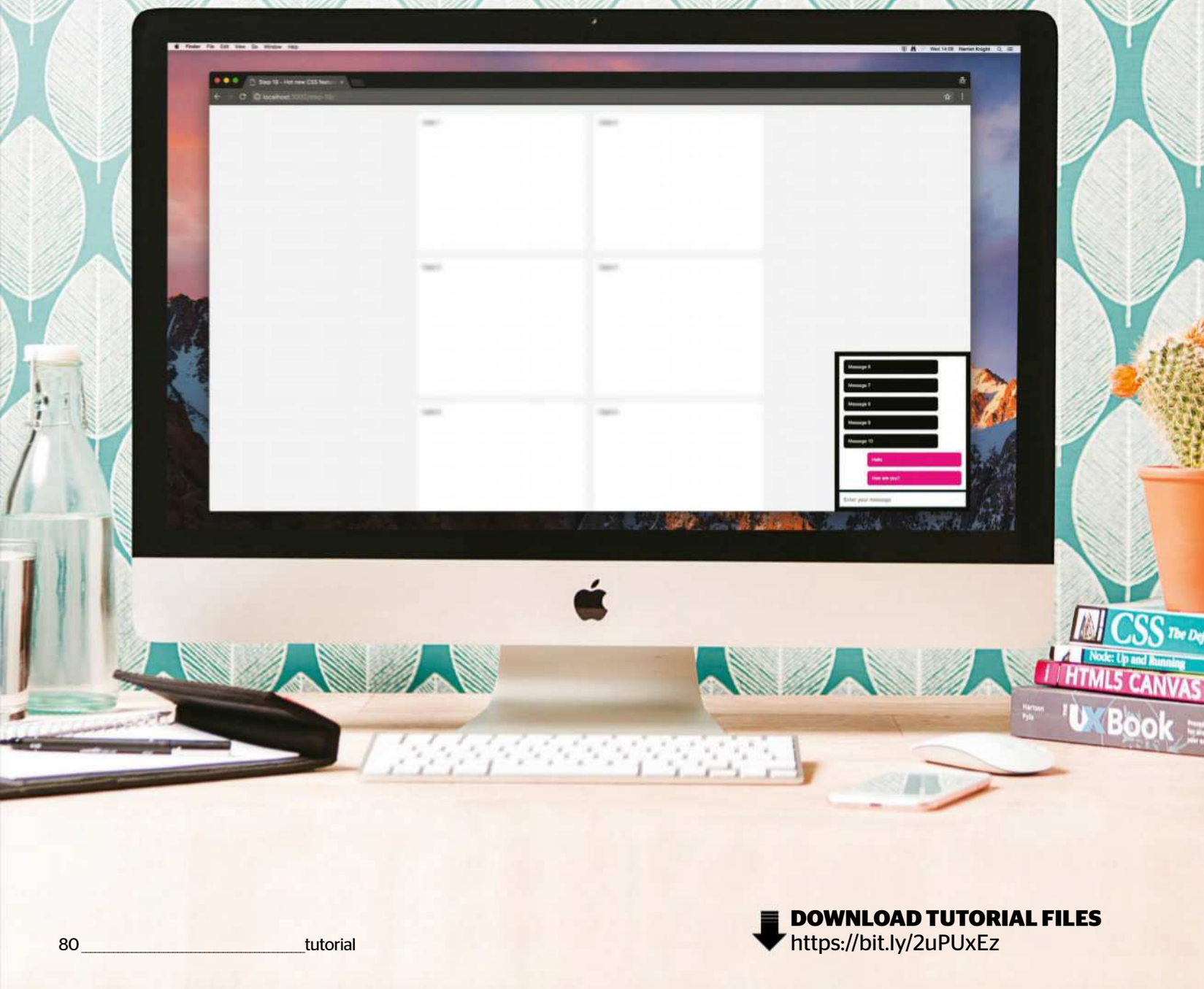
Print edition available at www.myfavouritemagazines.co.uk

Digital edition available for iOS and Android

facebook.com/PhotoshopCreative twitter.com/PshopCreative

5 new CSS features & how to use them

Check out five new properties coming to CSS, find out what they do and how to put them to practical use in your projects



DOWNLOAD TUTORIAL FILES
https://bit.ly/2uPUxEz

We're going to be creating the layout for a page that includes a newsfeed and a little chat box in the corner. To do this we'll leverage the following features to create a better experience and work around some issues. You'll need Chrome 65+ or Firefox 59+.

- `display: contents;`
- `@supports(...) {...} feature queries`
- `overscroll-behavior: contain;`
- `:focus-within and :placeholder-shown pseudo-classes`
- `contain: paint;`

There is a repo to accompany this tutorial which contains the step-by-step code: **https://github.com/danielcrisp/hot-new-css-features**

1. The HTML for the newsfeed

First we need to set up some super simple, repeating markup for our newsfeed. Let's create a '.container' div with an unordered list inside. Give the '<ul>' the class of '.feed', then create 10 list-items each containing a div with the '.card' class and the text Card 1, Card 2 etc. Finally create another list-item in between 5 and 6 with a class of '.nested' – this will be helpful later – and add a '<ul>' inside with three list-items using the text Card A, Card B etc.

```
<body>
  <div class="container">
  <ul class="feed">
  <li><div class="card">Card 1</div></li>
  <li><div class="card">Card 2</div></li>
  <li><div class="card">Card 3</div></li>
  <li><div class="card">Card 4</div></li>
  <li><div class="card">Card 5</div></li>
     <li class="nested">
  <ul>
  <li><div class="card">Card A</div></li>
  <li><div class="card">Card B</div></li>
  <li><div class="card">Card C</div></li>
  </ul>
  </li>
  <li><div class="card">Card 6</div></li>
  <li><div class="card">Card 7</div></li>
  <li><div class="card">Card 8</div></li>
  <li><div class="card">Card 9</div></li>
  <li><div class="card">Card 10</div></li>
  </ul>
 </div>
</body>
```

2. Style the newsfeed

Now we need to add some quick styles so that this starts to look more like a newsfeed. First we can give '<body>' a subtle grey background colour. Then give '.container' a max-width of 800px and use 'margin: 0 auto;' to centre align it. Let's also give '.card' a white background, 10px of padding and margin and finally a min-height of 300px – this should give us enough to make the page scrollable. Lastly we'll sprinkle some flexbox magic on the '.feed' to make the items flow nicely with two cards per row.

```
.feed {
 display: flex;
 flex-wrap: wrap;
}
.feed li {
 flex: 1 0 50%;
}
```

3. display: contents to the rescue

If you scroll down the list you'll notice that our cards in the nested list, Card A - C, are causing some layout problems. Ideally we'd like them to flow in with the rest of the cards but they are all stuck together in one block. The reason for this is that a flex container – which is created using 'display: flex' – only makes its immediate children, ie the list-items, into flex items.

Now, normally the only way of fixing this is to change the markup, but let's pretend that you don't have that luxury. Perhaps the newsfeed markup is generated by a third-party script or it's legacy code that you're only trying to reskin. So how can we fix this?

Meet 'display: contents'. This little one-liner essentially makes an element behave as if it isn't there. You still see the descendants of the element but the element itself doesn't affect the layout.

Because we're pretending we can't change the markup (just for this step) we can be a bit smart about this and make the '.card' elements the flex items and almost entirely ignore the list markup.

First remove the existing '.feed li' class and then use 'display: contents' for both '<ul>' and '<li>' elements:

```
.feed ul,
.feed li {
 display: contents;
}
```

Now you should see the cards flowing in order, but we've lost the sizing. Fix that by adding the flex property to the '.card' instead:

```
.card {
 flex: 1 0 40%;
}
```

Tada! Our cards are now using the wonders of flexbox as if their structural unordered list markup doesn't exist.

As a side note you might be wondering why the flex-basis value is set to 40%. This is because the '.card' class has a margin, which isn't included in the width calculation as you would expect when using 'box-sizing: border-box'.

As a side note you might be wondering why the flex-basis value is set to 40%. This is because the '.card' class has a margin, which isn't included in the width calculation as you would expect it to when using 'box-sizing: border-box'.

So to work around this we just need to set the flex-basis high enough to trigger the wrapping at the necessary point and flexbox will fill the remaining space automatically.

4. Perfect solution, right?

Although 'display: contents' does exactly what we need, it is still only at Working Draft status with W3C. And in Chrome support only arrived in version 65 released March 6th, 2018. Incredibly Firefox has had support since April 6th, 2015!

If you disable the style in DevTools you'll see that our changes have made a bit of a mess with the layout when 'display: contents' isn't applied. So what can we do about this? Time for our next new feature – feature queries. They work just like media queries but they allow you to ask the browser – using CSS alone – if a property and value expression is supported. If they are the styles contained inside the query will be applied.

