DZone

THE DZONE GUIDE TO

WEB DEVELOPMENT

APPLICATIONS & FRAMEWORKS VOLUME I

BROUGHT TO YOU IN PARTNERSHIP WITH

Optimizely

Qlik

DEAR READER,

Welcome to the first edition of the DZone Web Developer Guide and our 12th DZone Research Guide of 2016. At the end of each year I often reflect on the past months. The rate of change around web development has been increasing. Hardly a day goes by where we don't see a new framework, or a new technique. Methods that seemed advanced when we overhauled DZone.com just 1.5 years ago, seem much more commonplace now. And who remembers "Web 2.0?" That seems like ancient history today, when in truth it was just a few short years ago.

The world of Web Development is growing. JavaScript, and other front-end technologies, are by far the most popular languages on Stack Overflow. The DZone Web Dev portal has been steadily growing, and is now our second most popular portal. Modern-day web development spans both the front-and the back-end. Today, the full-stack developer often reaches into their toolbox and pulls out JavaScript, as tools like Node.js allow the developer to leverage a single language across the entire stack—allowing for focus and productivity. The number of developers familiar with JavaScript ensures that your projects will have an easier time finding qualified people as well.

With a low barrier to entry, many new developers (and experienced ones as well) are building web applications. Looking back to our recent Application Security Guide, XSS (Cross-Site Scripting) and other front-end attacks are the most prevalent. How will we enforce security in this fast-moving and fast-growing world where new frameworks appear nearly daily? As we begin to rely more and more on third-party APIs, how do we ensure that those services are following the best practices to secure our data? Are we up to the task of training this next generation of developers on the security requirements of the modern web?

This guide attempts to answer some of these questions and more, starting by walking you through some of the research findings that we discovered while talking to our audience and then diving into some great articles by industry experts. It is to-the-point and one of the most technically deep guides that we have published.

The articles are filled with code snippets and examples on topics from arranging your projects more effectively and tips for debugging your JavaScript, to being able to host sites or even handle user management and authentication all without running a server. Some in the industry will say that this is the way of the future—everything delegated to third-party services. Time will tell—and given the rate of change lately, I imagine that time will be soon!

Thank you for reading the first edition of *The DZone Guide to Web Development*, and thank you to all our sponsors and contributors who helped make this year's series be the best one yet. Happy Holidays to everyone and I hope you enjoy reading this guide as much as we enjoyed making it!

BY MATT SCHMIDT
PRESIDENT & CTO AT DZONE
RESEARCH@DZONE.COM

TABLE OF CONTENTS

3 **EXECUTIVE SUMMARY**
BY MATT WERNER

4 **KEY RESEARCH FINDINGS**
BY G. RYAN SPAIN

8 **FUNCTIONAL REACTIVE UI PROGRAMMING**
BY MICHAEL HEINRICHS

10 **TOTALLY MODULAR, DUDE!**
BY BILL SOUROUR

13 **CHECKLIST: DEBUGGING JS WITH THESE 10 TIPS**
BY FREYJA SPAVEN & DANIEL WYLIE

14 **INFOGRAPHIC: SURVIVAL INSTINCTS: MOST TRUSTED TECH FOR THE WEB DEV JUNGLE**

16 **ADDING AUTHENTICATION TO A WEB APPLICATION WITH AUTH0, REACT, AND JWT**
BY JOSÉ AGUINAGA

19 **DIVING DEEPER INTO WEB DEV**

22 **EXECUTIVE INSIGHTS ON WEB APPLICATION DEVELOPMENT**
BY TOM SMITH

24 **EMBRACING SIMPLICITY WITH STATIC SITE GENERATORS**
BY RAYMOND CAMDEN

27 **WEB DEV SOLUTIONS DIRECTORY**

31 **GLOSSARY**

MARKETING
KELLET ATKINSON
DIRECTOR OF MARKETING

LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

NATALIE IANNELLO
MARKETING SPECIALIST

PRODUCTION
CHRIS SMITH
DIRECTOR OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORDINATOR

G. RYAN SPAIN
PRODUCTION PUBLICATIONS EDITOR

ASHLEY SLATE
DESIGN DIRECTOR

EDITORIAL
CAITLIN CANDELMO
DIRECTOR OF CONTENT + COMMUNITY

MATT WERNER
CONTENT + COMMUNITY MANAGER

MICHAEL THARRINGTON
CONTENT + COMMUNITY MANAGER

NICOLE WOLFE
CONTENT COORDINATOR

MIKE GATES
CONTENT COORDINATOR

SARAH DAVIS
CONTENT COORDINATOR

TOM SMITH
RESEARCH ANALYST

BUSINESS
RICK ROSS
CEO

MATT SCHMIDT
PRESIDENT & CTO

JESSE DAVIS
EVP & COO

MATT O'BRIAN
DIRECTOR OF BUSINESS DEVELOPMENT
sales@dzone.com

ALEX CRAFTS
DIRECTOR OF MAJOR ACCOUNTS

JIM HOWARD
SR ACCOUNT EXECUTIVE

JIM DYER
ACCOUNT EXECUTIVE

ANDREW BARKER
ACCOUNT EXECUTIVE

CHRIS BRUMFIELD
ACCOUNT MANAGER

ANA JONES
ACCOUNT MANAGER

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

SPECIAL THANKS
to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this report a great success.

Executive Summary

BY MATT WERNER
CONTENT AND COMMUNITY MANAGER, DZONE

While the launch of smartphones has truly changed the world, most of us interact with the online world through web apps rather than mobile apps. JavaScript in particular has been one of the most important technologies in modern web development, allowing web apps to have similar capabilities as native mobile apps. In our first annual web dev survey, 97% of respondents said they wrote JavaScript for the browser, compared to 39% for mobile-only or 26% for hybrid mobile apps. The prevalence of responsive design has also kept web apps relevant, as it's now standard practice to design responsive sites to adapt to any screen or device. To discover what developers were using for their web development needs, DZone surveyed 598 developers and tech professionals across the world, most of whom had extensive experience working at companies with over 100 people.

IS ANGULAR 1 DEAD?

DATA Angular was the most popular framework among respondents (69%). 19% of total respondents who have used Angular 1 would not use it again. Less than 1% of users have used either Angular 2 or React and would not use either again.

IMPLICATIONS Angular 1 took the development world by storm, but its issues have been well-documented, and there is a statistically significant difference between the number of users unhappy with Angular 1 over the next two most popular client-side frameworks: React and Angular 2.

RECOMMENDATIONS Angular 1 is clearly not without its faults. The Angular team has attempted to address them in Angular 2, which was released in mid-2016. If developers are finding that they have a lot of issues with their Angular 1 applications, Google has extensive documentation on how to use Angular 2's migration tools. Based on audience responses, the satisfaction with Angular 2 is high, and it may be worth the time to migrate Angular 1 apps.