Now, let's move our 'display: contents' styles into a feature query.

```
@supports (display: contents) {
 .feed ul,
 .feed li {
   display: contents;
 }
 .card {
   flex: 1 0 40%;
 }
}
```

5. Using 'not' for a cleaner outcome

Normally in this kind of progressive enhancement scenario we'd use the query to add the new styles, but it would also have to disable some of the original styles necessary for the fallback layout.

However you might decide that because support for feature queries is pretty good (if you ignore IE) you actually want to use the feature query 'not' operator.

It works just like you'd expect, so we can re-apply our original flex property to the list-items when 'display: contents' is not supported:

```
@supports not (display: contents) {
 .feed li {
   flex: 1 0 50%;
 }
}
```

Inside the 'not' query we can add some styles so that the '.nested' items basically re-apply what was being inherited by using 'display: contents'.

```
feed li.nested {
 flex-basis: 100%;
}
.feed li.nested ul {
  display: flex;
  flex-wrap: wrap;
}
```

6. Taking it one step further

You can already see the potential of feature queries, but the really cool thing is that you can combine expressions using the three available operators: 'and', 'or' and 'not'. Perhaps we could also check for display: flex support and then add a float-based fallback. We're not going to do that here, but if we were we'd first modify the two feature queries like so:

```
@supports (display: flex) and (display:
contents) {
 ...
}
@supports (display: flex) and (not (display:
```

```
contents)) {
 ...
}
```

As a bonus you can also test for custom properties support like this:

```
@supports (--foo: green) {
 ...
}
```

7. Adding a chat box

Now we have a beautiful newsfeed in place, let's add a little chat box that is fixed to the bottom right of the screen. We'll need a list of messages and a text field for the user to enter their message. Add this block just after the opening '<body>' tag:

```
<div class="chat">
  <div class="messages">
  <ul>
<li><div class="message">Message 1</div></
li>
<li><div class="message">Message 2</div></
li>
<li><div class="message">Message 3</div></
li>
<li><div class="message">Message 4</div></
li>
<li><div class="message">Message 5</div></
li>
<li><div class="message">Message 6</div></
li>
<li><div class="message">Message 7</div></
li>
<li><div class="message">Message 8</div></
li>
<li><div class="message">Message 9</div></
li>
<li><div class="message">Message 10</div></
li>
  </ul>
  </div>
  <input type="text" class="input">
 </div>
```

8. Styling the chat box

Time to quickly add some styling so it looks half decent.

```
.chat {
 background: #fff;
 border: 10px solid #000;
 bottom: 0;
 font-size: 10px;
 position: fixed;
 right: 0;
 width: 300px;
 z-index: 10;
}
.messages {
 border-bottom: 5px solid #000;
 overflow: auto;
 padding: 10px;
 max-height: 300px;
}
.message {
 background: #000;
 border-radius: 5px;
 color: #fff;
 margin: 0 20% 10px 0;
 padding: 10px;
}
.messages li:last-child .message {
 margin-bottom: 0;
}
.input {
 border: none;
 display: block;
 padding: 10px;
 width: 100%;
}
```

9. Scroll chaining

Hopefully now you'll have a little chat box with a scrollable list of messages sitting on top of your newsfeed. Great. But have you ever noticed what happens when you scroll inside a nested area and you reach the end of its scrollable range? Try it. Scroll all the way to the end of the messages and you'll notice that the page itself will start scrolling instead. This is called scroll chaining.

It's not a big deal in our example but in some cases it might be. We've had to work around this feature before when creating scrollable areas inside modals or context menus. The dirty fix is to set the '<body>' to 'overflow: hidden', but there is a nice, shiny new CSS property that fixes all of this and it's a simple one-liner. Say hello to 'overscroll-behavior'.

There are three possible values:

- **auto –** the default, which allows scroll chaining
- **contain –** disables scroll chaining
- **none –** disables scroll chaining and other overscroll effects, such as rubberbanding

We can use the shorthand 'overscroll-behavior' or we can target a specific direction with 'overscroll-behavior-x' (or '-y'). Let's add it to our '.messages' class:

```
.messages {
 ...
 overscroll-behavior-y: contain;
 ... }
```

Now try the scrolling again and you'll see that it no longer affects the page's scroll when you reach the end of the content.

This property is also pretty handy if you wanted to implement a pull-to-refresh feature in your PWA, say to refresh the newsfeed. Some browsers, such as Chrome for Android, automatically add their own and until now there has been no easy way to disable it without using JS to cancel events using performance-impacting non-passive handlers.

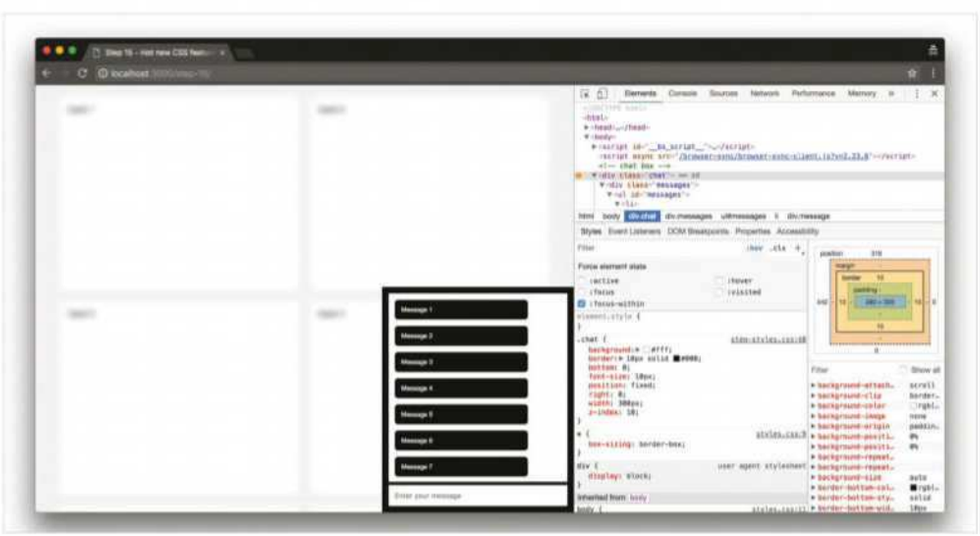
Now you just need to add 'overscroll-behavior: contain' to the viewport-containing element, probably '<body>' or '<html>'.

It's worth noting that this property is not a W3C Standard, rather a proposal by the Web Incubator

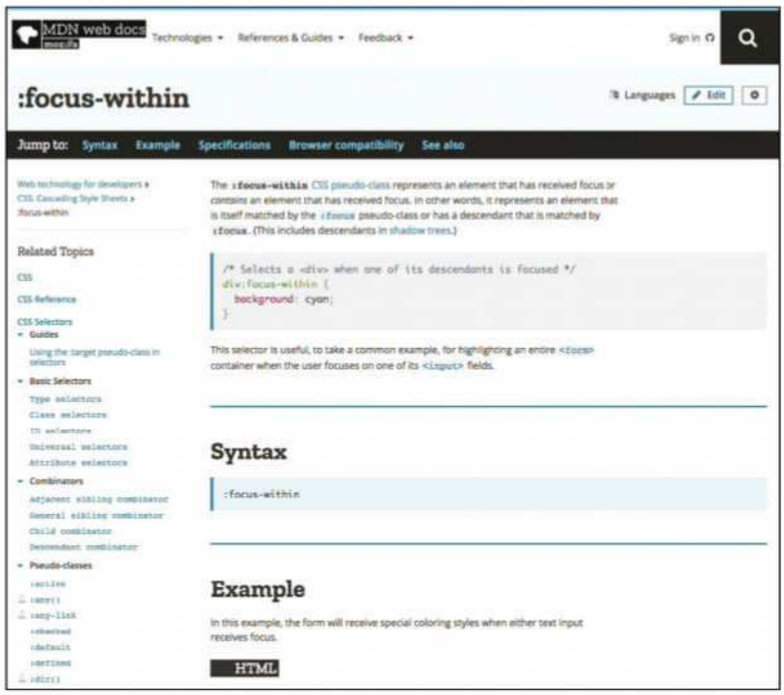
Force focus

To keep the chat box open even when it doesn't have focus you can now force `:focus-within` in Chrome DevTools, just like you can with `:hover`.

Top
The potential of ':focus-within' for UX improvements is exciting, but with great power comes great responsibility!



Right
In DevTools you can select the element and toggle the ':focus-within' state making it easier to inspect



Recommended reading

CSS Containment Module Level 1 - contain: paint;
W3C status - Candidate Recommendation
https://drafts.csswg.org/css-contain-1
https://caniuse.com/#feat=css-containment
https://developers.google.com/web/updates/2016/06/css-containment

CSS Display Module Level 3 - display: contents;
W3C status - Working Draft
https://drafts.csswg.org/css-display
https://caniuse.com/#feat=css-display-contents

CSS Overscroll Behavior Module Level 1 - overscroll-behavior: contain;
W3C status - Unofficial
https://wicg.github.io/overscroll-behavior
https://caniuse.com/#feat=css-overscroll-behavior
https://developers.google.com/web/updates/2017/11/overscroll-behavior

CSS Conditional Rules Module Level 3 - @support(...) {...}
W3C status - Candidate Recommendation
https://www.w3.org/TR/css3-conditional
https://developer.mozilla.org/en-US/docs/Web/CSS/%40supports

CSS Selectors Module Level 4 - :focus-within
W3C status - Working Draft
https://www.w3.org/TR/selectors-4
https://caniuse.com/#search=pseudo-class
https://developer.mozilla.org/en-US/docs/Web/CSS/:focus-within
https://developer.mozilla.org/en-US/docs/Web/CSS/:placeholder-shown

Community Group (WICG). Popular, stable and mature WICG proposals are considered for migration to a W3C Working Group at a later stage.

10. Collapse the chat box when not in use

At the moment the chat box takes up quite a bit of space, which unless we're interacting with it is a bit distracting. Fortunately we can do something about this with a little CSS magic.

But first of all we need to modify our existing styles slightly. By default we want the chat box to be collapsed, so we need to reduce the 'max-height' value in the '.messages' class. While we're there we can also add a transition to the 'max-height' property:

```
.messages {
 ...
 max-height: 25px;
 transition: max-height 500ms; }
```

11. Expand the chat box when it receives focus

So our chat box is currently collapsed and unusable. How can we expand it using only CSS? Time for a new feature – the Focus Container pseudo-class `:focus-within`.

This is just like the old faithful ':focus' pseudo-class selector, but it matches if any of the element's descendants have focus. This is interesting because it is the reverse of how CSS usually works, where you can only select elements based on their ancestors. It opens up some interesting potential.

Adding this class will set the 'max-height' of the '.messages' element to 300px, when anything inside the '.chat' area receives focus.

Remember that an element must accept keyboard or mouse events, or other forms of input, in order to receive focus. Clicking in the `<input>` will do the job.

```
.chat:focus-within .messages {
 max-height: 300px; }
```

12. Taking focus further

This is nice, but what else can we do? How about blurring the newsfeed when the chat box has focus? Well with the help of the subsequent-sibling combinator ('~') we can do this really easily because the markup for the chat box is before the newsfeed markup:

```
.chat:focus-within ~ .container {
 filter: blur(5px); }
```

That's all it takes! We can soften the effect slightly by adding a transition on the filter property to the original '.container' class.

Now we're not suggesting this is a good idea, but we think it's pretty cool that we can do this with CSS alone.

13. Another new pseudo-class

There are a number of new pseudo-classes described in CSS Selectors Module Level 4. Another interesting one is ':placeholder-shown' which matches any input that is showing placeholder text.

At the moment our text field doesn't have any placeholder text so let's add some.

```
<input type="text" class="input"
placeholder="Enter your message">
```

And then immediately after the text field add a helper message to show the user.

```
<div class="prompt">Press enter to send</
div>
```

Now add some styling for the helper message, so that by default it is collapsed.

```
.prompt {
 line-height: 2em;
 max-height: 0;
 overflow: hidden;
 padding: 0 10px;
 text-align: right;
 transition: max-height 500ms; }
```

14. Making the prompt visible

At the moment the prompt is hidden so how can we use ':placeholder-shown' here? Well (most) browsers will display the placeholder text until the field has a value. Just setting focus won't hide the placeholder. This is helpful because we don't want the user to send a blank message anyway, so we can hook into this behaviour to show the prompt only when the user has entered a value.

Of course in the real world we'd need some validation too. We can use the negation pseudo-class ':not' to flip the meaning of ':placeholder-shown' and show the prompt when the placeholder text is not shown, ie the field has a value.

```
.input:not(:placeholder-shown) + .prompt {
 max-height: 2em; }
```

Setting the 'max-height' to double the 10px font-size, using 2em, will expand the block and the text becomes visible. Neat.

Expect to see some clever uses for this seemingly mundane pseudo-class if it makes it through to the final spec.

15. Bringing it to life

We've built the bare bones of a newsfeed with a chat feature using some of the new properties coming to CSS, but at the moment it is lifeless – you can't actually do anything with it. One particularly interesting new feature is CSS Containment, but it is difficult to demo without modifying the DOM. Let's add some super-simple JS to allow us to add messages to our chat box.

First of all we need to add an ID to the '<input>' and the

messages '<ul>'. We can also add a required attribute to the input too.

```
<ul id="messages" ...
<input type="text" id="input" required ...
```

Then create an empty JS file called script.js and add it to the page just before the closing '</body>' tag so we don't have to wait for DOM ready.

```
<script src="script.js"></script>
</body>
```

16. Add a little sprinkle of JavaScript

We need to add an event handler to the '<input>' which listens for an Enter keypress, validates the field, takes the value (if valid) and adds it to the end of the messages list, clears the field and scrolls to the bottom of the messages. Something like this will do the job for our demo but don't use this in production!

```
// Finds the important elements
const input = document.
getElementById('input');
const messages = document.
getElementById('messages');
// Listens for keypress events on the input
input.addEventListener('keypress', (event)
=> {
  // Checks if the key pressed was ENTER
   if (event.keyCode === 13) {
  // Checks the field is valid
  if (input.validity.valid) {
  // Creates a DOM element using the value
  const message = createMessage(input.
value);
  // Appends the new message to the list
  messages.appendChild(message);
  // Clears the field
  input.value = '';
  // Scrolls the messages to the bottom
  messages.parentNode.scrollTop = messages.
parentNode.scrollHeight;
  }
  }
});
// Converts value to string of HTML
function createMessage (value) {
  // Warning: Don't do this in production
without sanitizing the string first!
  return stringToDom(`<li><div
class="message">${value}</div></li>`)
}
// Converts string to real DOM
function stringToDom (string) {
  const template = document.
createElement('template');
  template.innerHTML = string.trim();
  return template.content.firstChild; }
```

Now when you type in the field and press Enter you'll see your message added to the bottom of the element's list.

17. Adding some additional info

In order to demonstrate one of the benefits of the final new CSS property: 'contain' we need to do something a little contrived. We're going to implement a feature that displays the time a new message was sent in a box at the top of the messages list. This will appear when you hover your mouse over a message.

First up we need to add this information to our new messages. We can modify the 'return' line of the 'createMessage' function to add the current time to an attribute.

```
  return stringToDom(`
  <li>
  <div class="message message--mine"
data-timestamp="${new Date().toString()}">
  ${value}
  </div>
  </li>
  `);
```

You might have noticed that new messages have an additional class 'message--mine'. Let's add a quick class so they stand out and align to the right of the list.

```
.message--mine {
 background: #ff2089;
 margin-left: 20%;
 margin-right: 0;
}
```

18. Displaying the timestamp

So our goal is to make the message's timestamp appear at the top of the list of messages, but we need to do it so it is always visible even when the list is scrolled.

First off let's try and render the value using a pseudo-element. We only want to do this for our messages. See below.

```
.message--mine::after {
 content: attr(data-timestamp);
}
```

That should display the timestamp inside the message. Let's modify it slightly so it only appears on ':hover', has a bit of styling and is fixed to the top of the messages area so it is visible even when scrolled.

```
.message--mine:hover::after {
 background: #000;
 color: #ff2089;
 content: attr(data-timestamp);
 left: 0;
 padding: 5px;
 position: fixed;
 top: 0;
 width: 100%;
}
```

Hmm, it sort of works but it is appearing at the top of the page rather than the scrollable area. Let's add 'position: relative' to the '.messages' pane to try and contain it.

```
.messages {
 ...
 position: relative;
 ...
}
```

Ah, it doesn't work either because fixed positioning is anchored to the viewport, not its relative ancestor. Time for our final new property – 'contain'.

19. Containment

CSS Containment is an exciting new proposition. It has a number of options which give you the ability to limit the browser's styling, layout and paint work to a particular element. This is of particular benefit when modifying the DOM. Some changes – even small ones – require the browser to relayout or repaint the entire page, which obviously can be an expensive operation even though browsers work hard to optimise this for us.

Using CSS Containment we can essentially ring-fence parts of the page and say "what happens in here, stays in here". This also works the other way around and the ring-fenced element can be protected from changes made outside.

There are four values for 'contain' each with a different purpose. You can combine them as you require.