DOMINANT TECHNOLOGIES EMERGE

DATA Across several questions, clear technological favorites emerged. For instance, Chrome was the most popular browser to develop for vs. second-place Firefox (96% to 77%). For package management, npm is used more than bower (59% to 37%), for testing, Selenium is used more than Jasmine (64% to 31%), and for real-time streaming apps, WebSocket API (77%) was used far more than the next most popular method: polling (32%).

IMPLICATIONS While there may be debates over the quality of one tool or framework versus another, tools like jQuery, used by 84% of respondents, and AngularJS can quickly become the dominant technology in a particular category. This can be due to the strength of the community, documentation, or familiarity with older tools, such as jQuery or MySQL.

RECOMMENDATIONS Keeping up with what's considered the most popular language or technology is essential for web developers in the market, as these are skills that will be easily transferable between projects or jobs. However, as demonstrated by the statistically significant number of respondents who were unhappy with Angular 1, the most popular JavaScript framework, the most popular tool may not be the best, so it would behoove developers to explore other options either at work or in their spare time.

MORE FOCUS ON THE BACKEND

DATA 97% of developers said they work on the browser, 39% said they do server side. 56% said they worked on the full stack, and only 8% work mostly on the front end.

IMPLICATIONS More web development is being done on the backend except in the cases of mobile web development, where more is done on the frontend. Since more and more developers have determined that capabilities like business logic and enterprise integration should be relegated to the backend, more web developers need to have a knowledge of full-stack development.

RECOMMENDATIONS It's clear from the data that most web developers need a knowledge of full-stack development. Continued education into new technologies like Node.js or Express.js will be essential to staying competitive in the job market and adding functionality to web apps. 53% of respondents already deliberately switch frontend and backend development roles to build full-stack skills. In addition, DZone research has found that full-stack developers are more likely to expand their knowledge of web development. For example, full-stack developers are more focused on responsive design than backend developers (95% vs. 89%), and are more likely to use newer JS features such as modules and classes than developers who only work on the front or backend.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

598 software professionals completed DZone's 2016 Web Development Reesearch Guide survey. Of these respondents:

- 38% have a primary role as developers/ engineers, and 33% are developer team leads.
- 60% have 10 years of experience or more as software professionals, and 37% have 15 years or more.
- 40% work for companies headquartered in Europe, 29% for companies headquartered in the US, and 12% for companies headquartered in South America.
- 17% work for companies with 10,000 employees or more, 19% for companies with between 1,000 and 10,000 employees, and 24% for companies with between 100 and 1,000 employees.

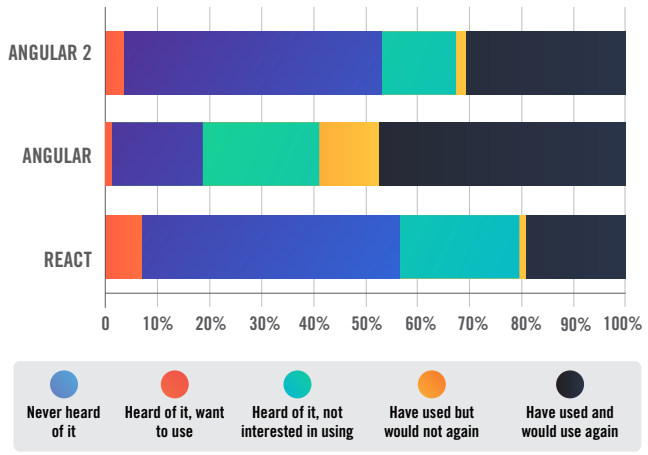
FRAMEWORK POPULARITY

The AngularJS framework has had an enormous impact on web development since its release in 2010. Among our survey respondents, Angular was the most widely used framework, with 69% use, well ahead of second- and third-place frameworks Node.js (50%) and React (30%). However, attitudes towards Angular, at least Angular 1, suggest that developers may be searching for alternatives to the original Angular framework. 11% of respondents said they have used Angular 1 and would not use it again, almost 1/5 of the respondents who said they have used Angular at all. Compared to attitudes about React and Angular 2, the two next most popular client-side JavaScript frameworks, each of which having <1% of users saying they would not use again, this number is not insignificant. And while React and Angular 2 are not yet as widely used, with 30% of respondents saying they have used each in some capacity, interest in both of these frameworks is high: 50% of respondents said they have "heard of and want to use" each framework. While the broad interest in Angular 2 likely means Angular is not going anywhere soon, these results indicate a move away from Angular 1, and possibly toward other, non-Angular frameworks.

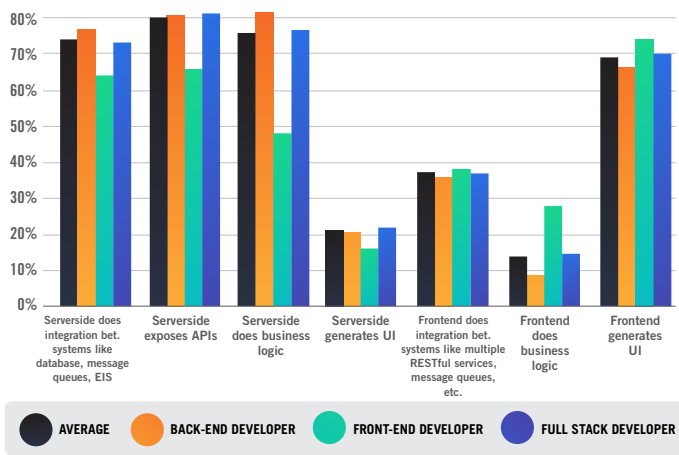
FRONTEND VS. BACKEND

Almost all survey respondents (97%) said they do some development for the browser, compared to 39% who said they do serverside development. When asked if they worked more on the frontend, backend, or the full stack of their web applications, 56% of respondents said they work on the full stack, while only 8% said they work more on the frontend. Regarding the division of work between client and server, a large majority of respondents said they have the serverside integrate between systems "like database, message queues, EIS" (74%), expose APIs (80%), and do the business logic of the application (76%), and the frontend generates the user interface (69%). These behaviors change based on whether respondents said they

HOW FAMILIAR ARE YOU WITH THESE CLIENT-SIDE JAVASCRIPT FRAMEWORKS?



IN YOUR WEB APPLICATIONS, HOW DO YOU TYPICALLY DIVIDE WORK BETWEEN CLIENT AND SERVER?



worked more on application frontends, as well as what kind of apps respondents said they are working on (for example, respondents working on mobile versions of a website were more likely than most to have the frontend rather than the backend perform business logic). And although respondents may have differing views of frontend vs. backend work, there seems to be some consensus that the way work is divided between them is important.