- **layout –** ensures that the containing element is totally opaque for layout purposes; nothing outside can affect its internal layout, and vice versa
- **paint –** ensures that the descendants of the containing element don't display outside its bounds
- **style –** ensures that certain properties don't escape the containing element (this sounds like the encapsulation you get with the Shadow DOM, but it really isn't)
- **size –** ensures that the containing element can be laid out without using the size of its descendants

Alternatively you can use one of two keywords as a kind of shorthand:

- **strict** is an alias for 'layout paint style size'
- **content** is an alias for 'layout paint style'

Each of these values are a little opaque so I would recommend reading the spec and playing around with them in DevTools to see what actually happens.

The two important values are 'layout' and 'paint' as they offer performance optimisation opportunities when used within complex web apps that require a lot of DOM manipulation. In our rudimentary demo however we can leverage one of the consequences of using 'contain: paint' to help us with the timestamp positioning.

According to the spec, when using 'paint' the "element acts as a containing block for absolutely positioned and fixed positioned descendants". This means we can set 'contain: paint' on the '.chat' class and the fixed positioning will be based on the card rather than the viewport. You'd get the same effect by using 'transform: translateZ(0)' but containment feels less hacky.

```
.chat {
 ...
  contain: paint;
 ... }
```

20. Wrapping up

That concludes our whistle-stop tour of some new CSS features. Most of them are pretty straightforward, although we'd definitely recommend delving into CSS Containment in more detail. We've only really touched on a small part of it here and it is the kind of feature that sounds like it does something that it actually doesn't – the encapsulation of styling. That said it could prove very helpful, and it is already at the Candidate Recommendation stage, so it's well worth a look.

Special offer for readers in ***North America***

6 issues FREE

When you subscribe*

FREE resource downloads every issue

"*The only magazine you need to design and develop stunning websites*"

Order hotline **+44 (0) 344 848 2852**

Online at **myfavouritemagazines.co.uk/WEDPQ17**

Expires 30 June 2018

***Terms and conditions** This is a US subscription offer. 6 free issues refers to the USA newsstand price of $14.99 for 13 issues being $194.87, compared with $103.20 for a subscription. You will receive 13 issues in a year. You can write to us or call us to cancel your subscription within 14 days of purchase. Payment is non-refundable after the 14 day cancellation period unless exceptional circumstances apply. Your statutory rights are not affected. Prices correct at point of print and subject to change. Full details of the Direct Debit guarantee are available upon request. UK calls will cost the same as other standard fixed line numbers (starting 01 or 02) are included as part of any inclusive or free minutes allowances (if offered by your phone tariff). For full terms and conditions please visit: bit.ly/magtandc Offer ends May 31 2018.

Code dynamic backgrounds

Programmatically generate images for backgrounds and masks with the new CSS Paint API

DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/webdesigner

As developers, we sometimes take CSS for granted. By changing a couple of properties, we can change the look of an entire website. Browser vendors are doing a lot of work on our behalf to get to that point.

But as new CSS features come out, it can be a while until we get to use them for production sites. Users with old browsers will be stuck with a downgraded, progressively enhanced set of styles that do not look as good as those in modern browsers.

Houdini is set to change all that. It is a collection of draft specifications that gives developers access to some of the behind-the-scenes magic provided by CSS. Once all the specifications are complete, developers will be able to use JavaScript to determine how aspects such as layout and style calculations are performed.

While some of these APIs are still in their early stages, the first has now landed in Chrome Stable as of v65. The CSS Paint API enables developers to create images in JavaScript for use wherever an image would normally be used in CSS. At the moment the API is only supported in Google Chrome, but other browsers are working to include it soon.

In this tutorial we will be using the API in a couple of ways. One will create a dynamic background that follows the cursor, while the other uses the image as a mask to shape other elements. The same techniques are used in both scenarios, which goes to show how versatile this new API can be.

1. Install local server

While all the code that we will write has no dependencies, we need to use a package called 'http-server' to serve the files while we are developing. This creates a lightweight server based on the project directory.

Open Terminal and install the server dependency from 'npm'. Then start the server and open it up in Chrome.

```
> npm install
> npm start
```

2. Load a paint worklet

All the work is going to happen inside a paint worklet. A worklet is a small module that performs a specialised job. A paint worklet deals with what the browser renders when it updates the screen.

Open up 'squiggle.html' and load the worklet in the script block. As the feature is not in every browser yet, we also need to check to see if it is supported.

```
if ('paintWorklet' in CSS) {
  CSS.paintWorklet
   .addModule('/scripts/squiggle.js');
}
```

3. Track mouse movement

Once finished, the generated image will orientate itself based on the position of the cursor. We need to track the movements by adding them as custom properties, which we will use in later steps.

Under where we added the module, add an event listener for mouse movement events. When that fires, it records the position as a custom property.

```
document.addEventListener('mousemove',
 event => {
  document.documentElement.style
  .setProperty('--x-position',
  event.clientX);
  document.documentElement.style
  .setProperty('--y-position',
  event.clientY);
});
```

4. Draw in the context

Open up 'scripts/squiggle.js' to see the loaded module. The important method here is 'paint', which runs each time the browser needs to repaint. That can be for anything from a screen resize to another element changing position.

The first parameter here is the context, which is very similar to a canvas context and uses some of the same controls.

Draw a small square in the context to make sure it works correctly.

```
ctx.lineWidth = width;
ctx.strokeStyle = "white";
ctx.rect(0,0,100,100);
ctx.stroke();
```

5. Register the worklet

Before we can start to link up the worklet with CSS, we need to register it with the page. To do that, the 'registerPaint' function will create a mapping for us to refer to it in any CSS file.

Right at the bottom of 'squiggle.js', register the worklet module with the name 'squiggle'. This name will be used to link everything together in the next step.

```
registerPaint('squiggle',
 SquiggleModule);
```

6. Set a background image

Paint worklets create images for CSS to use. These can be used wherever a property can accept an image, similar to how 'url()' or 'linear-gradient()' can be used.

Open up 'styles/squiggle.css' and set the background of the 'test' class to the new worklet. Use an extra background colour to fill in everything that the worklet does not render.

```
background-image: paint(squiggle);
background-color: var(--background-color);
```

7. Set up input properties

In the previous step, we defined the background colour using a custom property, which were defined at the top of the file against ':root' – the <html> element. These are

Keeping it secure

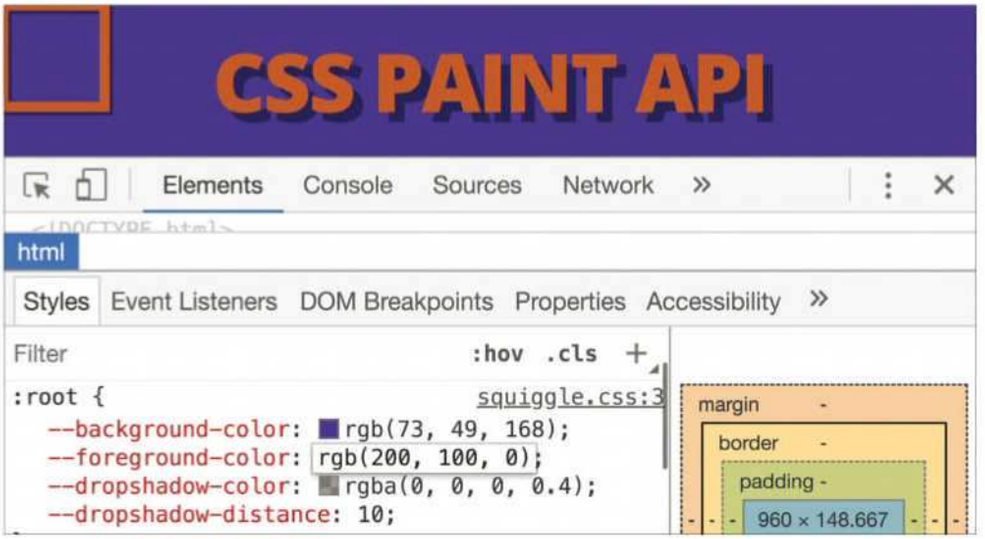
This API – alongside any new ones created in the future – must be called on a script served over HTTPS in order to function. The only exception is when developing on localhost.

Left
Similar to canvases, the context of paint worklets start from the top-left corner. The containing element defines its width and height

great to change values in one place and have them update across the document including paint worklets. All we need to do is tell the worklet which custom properties we are interested in.

At the top of 'scripts/squiggle.js', define the input properties for this worklet. These include the colours used for the page, as well as other inputs we will use in later steps.

```
static get inputProperties() {
  return [
  '--dropshadow-color',
  '--foreground-color',
  '--dropshadow-color',
  '--dropshadow-distance',
  '--x-position',
  '--y-position'
  ];
  }
```



8. Define stoke settings

The third argument on the 'paint' method holds the values for all the input properties at the time of painting. We can use those to change the style of what gets drawn on the canvas.