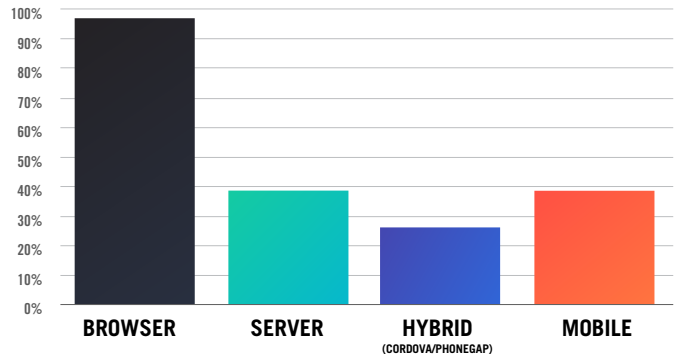
FULL-STACK PREFERENCES

The way respondents answered whether they worked more on the frontend, backend, or full-stack of web apps often have changed how they answered other questions. Frontend and backend respondents generally leaned more towards options that emphasized their frontend or backend focus, and the full-stack respondents generally fell somewhere in between. So it was interesting to learn which way full-stack respondents were leaning. As one example, full-stack respondents leaned much more closely to backend respondents regarding client/server work division. On the other hand, full-stack respondents leaned towards frontend respondents when it comes to responsive design; backend respondents were about twice as likely (11%) to say they did not use responsive design as frontend (4%) and full-stack (5%) respondents. But full-stack respondents were more likely than frontend and backend respondents to say they would not use Angular 1 again (14% versus 6% and 8%, respectively). And full-stack respondents were much more likely to frequently use popular JavaScript ES6 features like modules (34% vs. 21% and 20%), classes (30% vs. 21% and 24%), and arrow functions (36% vs. 17% and 28%) than frontend or backend respondents.

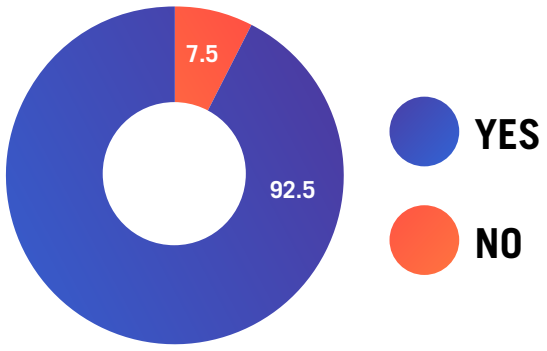
CHROME AND OTHER FAVORITES

Google Chrome was shown to be the most popular browser to develop for, based on our survey responses. 96% of respondents said they actively Chrome, a wide margin above runner-up Mozilla Firefox (77%) and third-place Microsoft Internet Explorer (51%). Chrome wasn't the only favorite that stood out in this year's survey results, though. Several different solutions stood well over alternatives in their field and were used by over half of survey respondents. For module/package management, npm (59%) was used much more than next-place Bower (37%). For testing, Selenium ranked first with 64% of respondents using the testing tool over others, such as Jasmine (31%) and PhantomJS (28%). The WebSocket API was favored for pushing data from server to browser among respondents whose apps need real-time or streaming communication, with 77% using this method; polling (32%) and HTTP streaming (26%) were the next most popular. 84% of all respondents said they use jQuery in developing their applications, and 61% said they use MySQL for their web app's database, over MongoDB and PostgreSQL (both 42%).

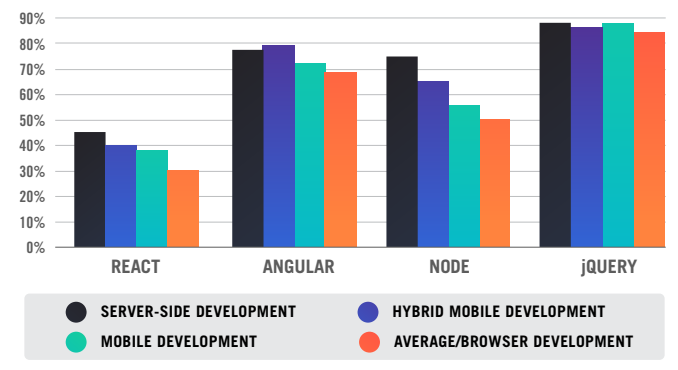
WHAT ENVIRONMENTS DO YOU WRITE JAVASCRIPT FOR?



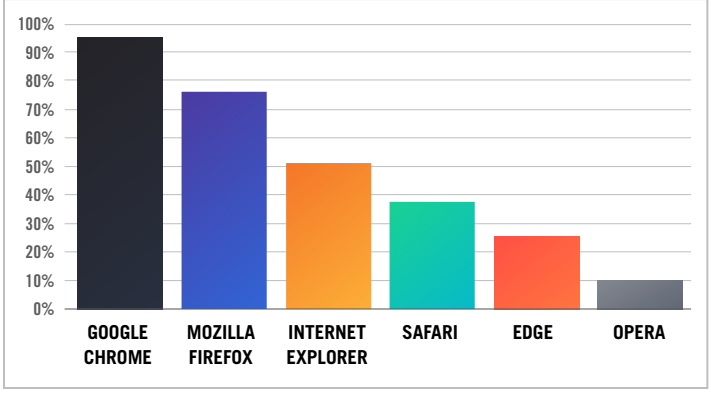
ARE YOUR WEB APPS RESPONSIVE?



WHICH FRAMEWORKS OR LIBRARIES DO YOU USE?



WHICH BROWSERS DO YOU ACTIVELY DEVELOP FOR?



Every great invention and scientific discovery was born from experimentation. Take Marie Curie, who was a relentless experimenter. Through experimentation, she became the first woman to be awarded a Nobel Prize, in 1903, for Physics. Further, she was the only person to win two Nobel Prizes in different sciences when she won the 1911 Nobel Prize for Chemistry. She said, "Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less."

Experimentation is not only a cornerstone of science, but it is also critical to success in politics, athletics, and business. Companies often have great ideas, but their inability to experiment easily leads to the design of safe and often uninspiring experiences for their customers. In a world that is changing faster than ever before—a world where a customer starts a task on one device and finishes it on another, a world where customers expect companies to know who they are—companies can't afford to play it safe or they'll be surpassed by their competition. As leaders in this constantly evolving world, our job is not to have all the answers. Our job is to enable our organization to be brave and experiment.