Replace the 'lineWidth' and 'strokeStyle' assignments from Step 4 with the following. The getter will return an object, which needs converting to a string to use with the context:

```
ctx.strokeStyle = properties.get(
 '--foreground-color').toString();
ctx.shadowOffsetX = properties.get(
 '--dropshadow-distance').toString();
ctx.shadowOffsetY = properties.get(
 '--dropshadow-distance').toString();
ctx.shadowColor = properties.get('
 --dropshadow-color').toString();
ctx.lineWidth = width;
ctx.lineCap = 'round';
```

Mask transparency

When creating a mask image, anything not transparent will let the layer below through. Create a vanishing effect by using a gradient that fades to transparent.

9. Set up shape loop

To spread the squiggle shapes in both dimensions, we can use a loop to iterate over the width and height. The second parameter for the 'paint' method provides the dimensions of the paint area, which we can use as limits for the loop.

We also need to apply an offset to keep the shapes centred. This is done by calculating the space left over once all the shapes have been placed.

Clear the 'rect' and 'stroke' calls from Step 4 and replace them with the following:

```
let offsetX = (geom.width / 2)
 % (width * spacing);
let offsetY = (geom.height / 2)
 % (width * spacing);
for(let y = 0; y < geom.height/
 (width * spacing) + 1; y++) {
  let count = 0;
  for(let x = 0; x < geom.width/
   (width * spacing) + 1; x++) {
  }
}
```

10. Calculate draw position

These loops will add as many shapes as they can for each row, then move down to the next row. By drawing the shapes inside the innermost loop, they will be spread across the painted element.

First, we need to calculate each position based on the iteration of 'x' and 'y' we are on.

We can do that by multiplying the line thickness and the spacing between them by how many times we have looped so far. From that, take off any offset calculated to keep things lined up.

```
let pointX = (x * width * spacing)
 - ((width * spacing) - offsetX);
let pointY = (y * width * spacing)
 - ((width * spacing) - offsetY);
```

11. Call to draw the shapes

With the values in place, we can call the dedicated 'drawPoint' method we will use to draw the shapes. This

Top
Without moving the context to 0,0 the first line will instead act as the starting point. The result is an arrow shape rather than a lightning bolt

Right
A text zoom bug in Google Chrome means that the context will not render any smaller than its original size at 100 per cent when zoomed out

Is Houdini ready yet?

	Apple Safari	Google Chrome	Microsoft Edge	Mozilla Firefox	Opera
Layout API (Spec Explainer)	no signal	intent Details	no signal	no signal	no signal
Paint API (Spec Demos Article)	no signal	yes (Chrome 65) Details	no signal	intent (Servo) Details	no signal
Parser API (Explainer)	no signal	no signal	no signal	no signal	no signal
Properties & Values API (Spec)	no signal	partially (Canary) Details	no signal	development Details	no signal
AnimationWorklet (Spec Explainer Demos)	no signal	intent Details	no signal	no signal	no signal
Typed OM (Spec)	intent Details	partially (Canary) Details	no signal	intent Details	no signal
Font Metrics API (Spec)	no signal	no signal	no signal	no signal	no signal

What is Houdini capable of?

The Paint API forms part of Houdini – a collection of APIs and specifications designed to decrease the friction between JavaScript and CSS. By exposing some of the magic that browsers perform to developers, we are able to make adjustments to the page that work best for our specific use-case.

Fixed in the middle of these advancements is the concept of a worklet – a lightweight, highly specialised class focused on specific actions such as animations. They are similar to Web Workers, which execute away from the main thread and keep the page responsive. These can be expanded upon in future as the needs of developers evolve.

The draft APIs are constantly changing and those present at the moment may not resemble what eventually lands in browsers. Keep an eye on **ishoudinireadyyet.com** to stay ahead of the latest improvements.

will take all the information we have at the time in order to draw it correctly.

As well as the calculated points, we should supply the position values of the mouse, the defined width of each squiggle line, and a reference to the context to draw upon.

```
this.drawPoint(pointX, pointY,
 properties.get('--x-position'),
 properties.get('--y-position'),
 width, ctx);
```

12. Draw from each point

The position values supplied to drawPoint indicate the location where the squiggle lines will start from and rotate around.

For each point we translate the context to the defined position and move our reference point to '0,0'. This lets us draw each shape without worrying about offsets against the start position. From there we describe a line, draw it and then restore the original context positioning for the next point.

```
ctx.save();
ctx.beginPath();
ctx.translate(x, y);
ctx.moveTo(0,0);
ctx.lineTo(10, 8);
ctx.lineTo(20, 0);
ctx.lineTo(30, 8);
ctx.stroke();
ctx.restore();
```

13. Orientate towards cursor

By moving the context to '0,0' the starting point of each line also becomes its rotation point. As we save and restore the context after drawing it means we can rotate each shape independently of the others.

Just after translation, rotate the context to the appropriate angle calculated in a separate method. If the cursor has not moved yet, it will not have any position set. Avoid rotating at all if that is the case.

```
if (cursorX && cursorY) {
  ctx.rotate(
  this.calculateAngle(
  x, y,
  cursorX.toString(),
  cursorY.toString()
  )
  );
}
```

14. Calculate the rotation angle

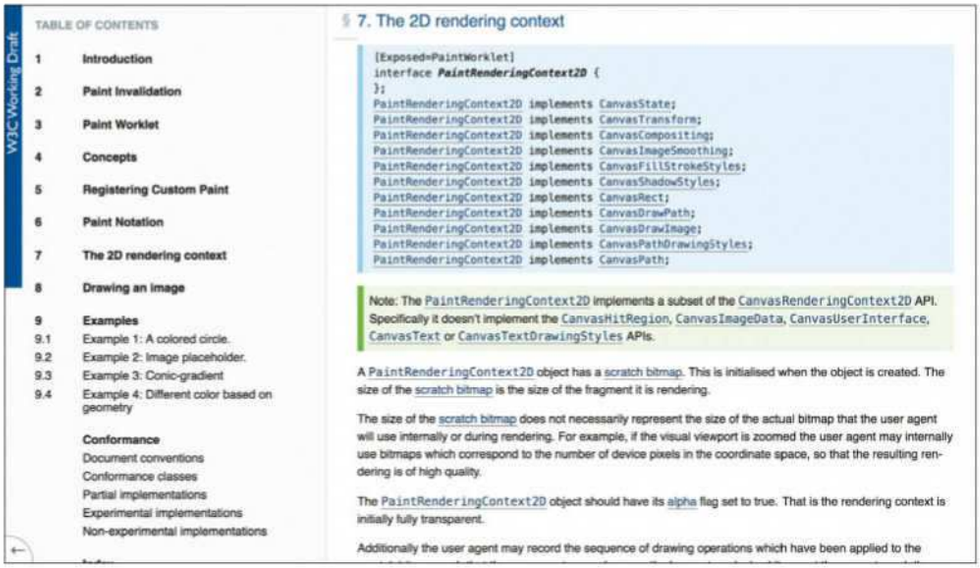
The 'calculateAngle' method does not yet return anything, which is why the shapes have not yet rotated.

We know the two points – the shape and cursor positions – and by measuring the arctangent between these two points, we get the angle in radians. JavaScript includes the necessary function to calculate this for us. Update 'calculateAngle' to return the correct value.

```
return Math.atan2(cursorY - pointY, cursorX
- pointX);
```

15. Mix up the rows

This works great, but having each shape lined up is not the best effect. We can update the 'for' loop from earlier to offset every other shape to sit in between the other rows for a polka dot effect.



Limitations of Paint Worklets

Paint worklets work on a context system very similar to the rendering context found when working with <canvas> elements. Both let you draw paths, fill shapes and apply gradients that work in the same way. But for safety and performance reasons, some things are not yet possible inside these worklets.

Text drawing methods are not present for security reasons. Different browsers render the same string within a context with slight differences. Malicious could detect these anomalies and calculate the browser and machine being used – a technique known as 'fingerprinting'.

It is also not possible to use any ImageData methods to read and write pixels from the context. Since worklets have been designed to be fast and lightweight, reading the data in this manner on each frame can create a performance bottleneck.

Back in the paint method, within the inner 'for' loop, use the count variable to offset every other shape. By adding another half of the spacing, this means it lands halfway between each row.

```
count++;
if (count % 2 === 1) {
  pointY += (width * (spacing / 2)); }
```

16. Another use of the API

So far, we have been able to create an interactive background without creating any new elements. But the CSS Paint API is capable of much more. For example, we can combine it with image masks to reshape elements on demand.

To show how this might work, navigate to '/mask' for a different example.

17. Apply masking CSS

Before we start creating the worklet, we need to set the mask on each image.

While the CSS property needed is 'mask-image', it is still prefixed in Chrome. This means we also need the Webkit prefix alongside it.

Open up 'styles/mask.css' and update the styles for the 'masked' class.

Part of the effect is to fully reveal the image when the user hovers over it. Add a 'reveal' custom property to the element in that instance.

```
.masked {
  […]
  -webkit-mask-image: paint(mask);
  mask-image: paint(mask);
}
.masked:hover, .masked:focus {
  --mask-reveal: true;
}
```

18. Apply mask shape property

Custom properties do not need to be on the root element to be picked up by the API. They can be applied to the element being painted to, which will override any selectors with a lower specificity.

Now we're going to apply a special 'mask-shape' custom property to three shape classes. This tells the worklet which shape to render.

```
.masked.square {
  --mask-shape: square;
}
.masked.circle {
  --mask-shape: circle;
}
.masked.triangle {
  --mask-shape: triangle;
}
```

19. Define which shape to use

Open 'scripts/mask.js'. This has already been filled out with some of the logic to create the shapes. We still need to tie it together with the classes we just wrote.

The 'mask-shape' custom property defines the shape that's going to be used. We have a circle, a square or a triangle to choose from. If someone chooses a different shape or no shape at all, default to a circle shape. Trim off any spaces left by CSS formatting.

```
let shape = 'circle';
if (properties.get('--mask-shape') &&
  ['square', 'triangle', 'circle']
  .includes(properties.get(
    '--mask-shape').toString().trim())){
  shape = properties.get(
    '--mask-shape').toString().trim(); }
```

20. Decide whether to reveal

By default, the masks provide the outline of the defined shape. When the user hovers over them, the 'reveal' custom property becomes true, so that should fill the shape in.

Custom property types are coming at a later point, which means for now a 'true' value will be passed as a string. Match against the string and set a new variable if it is set.

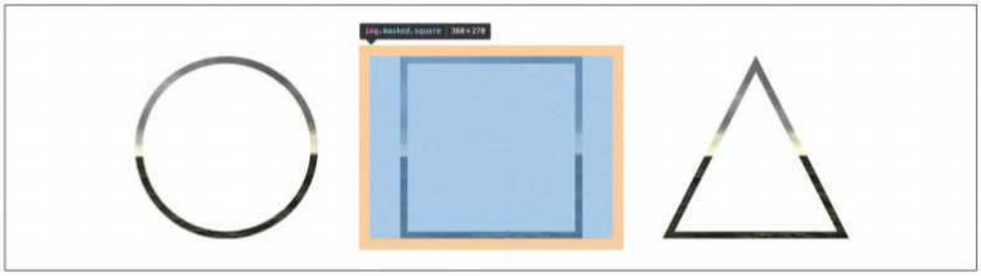
```
let reveal = false;
if (properties.get('--mask-reveal') &&
  properties.get('--mask-reveal')
  .toString().trim() == 'true') {
  reveal = true; }
```

21. Find the smallest length

Finally, we want the shapes to have sides of equal length. This means we need to find the smallest of either the width or height of the image and use that length to calculate the shapes.

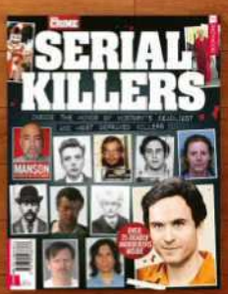
Define the 'maxLength' variable to tell the rest of the code what size to make the shapes. Set the line width of the context to a proportion of this size in order to keep things to scale.

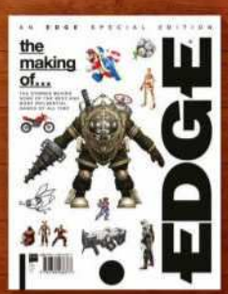
```
const maxLength =
  Math.min(geom.width, geom.height);
ctx.lineWidth = maxLength / 25;
```



Discover another of our great bookazines

From science and history to technology and crafts, there are dozens of Future bookazines to suit all tastes





Get great savings when you buy direct from us

1000s of great titles, many not available anywhere else

World-wide delivery and super-safe ordering

FUTURE

www.myfavouritemagazines.co.uk

Magazines, back issues & bookazines.

HOSTING LISTINGS

Featured host: **Netcetera**

netcetera.co.uk
03330 439780

About us

Formed in 1996, Netcetera is one of Europe's leading web hosting service providers, with customers in over 75 countries worldwide.

As the premier provider of data centre colocation, cloud hosting, dedicated servers and managed web hosting services in the UK, Netcetera offers an array of services designed to more effectively manage IT infrastructures. A state-of-the-art data centre environment enables Netcetera to offer your business enterprise-level colocation and hosted solutions. Providing an unmatched value for your budget is the driving force behind our customer and managed infrastructure services. From single server to fully customised data centre suites, we focus on the IT solutions you need.

What we offer

- **Managed hosting –** A full range of solutions for a cost-effective, reliable, secure host.
- **Cloud hosting –** Linux, Windows, Hybrid and Private Cloud Solutions with support and scalability features.
- **Data centre colocation –** Single server through to full racks with FREE setup and a generous bandwidth.
- **Dedicated servers –** From QuadCore up to Smart Servers with quick setup and fully customisable.

5 tips from the pros

1. Reliability, trust & support

Reliability is a major factor when it comes to choosing a hosting partner. Netcetera guarantees 100 per cent uptime, multiple internet routes with the ability to handle DDOS attacks, ensuring your site doesn't go down when you need it.

2. Secure and dependable

Netcetera prides itself on offering its clients a secure environment. It is accredited with ISO 27001 for security along with the options of configurable secure rackspace available in various configurations.

3. 24/7 technical support

Netcetera has a committed team of knowledgeable staff available 24/7 to provide you with assistance when you need it most. Our people make sure you are happy and your problems are resolved as quickly as possible.

4. Value for money

We do not claim to be the cheapest service available, but we do claim to offer excellent value for money. We also provide a price match on a like-for-like basis, as well as a price guarantee for your length of service.

5. Eco-friendly

Netcetera's environmental commitment is backed by use of eco-cooling and hydroelectric power. This makes Netcetera one of the greenest data centres in Europe.

Testimonials

Roy T

"I have always had great service from Netcetera. Their technical support is second to none. My issues have always been resolved very quickly."

Suzy B

"We have several servers from Netcetera and their network connectivity is top-notch, with great uptime and speed is never an issue. Tech support is knowledgeable and quick in replying. We would highly recommend Netcetera."

Steve B

"We put several racks into Netcetera, basically a complete corporate backend. They could not have been more professional, helpful, responsive or friendly. All the team were an absolute pleasure to deal with, and nothing was too much trouble, so they matched our requirements 100 per cent."

Supreme hosting

CWCS MANAGED HOSTING — SUPREME HOSTING. SUPREME SUPPORT.

cwcs.co.uk
0800 1777 000

CWCS Managed Hosting is the UK's leading hosting specialist. They offer a fully comprehensive range of hosting products, services and support. Their highly trained staff are not only hosting experts, they're also committed to delivering a great customer experience and are passionate about what they do.

- Colocation hosting
- VPS
- 100 per cent network uptime

UK-based hosting

CYBERHOSTPRO

cyberhostpro.com
0845 5279 345

Cyber Host Pro are committed to providing the best cloud server hosting in the UK; they are obsessed with automation. If you're looking for a hosting provider who will provide you with the quality you need to help your business grow, then look no further than Cyber Host Pro.

- Cloud VPS servers
- Reseller hosting
- Dedicated servers

Cluster web hosting

fasthosts

fasthosts.co.uk
0808 1686 777

UK-based and operating 24/7 from dedicated UK data centres. Fasthosts keep over one million domains running smoothly and safely each day. Services can be self-managed through the Fasthosts Control Panel.

- Dedicated servers
- Cloud servers
- Hosted email

Budget hosting

HETZNER ONLINE

hetzner.com
+49 (0)9831 505-0

Hetzner Online is a professional web hosting provider and experienced data centre operator. Since 1997, the company has provided private and business clients with high-performance hosting products as well as the infrastructure for the efficient operation of sites. A combination of stable technology, attractive pricing, flexible support and services has enabled Hetzner Online to strengthen its market position nationally and internationally.

- Dedicated/shared hosting
- Colocation racks
- SSL certificates

All-inclusive hosting

1&1

1and1.co.uk
0333 336 5509

1&1 Internet is a leading hosting provider that enables businesses, developers and IT pros to succeed online. Established in 1988, 1&1 now operates across ten countries. With a comprehensive range of high-performance and affordable products, 1&1 offers everything from simple domain registration to award-winning website building tools, eCommerce packages and powerful cloud servers.