In a world that is changing faster than ever before—companies can't afford to play it safe or they'll be surpassed by their competition

That's why we launched Optimizely X, the world's first experimentation platform. Optimizely X empowers executives, marketing teams, product teams, and development teams to make better business decisions and deliver better customer experiences through experimentation. This represents a quantum leap forward in what our customers can achieve, by pushing the boundaries of experimentation and transforming digital experiences for their customers. With Optimizely X, you can experiment on any channel and any device with an internet connection. For more information, visit optimizely.com/products.

WRITTEN BY JOHN PROVINE
DIRECTOR, PRODUCT MANAGEMENT, **OPTIMIZELY**

PARTNER SPOTLIGHT

Optimizely X Full Stack

By Optimizely

Optimizely X Full Stack lets you quickly and reliably run experiments, learn, and launch new features anywhere in your technology stack.

CATEGORY
Experimentation Platform

NEW RELEASES
Monthly

OPEN SOURCE
All SDKs

CASE STUDY

Tripping is the world's largest search engine for vacation homes, showcasing about 8 million properties worldwide. A data-driven organization, Tripping runs iterative A/B tests to optimize customer experience, and Optimizely X Full Stack has become the backbone of their 'test before release' program. When Tripping releases a new product or feature, the company's development team uses Full Stack to test and benchmark changes against current metrics to make sure that there are no unintended consequences post-release. By having testing integrated on the server-side rather than the front-end, when a feature is already live, Tripping is able to ensure optimal performance prior to any launch across marketing, product, and engineering.

STRENGTHS

- **Fast decision engine** - Our SDKs bucket users in memory so experiments have no impact on latency. Feel free to conduct experiments in performance-critical code paths.
- **Scalable architecture** - Our real-time event collection servers used by thousands of customers across the globe allow you to get experiment results immediately.
- **Traffic splitting** - Split traffic to different code paths anywhere in your technology stack. No network requests needed means no impact on your app's performance.
- **Event tracking** - Monitor all the conversion metrics and KPIs you care about.
- **Open-source SDKs** - Install any of our SDKs and run your first experiment in 10 minutes.

NOTABLE CUSTOMERS

- CBS
- Clorox
- HP
- Microsoft
- ADP
- Spotify
- L'Oreal
- Adidas

BLOG blog.optimizely.com

TWITTER @optimizely

WEBSITE optimizely.com

Functional Reactive UI Programming

BY **MICHAEL HEINRICHS**
SOFTWARE ENGINEER AT **CANOO ENGINEERING AG**

QUICK VIEW

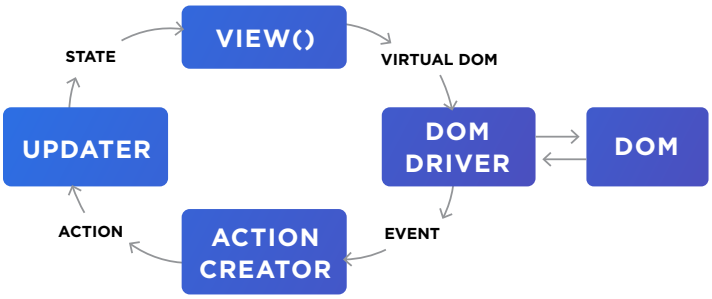
01 Using principles of functional reactive programming for UI development results in a straight-forward architecture: all events and changes move circular in a single direction.

02 Data structures are immutable and the individual components are purely functional.

03 This allows for simpler testing, an easy to understand and follow flow of events, and great tool support.

FRONTEND DEVELOPMENT BEYOND THE MVC PATTERN

React.js, Elm, Cycle.js, and other UI frameworks introduced a new way of building user interfaces. By applying principles from functional reactive programming to UI development, they even changed how we think about user interfaces. This article gives a brief introduction into this new way of building UIs and lists the main advantages it has over traditional approaches.



The picture above shows a rough overview of the concepts, which are shared between pretty much all modern UI frameworks that foster reactive programming. The first thing to note is that everything—all changes, events, and updates—flow in a single direction to form a cycle.

FUNCTIONAL REACTIVE UI DEVELOPMENT

The cycle consists of four data structures (State, Virtual DOM, Event, and Action) and four components (View()-Function, DOM-Driver, ActionCreator, and Updater).

The DOM-Driver is provided by the framework, while the other components have to be implemented by the application developer.

Let's assume our application, a todo list, has already been running for a while and the user presses a button to create a new entry in the todo list. This will result in a button-clicked event in the DOM, which is captured by the DOM-Driver and forwarded to one of our ActionCreators.

The ActionCreator takes the DOM-event and maps it to an action. Actions are an implementation of the Command Pattern, i.e. they describe what should be done, but do not modify anything themselves. In our example, we create an AddToDoItemAction and pass it to the Updater.

The Updater contains the application logic. It keeps a reference to the current state of the application. Every time it receives an action from one of the ActionCreators, it generates the new state. In our example, if the current state contains three todo items and we receive the AddToDoItemAction, the Updater will create a new state that contains the existing todo items plus a new one.

The state is passed to the View()-Function, which creates the so-called Virtual DOM. As the name suggests, the Virtual DOM is not the real DOM, but it is a data-structure that describes how the DOM should look like. The code snippet on the top left of the next page shows an example of a Virtual DOM for a simple `<div>`.

```
{ tagName: 'div',
    attributes: {
        class: 'view'
    },
    children: [
        { tagName: 'input', ... },
        { tagName: 'label', ... },
        { tagName: 'button', ... }
    ]
}
```

The Virtual DOM is passed to the DOM-Driver which will update the DOM and wait for the next user input. With this, the cycle ends.

ADVANTAGES

Functional reactive UI development has three major advantages over traditional approaches, all of which are huge: straightforward testing, a comprehensive flow of events, and time travel (yes, seriously).

STRAIGHTFORWARD TESTING

The View()-Function and the ActionCreators are simple mappings, while the Updater performs a fold (also often called a reduce) on the Actions it receives. All components are pure functions and pure functions are extremely easy to test.

The outcome of a pure function depends only on the input parameters and they do not have any side effects. To test a pure function, it is sufficient to create the input parameter, run the "function under test" and compare the outcome. No mockups, no dependency injection, and no complex setup are necessary.

COMPREHENSIVE FLOW OF EVENTS

The control flow of graphical user interfaces is inherently event-based. An application has to react to button-clicks, keyboard input, and other events from users or servers. Applying reactive techniques, be it the Observer Pattern, data-bindings, or reactive streams, comes naturally.

Unfortunately, these techniques come with a price. If a component A calls a component B, it is simple to see the connection in your IDE or debugger. But if both components are connected via events, the relationship is not as obvious. The larger the application becomes, the harder it gets to understand its internals.

The architecture of a functional reactive application avoids these problems by defining a simple flow of events that all components must follow.

Functional reactive UI development has three major advantages over traditional approaches, all of which are huge: straightforward testing, a comprehensive flow of events, and time travel (yes, seriously).

No matter how large your application grows, the flow of events will never change.

TIME TRAVEL

Functional reactive applications allow you to travel back and forth in time. If we store the initial state and all actions, we can use a technique called "Event Sourcing." By replaying the actions, we can recalculate every state the application was in. If we replay only the last n-1, n-2, n-3... actions, we can actually step back in time. And by modifying the recorded stream of actions while applying them, we can even change the past. As you can imagine this can be very handy during development and bugfixing.

The first time-traveling debuggers have been built, but I think we have only started to understand the possibilities, and more amazing tools will be released in the future.

SUMMARY

So far we have only scratched the surface of functional reactive UI development, but by now it should be clear that this approach has tremendous advantages. Most people, who have made the switch to functional reactive UI programming, never want to go back.

MICHAEL HEINRICHS is a user interface creator by passion. He is convinced no matter which technology and which device, if it has a screen, one can build a truly amazing experience. And pure magic. Michael works at the Canoo Engineering AG as a software engineer on next generation user interfaces. Before that, he was responsible for performance optimizations in JavaFX Mobile at Sun Microsystems and later became the technical lead of the JavaFX core components at Oracle. You can find him on Twitter @net0pyr and occasionally he blogs at blog.netopyr.com.

Totally Modular, Dude!

BY **BILL SOUROUR**
FOUNDER AT **DEVMASTERY.COM**

QUICK VIEW

01 Modular programming leads to better readability, collaboration, reusability, and encapsulation. This means fewer bugs and cleaner code.

02 Start using ES2015 module syntax today (you'll need a transpiler like Babel, but it's worth it).

03 Favor exporting a single object or factory function per module defined near the top of your code so that your module's API is immediately apparent.

If you're building a web application today, chances are there's a lot of JavaScript involved. Keeping that JavaScript properly structured so that it's easy to understand, maintain, and change over time is a big challenge. That's why, since the mid-late 2000s most web applications have taken a modular approach.

JavaScript code went from being written as a long list of functions and variables all contained in a single file and attached to a single global scope, to being written as a set of independent, interchangeable modules that each contain all the code necessary to deliver only one aspect of an application's complete, desired functionality.

The trouble is, until the advent of ES2015, JavaScript didn't inherently support this kind of separation, so developers had to rely on a number of clever workarounds and third-party libraries to make modules work.

Now that Modules are officially here, it's worth exploring how best to use them.

WHAT ARE MODULES?

The concept of Modules comes from Modular Programming.

Modular Programming is a software design technique in which the functionality of an application is separated into interchangeable, independent components. In Modular Programming each component is self-contained; meaning that each component contains everything necessary to execute a single, particular aspect of an application's desired functionality. These components are called "Modules."

As of ES2015, JavaScript's definition of a module is far more broad; any file containing exclusively JavaScript code is considered a module.

THE BENEFITS OF MODULAR PROGRAMMING

Modular programming has a number of benefits:

READABILITY

Suppose that the entire text of this article was presented to you as a single, extremely long paragraph. No subheadings. No line breaks. Just an unending stream of words. Chances are that would make it very hard – and quite unpleasant – to read and understand. So, instead, this article contains a number of paragraphs all neatly divided into sections.

Likewise, separating an application into modules makes the application easier to understand and reason about.

This enhanced readability, in turn, makes it easier to find bugs and also to work with legacy code.

COLLABORATION

By breaking an application up into several independent modules, multiple developers can work together on the same application with less fear of damaging or interfering with each other's code.

REUSABILITY

Because of the self-contained nature of Modules, Modular Programming lends itself to the creation of highly re-usable components. In JavaScript, this is evidenced by the rise in popularity of package managers like Bower and npm

ENCAPSULATION

Modules can be selective about what information and functionality they make available to other modules. This prevents unintended consequences and bugs by increasing predictability and control.

Specifically in JavaScript, before ES2015, anything declared outside the scope of a function was automatically added to the global scope. When working with third-party code or even when simply collaborating with other developers on a large project, the risk of name collisions and overwriting values was high.

To solve this, developers resorted to building modules using complicated patterns like Immediately Invoked Function Expressions (IIFE), Asynchronous Module Definitions (AMD), Universal Module

Definitions (UMD), and Common JS (see Appendix A for details).

In ES2015 this paradigm is inverted. Each module creates its own lexical scope, meaning that nothing declared within a module is available or visible to any other module unless it is explicitly exported.

ES2015 EXPORT AND IMPORT SYNTAX

As part of the formal introduction of modules into the JavaScript language, the ES2015 Language Specification also introduced syntax for importing and exporting values to and from modules. The full details can be found in section 15.2.2 and 15.2.3 of the EcmaScript 2015 Language Specification. Below is a summary of the new syntax.

:: EXPORT SYNTAX

```
// default exports
      export default 42;
      export default {};
      export default [];
      export default (1 + 2);
      export default foo;
      export default function () {}
      export default class {}
      export default function foo () {}
      export default class foo {}

// inline exports
      export var foo = 1;
      export var foo = function () {};
      export var bar;
      export let foo = 2;
      export let bar;
      export const foo = 3;
      export function foo () {}
      export class foo {}

// named exports
      export {};
      export {foo};
      export {foo, bar};
      export {foo as bar};
      export {foo as default};
      export {foo as default, bar};

// exports from (re-exports)
      export * from “foo”;
      export {} from “foo”;
      export {foo} from “foo”;
      export {foo, bar} from “foo”;
      export {foo as bar} from “foo”;
      export {foo as default} from “foo”;
      export {foo as default, bar} from “foo”;
      export {default} from “foo”;
      export {default as foo} from “foo”;
```

:: IMPORT SYNTAX

```
// default imports
     import foo from “foo”;
     import {default as foo} from “foo”;

// named imports
     import {} from “foo”;
     import {bar} from “foo”;
     import {bar, baz} from “foo”;
     import {bar as baz} from “foo”;
     import {bar as baz, xyz} from “foo”;

// glob imports
     import * as foo from “foo”;

// mixing imports
     import foo, {baz as xyz} from “foo”;
     import foo, * as bar from “foo”;

// just import
     import “foo”;
```

Source: https://github.com/eslint/espree/pull/43

BROWSER AND NODE.JS SUPPORT

At the time of this writing, the export and import syntax described earlier is not yet fully supported by most browsers, nor is it supported in Node.js.

Edge already has experimental support for the import syntax and both Chrome and Mozilla have announced their intent to support the newly proposed `<script type=”module”>` specification which will add native support for modules in web browsers.