- Easy domain registration
- Professional eShops
- High-performance servers

SSD web hosting

bargainhost

bargainhost.co.uk
0843 289 2681

Since 2001, Bargain Host have campaigned to offer the lowest possible priced hosting in the UK. They have achieved this goal successfully and built up a large client database, which includes many repeat customers. They have also won several awards for providing an outstanding hosting service.

- Shared hosting
- Cloud servers
- Domain names

Value Linux hosting

PATCHMAN WEB HOSTING

patchman-hosting.co.uk
01642 424 237

Linux hosting is a great solution for home users, business users and web designers looking for cost-effective and powerful hosting. Whether you are building a single-page portfolio, or you are running a database-driven eCommerce website, there is a Linux hosting solution for you.

- Student hosting deals
- Site designer
- Domain names

Flexible cloud servers

elastichosts

elastichosts.co.uk
020 7183 8250

ElasticHosts offer simple, flexible and cost-effective cloud services with high performance, availability and scalability for businesses worldwide. Their team of engineers provide excellent support 24/7 over the phone, by email and with a ticketing system.

- Cloud servers with any OS
- Linux OS containers
- 24/7 expert support

COURSE LISTINGS

Featured: Northcoders

northcoders.com
Twitter: @northcoders
Facebook: Northcoders

About us

Northcoders is the coding bootcamp for the north, based in the heart of Manchester and built upon northern values of grit, determination and community spirit. No matter what your background, you can fast-track your career and become a web or software developer in 12 weeks at their full-time bootcamp, or fit their course around your life with their 24-week part-time bootcamp. Their internal career support team will help find you work as a developer, setting up interviews with your choices of Northcoders Hiring Partners across the north of England.

What we offer

- **Full-time:** Fast-track your career in just 12 weeks.
- **Part-time:** Fit our curriculum around your life in 24 weeks.

5 tips from the pros

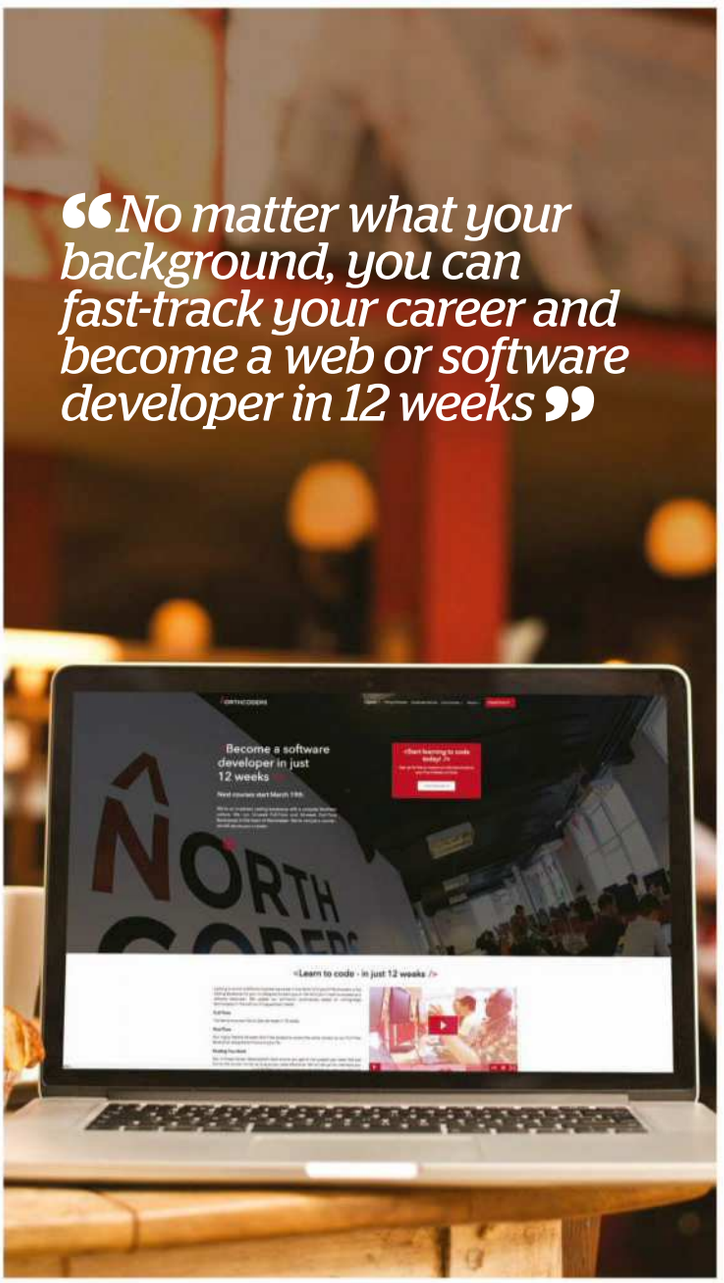
1. Get started with coding
The best way to know if coding is for you is to just try it! We recommend the free, online JavaScript track of Codecademy to get you started with the basics.

2. Do your research
Make sure you read plenty of student reviews to make sure you're applying somewhere reputable. Read their blog and have a look at their social channels.

3. Throw yourself in
Once you've decided it's right for you, set aside a few evenings each week to really start making progress! If coding is for you, this should be fun.

4. Be prepared
We'll be with you every step of the way when you apply. Make sure you go through all the materials we recommend and ask for help if you're stuck.

5. Get social
With Northcoders, you're not just on a course, you're part of a community that will stay with you long after you graduate. Make the most of it!



> Becoming part of this vibrant, caring community was something I hadn't expected before the course, but now I couldn't be without it. To be a Northcoder is to be enlightened, inspired and supported.
>
> **Joanne Imlay**
> ***Primary school teacher to software developer at Careicon***

> Northcoders delivered their part of the bargain in spades. They provided tremendous assistance in turning me into the full product – a well-rounded, capable, future tech employee – and they have the contacts to deliver the opportunities for such people.
>
> **Joe Mulvey**
> ***Maths teacher to software developer at Auto Trader***

udemy

UDEMY
udemy.com

Twitter: @udemy
Facebook: udemy

The inspiration for Udemy began in a small village in Turkey, where founder Eren Bali grew up frustrated by the limitations of being taught in a one-room school house. Realising the potential of learning on the internet, he set out to make quality education more accessible. Udemy is now a global marketplace for learning and teaching online. Students can master new skills by choosing from an extensive library of over 40,000 courses including HTML, CSS, UX, JavaScript and web development.

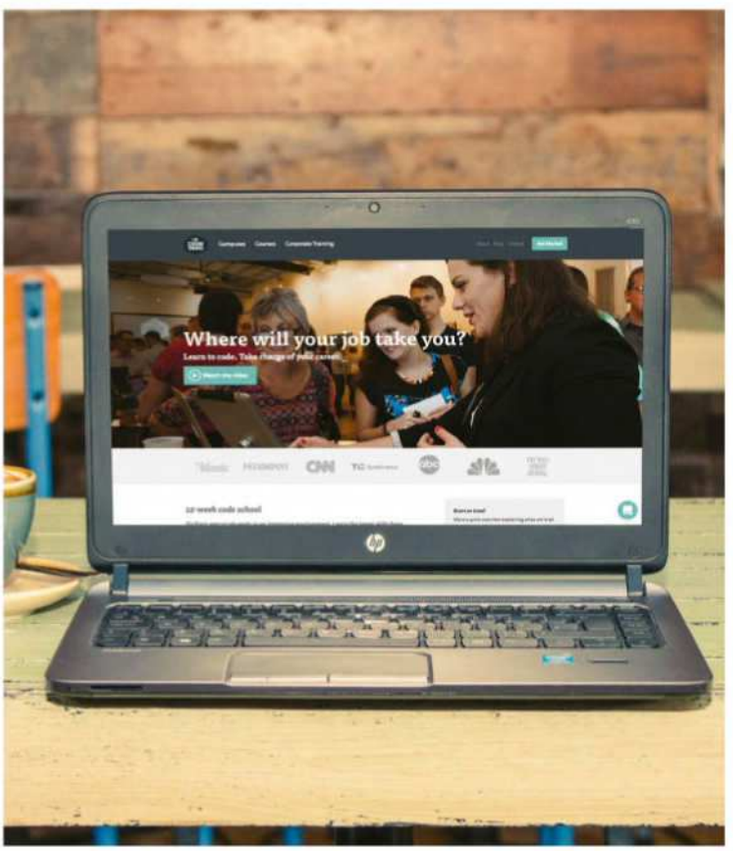
40,000+ courses: There is a course for every designer and dev.
Self-paced learning: Learn how to code at your own pace.