Node.js has not yet committed to adopting the ES2015 standard, but there is a solid proposal in place and a good deal of pressure from the community.

In the meantime, a transpiler (e.g. Babel) is needed in order to use the ES2015 module syntax consistently across all environments and in the browser this should be mixed with a module bundler like Webpack or SystemJS.

BEST PRACTICES

USE THE ES2015 SYNTAX

Despite the current lack of support across browsers and Node.js, now that there is a standard built into the language, it is likely to become the most stable, most widely supported way to write and use JavaScript modules over time.

Use the ES2015 syntax in combination with a transpiler and/or bundler to make your projects “future-ready.”

For legacy projects that already use modules with a different syntax, both SystemJS and Webpack when combined with Babel support projects with mixed module definitions. On the server-side Babel provides support for transpiling ES2015 module definition to CommonJS and also supports mixed module definitions.

STRIVE FOR A SINGLE DEFAULT EXPORT PER MODULE

One of the explicit design goals of ES2015 was to favor default exports; therefore, ES2015 has the most straightforward syntax for default exports.

This implies that modules should have a one-to-one relationship with a self-contained construct such as a function, object, or class.

EXPORT A SINGLE OBJECT THAT REPRESENTS THE MODULE'S PUBLIC API

Inspired by Christian Heilmann's Revealing Module Pattern, an elegant way to write a self-documenting module that makes its API clear and obvious is to encapsulate all publicly available functions and values in a single object and then to expose that object as the default export.

The use of ES2015's shorthand declaration syntax for object literals makes this technique even more powerful.

Consider the following contrived calculator module...

```
// calculator.js
const api = { add, subtract, multiply, divide };
function add(a,b) {...}
function subtract(a,b) {...}
function multiply(a,b) {...}
function divide(a,b) {...}
function somePrivateHelper() {...}
export default api;
```

Defining the API near the top of the module allows anyone reading the code to immediately understand its interface without the need to read the entire listing.

Note: One might expect that an ES2015 destructuring syntax would work when importing this module, but in fact, it does not. Specifically, the following line will fail to work as expected:

```
import ( add, subtract } from 'calculator';
```

This is because the import syntax treats add and subtract as named exports, rather than the destructured properties of an exported object.

EXPORT FACTORIES FOR COMPLEX MODULES

Building upon the aforementioned recommendation, when dealing with a complex object it is advisable to export a single default factory function.

This allows for dependency injection and more sophisticated object construction while respecting the favored, single-default-export paradigm.

Consider the following (even more) contrived calculator module...

```
// calculatorFactory.js
export default function(radix) {
  return {
       add,
       subtract,
       multiply,
       divide,
  };
  function add(a,b) {/* code that uses radix somehow */}
  function subtract(a,b) {/* code that uses radix somehow */}
  function multiply(a,b) {/* code that uses radix somehow */}
  function divide(a,b) {/* code that uses radix somehow */}
  function somePrivateHelper() {/* code that uses radix somehow */}
}
```

Here again the module's API is immediately apparent.

WHAT ABOUT EXPORTING ES2015 CLASSES?

This author prefers not to use ES2015 classes, but that will have to be a topic for another day.

CONCLUSION

The ES2015 specification has finally brought real Modules to JavaScript, further cementing the advocacy of Modular Programming in the JavaScript community. Using Modular Programming in JavaScript promotes better readability, collaboration, reusability, and encapsulation.

While not yet fully supported, third-party libraries make it possible to start using ES2015 today. Adoption of the ES2015 Module standard will make applications "future-ready". So, start now.

APPENDIX A - PRE-ES2015 MODULE PATTERNS

```
:: IIFE
     (function(){ /* code */ }());

:: AMD
     define('myModule', ['dep1', 'dep2'], function (dep1, dep2) {
           return function () {};
  });

:: AMD - Simplified Common JS Wrapping
     define(['require', 'dependency1', 'dependency2'], function
     (require) {
           var dependency1 = require('dependency1'),
           dependency2 = require('dependency2');
     return function () {};
  });

:: Common JS
     function myFunc(){ /* code */ };
      module.exports = myFunc;

:: UMD
     (function (global, factory) {
           if (typeof define === 'function' && define.amd) {
                 // AMD
                 define(['jquery'], factory);
           } else if (typeof exports === 'object') {
                 // Node, CommonJS-like
                 module.exports = factory(require('jquery'));
           } else {
                 // Browser globals (global is window)
                 global.returnExports = factory(global.jQuery);
     }
  }(this, function ($) {
     // methods
     function myFunc(){};
     // exposed public method
     return myFunc;
  }));
```

REFERENCES

ECMAScript® 2015 Language Specification
ecma-international.org/ecma-262/6.0/

Mozilla Developer Network - export
developer.mozilla.org/en/docs/web/javascript/reference/statements/export

Mozilla Developer Network - import
developer.mozilla.org/en/docs/web/javascript/reference/statements/import

ECMAScript 6 Modules: the final syntax
2ality.com/2014/09/es6-modules-final.html

Intent to implement <script type="module> (Chrome)
groups.google.com/a/chromium.org/forum/#!topic/blink-dev/uba6pMr-jec

Bug 1240072 - Implement milestone 0 <script type="module"> (Mozilla)
bugzilla.mozilla.org/show_bug.cgi?id=1240072

HTML Living Standard – Script Element
html.spec.whatwg.org/#the-script-element:module-script

Again with the Module Pattern – reveal something to the world
christianheilmann.com/2007/08/22/again-with-the-module-pattern-reveal-something-to-the-world/

JavaScript Design Patterns
addyosmani.com/resources/essentialjsdesignpatterns/book

BILL SOUROUR is the founder of DevMastery.com. A 20 year veteran programmer, architect, consultant, and teacher, he helps individual developers and billion dollar organizations become more successful every day.

Debugging JS with These 10 Tips

BY FREYJA SPAVEN & DANIEL WYLIE, RAYGUN

JavaScript has a reputation of being not very structured, and it can be hard to get an overview of *what* happened and *when* exactly...

By knowing your tools inside out, you can make a *major difference* when it comes to being able to efficiently debug JavaScript. Easier said than done! Luckily, the team at Raygun has put together this handy list of tips. Most of these are for Chrome inspector, but also there are some great ones for Firefox. Many will work in other inspectors as well. If you find yourself in a bind, just refer back to this guide for a quick and easy debug.

1. `debugger;`

After `console.log`, debugger is my favorite "quick and dirty" debugging tool. Put it in your code, Chrome will automatically stop there when it's executing. Simple as that. You can even wrap it in conditionals so it's only run when you need it.