THE IRON YARD
theironyard.com

Twitter: @TheIronYard
Facebook: TheIronYard

The Iron Yard is one of the world's largest and fastest-growing in-person code schools. It offers full-time and part-time programs in backend engineering, frontend engineering, mobile engineering and design. The Iron Yard exists to create real, lasting change for people, their companies and communities through technology education. The in-person, immersive format of The Iron Yard's 12-week courses helps people learn to code and be prepared with the skills needed to start a career as junior-level software developers.

12-week code school: Learn the latest skills from industry pros.
Free crash courses: One-night courses, the perfect way to learn.



WE GOT CODERS
WE GOT CODERS

wegotcoders.com
hello@wegotcoders.com

We Got Coders is a consultancy that provides experts in agile web development, working with startups, agencies and government. Take one of their 12-week training courses that covers all that is required to become a web developer, with highly marketable full-stack web development skills.

- Classroom-based training
- Real-world work experience
- Employment opportunities

FUTURELEARN
Future Learn

futurelearn.com
feedback@futurelearn.com

Choose from hundreds of free online courses, from Language & Culture to Business & Management; Science & Technology to Health & Psychology. Learn from the experts. Meet educators from top universities who'll share their experience through videos, articles, quizzes and discussions.

- Learn from experts
- Free courses
- All-device access

GYMNASIUM
GYMNASIUM

thegymnasium.com
help@thegymnasium.com

Gymnasium offers free online courses, designed to teach creative professionals in-demand skills. Courses are all self-paced and taught by experienced practitioners with a passion for sharing practical lessons from the design trenches.

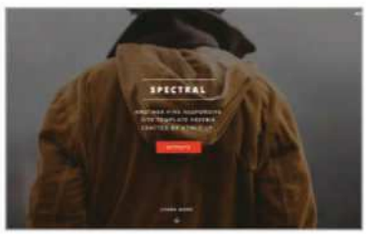
- Gain real-world skills
- Get expert instruction
- Career opportunities

Free with your magazine

Instant access to these creative resources…

Essential assets and resources

Get textures, fonts, backgrounds and more



Exclusive video tutorials

Learn to code/create with HTML, CSS, JS & PHP

Tutorial project files

All the assets you'll need to follow our tutorials

Plus, all of this is yours too…

- All-new tutorial files to help you master this issue's HTML, CSS and JavaScript techniques
- Over 79 minutes of Beginners JavaScript video from StudioWeb (**www.studioweb.com**)
- 14 pink Photoshop filters and 10 wooden picture frame mockups from Sparklestock (**www.sparklestock.com**)

Log in to www.filesilo.co.uk/webdesigner

Register to get **instant access** to this pack of must-have creative resources, how-to videos and tutorial assets

Free for digital readers, too!
Read on your tablet, download on your computer

FileSilo

The home of great downloads – exclusive to your favourite magazines from Future!

- Secure and safe online access, from anywhere
- Free access for every reader, print and digital
- Download only the files you want, when you want
- All your gifts, from all your issues, in one place

Get started

Everything you need to know about accessing your FileSilo account

01 Follow the instructions on screen to create an account with our secure FileSilo system. Log in and unlock the issue by answering a simple question about the magazine.

02 You can access FileSilo on any computer, tablet or smartphone device using any popular browser. However, we recommend that you use a computer to download content, as you may not be able to download files to other devices.

03 If you have any problems with accessing content on FileSilo, take a look at the FAQs online or email our team at the address below.

filesilohelp@futurenet.com

An incredible gift for subscribers

Subscribe today & unlock the free gifts from more than 50 issues

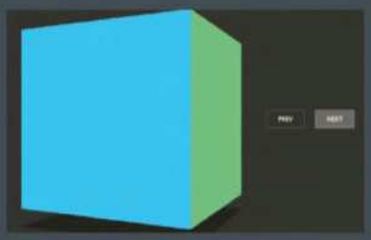
Access our entire library of resources with a money-saving subscription to the magazine – that's more than 900 free resources

Over 60 hours of video guides

Let the experts teach you to create and code

More than 400 tutorials

Get the code you need to get creative



Over 250 creative assets

Templates, fonts, textures and backgrounds

Head to page 32 to subscribe now

Already a print subscriber?
Here's how to unlock FileSilo today…

Unlock the entire Web Designer FileSilo library with your unique Web ID – the ten-digit alphanumeric code printed above your address details on the mailing label of your subscription copies – also found on any renewal letters.

More than 900 reasons to subscribe

+ More added every issue

NEXT MONTH

DESIGN FASTER

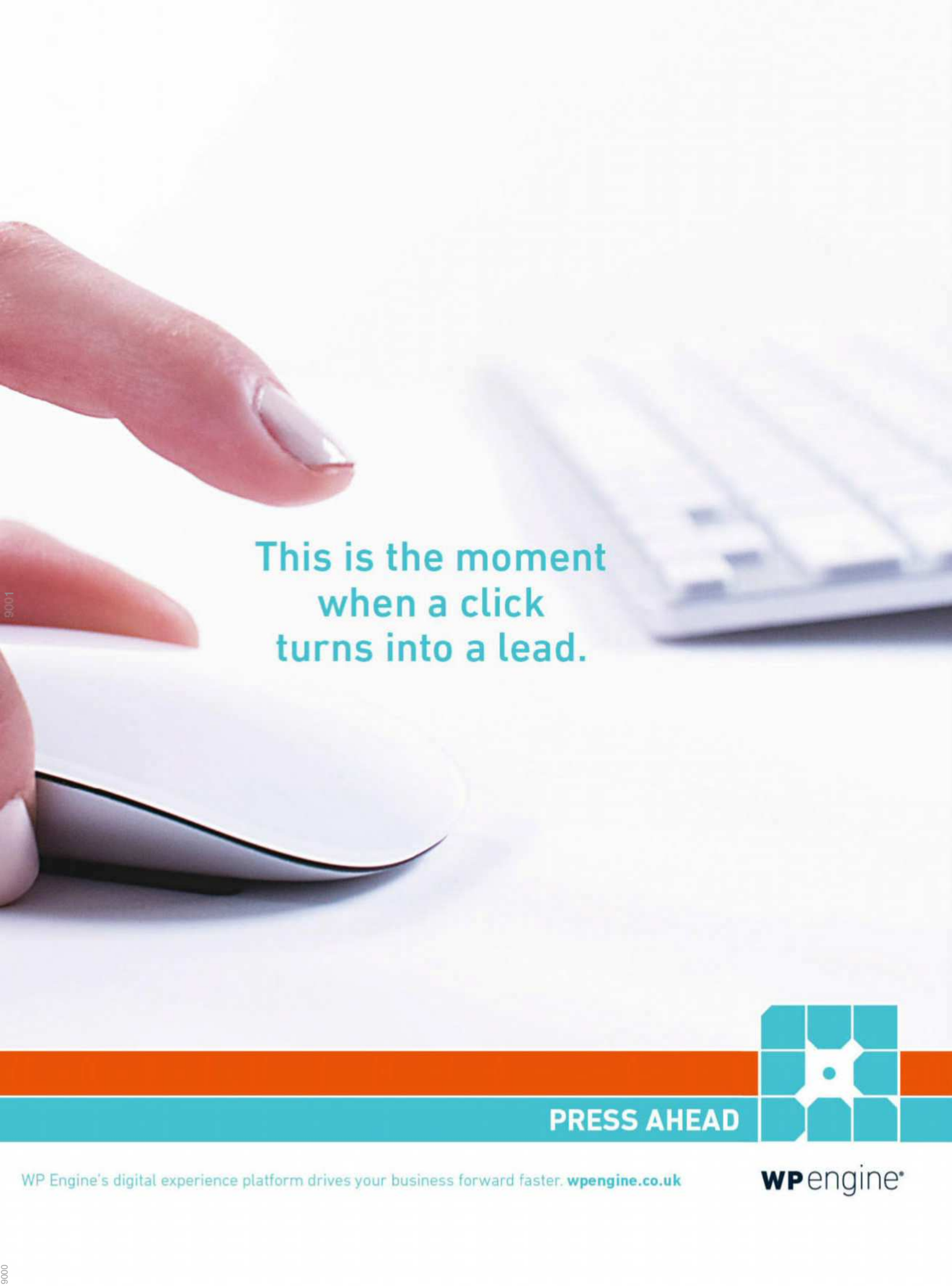
CODE QUICKER

50 essential tools, libraries, frameworks and extensions for the modern day designer and developer

REDUX AND ANGULAR
Find out how to combine application state management with your Angular projects

CREATE DIRECTION-AWARE ANIMATIONS
Determine the ways in which hidden content is revealed on hover

SAY HELLO TO GOOGLE'S FLUTTER?
Take a look at the mobile UI framework for crafting high-quality native interfaces on iOS and Android


This is the moment
when a click
turns into a lead.
PRESS AHEAD
WP Engine's digital experience platform drives your business forward faster. wpengine.co.uk
WP engine