2. TRY ALL THE SIZES

Whilst having every single mobile device on your desk would be pretty awesome, it's not really feasible in the real world. How about resizing your viewport instead? Chrome provides you with everything you need. Jump into your inspector and click the 'toggle device mode' button. Watch your media queries come to life!

3. HOW TO FIND YOUR DOM ELEMENTS QUICKLY

Mark a DOM element in the elements panel and use it in your console. Chrome inspector keeps the last 5 elements in its history, so the last marked element will be displayed with $0, the second to last marked element $1, and so on.

If you mark items in order 'item-4', 'item-3', 'item-2', 'item-1', 'item-0' then you can easily access the DOM nodes in the console.

4. BENCHMARK LOOPS USING `console.time()` AND `console.timeEnd()`

It can be super useful to know exactly how long something has taken to execute, especially when debugging slow loops and such. You can even set up multiple timers by providing a label to the method.

5. GET THE STACK TRACE FOR A FUNCTION

If you're using a JavaScript framework, you know that it produces a lot of code—quickly. Views are created, events are triggering functions, and in the end you want to know what caused this function call.

Since JavaScript is not a very structured language it can sometimes be hard to get an overview of what happened and *when*, especially when you jump into someone else's code. This is when `console.trace` (or just trace in the console) comes in handy for debugging JavaScript.

Even though you think you know your script well, this can still be useful. Let's say you want to improve your code. Get the trace and your great list of all related functions. Every single one is clickable and you can now go back and forth between these functions. It's like a menu of functions just for you.

6. UNMINIFY CODE AS AN EASY WAY TO DEBUG JAVASCRIPT

Sometimes you have an issue on production and your source maps don't quite make it to the server... fear not. Chrome can unminify your JavaScript files to a more human readable format. Obviously, the code won't be as helpful as your real code—but at least you can actually see what's happening. Click the `{}` Pretty Print button below the source viewer in the inspector.

7. QUICKLY FIND A FUNCTION TO DEBUG

Let's say you want to set a breakpoint in a function.

The two most common ways to do that are:

- to find the line in your inspector and add a breakpoint.
- to add a debugger in your script.
- In both of these solutions, you have to click around in your files to find the particular line you want to debug.

What's probably less common is to use the console. Use `debug(funcName)` in the console and the script will stop when it reaches the function you passed in.

It's quick, but the downside is it doesn't work on private or anonymous functions. But if that's not the case, it's probably the fastest way to find a function to debug. (**Note**: there's a function called console.debug which is not the same thing.)

8. FIND THE IMPORTANT THINGS IN COMPLEX DEBUGGING

In more complex debugging we sometimes want to output many lines. One thing you can do to keep a better structure of your outputs is to use more console functions like: `Console.log`, `console.debug`, `console.warn`, `console.info`, `console.error`, etc. You can then filter them in your inspector. But, sometimes this is not *really* what you want when you need to debug JavaScript. It's now that YOU can get creative and style your messages. Use CSS and make your own structured console messages when you want to debug JavaScript.

For example:

In the `console.log()` you can set **`%s` for a string, `%i` for integers,** and **`%c` for custom style**. You can probably find better ways to use this. If you use a single page framework you may want to have one style for view message and another for models, collections, controllers, etc. Maybe also name the shorter like wlog, clog, and mlog.

It's time to use your imagination.

9. QUICKLY ACCESS ELEMENTS IN THE CONSOLE

A faster way to do a `querySelector` in the console is with the dollar sign. `$('css-selector')` will return the first match of CSS selector. `$$('css-selector')` will return all of them. If you are using an element more than once, it's worth saving it as a variable.

10. BREAK ON NODE CHANGE

The DOM can be a funky thing—sometimes things change and you don't quite know why. However, when you need to debug JavaScript, Chrome lets you pause when a DOM element changes. You can even monitor its attributes. In inspector, right-click on the element and pick a *break on setting* to use.

SUMMARY

These tips should provide you with a starting point when it comes time to debug JavaScript. You can access these same tips and more with thorough examples right here on DZone.

If you'd like to speed up your workflow and add to your toolbox for debugging JavaScript, I also recommend you read:

- Speed Up Your Markup with Zen Coding/Emmet
- Emmet and CSS the Forgotten Part
- Chrome Command Line API
- Chrome Developer Tools, Tips, and Tricks
- Firefox Edit and Resend a Request
- Test Driven JavaScript Development: An Introduction

SURVIVAL INSTINCTS: MOST TRUSTED TECH FOR THE

WEB DEV JUNGLE

You find yourself stranded in the web development jungle and need to find your way out. You hear some vague requirements that you need to 'create a web app.' You don't know how much (or how rich) database access you'll need, how complicated the business logic will be, what your target devices will be, or really anything that imposes specific technical constraints. What is your first instinct in regards to which framework or full stack to use? We received over 700 responses as to which server-side or client-side frameworks come to mind first, as well as which full web stack would be the go-to one. Results are illustrated below. **Welcome to the world wild web.**

CLIENT-SIDE

REACT.JS

11% of respondents would use React.js

React is considered to be simpler and easier to learn that Angular, as well as slimmer. React popularized the concept of unidirectional flow, and uses JSX to compile React to JavaScript at compile time, so bugs are easier to catch.

JQUERY

11% of respondents would use JQuery.

While jQuery's overall popularity has fallen a bit in the wake of mobile app development, it's still very useful for traditional web development. Because it was so popular for so long, it's easy for developers to crank out a quick project with knowledge they already have.

ANGULAR.JS

49% of respondents would use Angular.js

AngularJS popularized the use of custom web components, and simultaneously is easy enough for designers to use and for developers to do almost anything they wanat. While Angular 1 does have issues with performance, Angular 2 has made several improvements.

SERVER-SIDE

SPRING

30% of respondents would use Spring.

Spring is the most popular choice here, because of it's connection to Java. Spring Core offers IoC and dependency injection. And, Spring MVC offers a relatively high-level of abstraction while still providing tight coupling to the Servlet API.

NODE.JS

11% of respondents would use Node.js

Node.js's major strength is that it enables developers to write server-side applications purely in JavaScript. It's highly performant, highly scalable, and highly extensible—all traits that make it highly desirable.

FULL STACK

LSAM (LINUX, SPRING, APACHE, MYSQL)

10% of respondents would use the LSAM stack

This is very telling of our Java-heavy audience! These technologies are likely used together and favorable because of their familiarity.

MEAN (MONGODB, EXPRESS, ANGULAR, & NODE.JS)

15% of respondents would use the MEAN stack

The simplicity, common structure, and performance offered by this framework make it the go-to stack for many web developers. MongoDB's flexibility with data storage, the ability to use JavaScript for client and server-side (Angular & Node.js), JSON format for all data—these are just some of the pros.

Adding Authentication to a Web Application with Auth0, React, and JWT

BY **JOSÉ AGUINAGA**
WEB ENGINEER

QUICK VIEW

01 Online services like Auth0 allow developers and companies to leverage on their infrastructure to ease the entry barrier to any web product that requires authentication.

02 They also let companies Integrate multiple social platforms as entry points for the application, avoiding the necessity of signing up with a new service.

03 And they allow companies to manage and organize their users and the services they consume, reducing the amount of business logic required for the application.

Setting up an authentication layer is, without doubt, one of the most challenging yet necessary tasks within any web application. Not only does the application in question always needs to ensure the most basic functionality is set up by default (such as login, logout, reset password), but additionally, it's required to develop all the libraries to handle the validation of the credentials, the connections to the database responsible for the user data, session management, and general security.

Enter Auth0. Auth0 is an online web service that handles authentication protocols like OAuth2, LDAP, and OpenID Connect to allow clients to create authenticated services without need to build the entire infrastructure. In particular, Auth0 uses the standard RFC 7519 approved by the IETF, better known as JSON Web Tokens (JWT), to communicate with its clients that an authentication flow has been performed. This way, an application can retrieve user related data and showcase protected information to only logged in users.

In the following article we will learn how to use React and Auth0 to enable authenticated-only sections within a web application, as well as to retrieve protected resources. The application will show the logged-in user their GitHub repositories, so we will use our application with Auth0 and OAuth's GitHub integration. Additionally, we will leverage the *auth0-lock* JavaScript library version 10 for performing calls against Auth0, and use the create-react-app library from Facebook to bootstrap our application. Knowledge about *Node.js* and *npm* will be assumed.

LINKS

- Auth0.com
  Free account required for performing OAuth2 with GitHub.
- Create React App
  Required for setting up our sample application.
- Download.Repos.Club
  Repository with final code.

SETUP

The initial setup is fairly easy thanks to Facebook's *create-react-app* utility. Calling the tool with a name will setup a repository with a development workflow toolchain that includes Webpack, Babel, Hot-Reload, and ES6 support.

```
\$ create-react-app download.repos.club
```

After running the command successfully, the folder will have the following structure:

```
── README.md
├── package.json
├── public
│ ├── favicon.ico
│ └── index.html
└── src
├── App.css
├── App.js
├── App.test.js
├── index.css
├── index.js
└── logo.svg
```

We will focus on the **src** folder for our application. As with many React applications, we want to make sure our view is in place

before we perform any business logic on the application. The key view components for our application will be the **Login/Logout** button, and the **Repositories List View**. The files **src/Navbar.js** and **src/Repo.js** showcase each view:

src/Navbar.js

```
import React from 'react'

export const Navbar = ({loggedIn, onClick}) => {
  return (
    <nav className='pt-navbar pt-dark'>
      <div className='pt-navbar-group pt-align-left'>
        <div className='pt-navbar-heading'>Download Repos
</div>
      </div>
      <div className='pt-navbar-group pt-align-right'>
        <button onClick={onClick()} className='pt-button pt-
minimal pt-icon-user'>
          {
            loggedIn
            ? 'Logout'
            : 'Log in'
          }
        </button>
      </div>
    </nav>
  )
}
```

src/Repo.js

```
import React from 'react'
import emoji from 'node-emoji'

import {
  Colors,
  AnchorButton
} from '@blueprintjs/core'

export const Repo = ({name, description, stars, forks,
updatedAt, forked, disabled, downloadRepo}) => (
  <div className='Card pt-card pt-elevation-3'>
    <div className='Repo'>
      <div className='Repo__container'>
        <div className='Repo__headline-container'>
          <span className='Repo__name'>
            <span className={`pt-icon-standard ${forked ? 'pt-
icon-git-branch' : 'pt-icon-git-repo'} Repo__icon`} />
            <b>{name}</b>
            { forked && <span className='pt-tag pt-intent-
warning Repo__tag'>Fork</span> }
          </span>
          <span style={{color: Colors.GRAY1}}className='Repo__
description'>{description ? emoji.emojify(description) : ''}</
span>
        </div>
        <div className='Repo__metadata-container'>
          <span className='Repo__stars'><b>{stars}</b> stars</
span>
<span className='Repo__forks'><b>{forks}</b> forks</span>
        </div>
      </div>
      <div className='Repo__container'>
        <span className='Repo__latest-update'>Last Updated:
<b>{new Date(updatedAt).toDateString()}</b></span>
```

Continued top right

```
        </div>
      </div>
      <div className='Repo__container'>
        <span className='Repo__latest-update'>Last Updated:
<b>{new Date(updatedAt).toDateString()}</b></span>
        <div className='Repo__actions-container pt-button-group'>
          <AnchorButton href={downloadRepo} download={`${name}.
zip`} iconName='download'> Download </AnchorButton>
          {/*<Button className='pt-intent-danger' iconName='delete'
disabled={disabled}> Delete </Button>*/}
        </div>
      </div>
    </div>
  </div>
)
```

Note: We are using Palantir's Blueprint React components module to display some of the objects in our application. Visit blueprintjs.com to learn more about BlueprintJS.

As we can see, both our view components receive a series of parameters, particularly the **src/Repo.js**, that contains authenticated information. We need to use Auth0's library Lock to help us retrieve that information from our user. Including *auth0-lock* in our library and adding an ES6 class wrapper around it ensures we can request the user GitHub credentials. The class, which will be located in **src/Auth.js**, will look something like this:

src/Auth.js

```
import Auth0Lock from 'auth0-lock'

export default class Auth {
  constructor (clientId, domain, callback) {
    // Configure Auth0
    this.lock = new Auth0Lock(clientId, domain, { redirect: true,
allowSignUp: false })
    // Add callback for lock `authenticated` event
    this.lock.on('authenticated', this._doAuthentication.bind(this,
callback))
    // binds login functions to keep this context
    this.login = this.login.bind(this)
  }

  _doAuthentication (callback, authResult) {
    this.setToken(authResult.idToken)
    callback()
  }

  login () {
    // Call the show method to display the widget.
    this.lock.show()
  }

 loggedIn () {
    // Checks if there is a saved token and it's still valid
    return !!this.getToken()
  }

  setToken (idToken) {
    // Saves user token to localStorage
    localStorage.setItem('id_token', idToken)
  }
```

Continued next page

```
  getToken () {
    // Retrieves the user token from localStorage
    return localStorage.getItem('id_token')
  }

  logout () {
    // Clear user token and profile data from localStorage
    localStorage.removeItem('id_token')
  }
}
```

The most important parts of this class are the **constructor** and **login** method. The constructor initializes the Auth0 library with your own credentials, while the login will trigger the Auth0 modal display for showing the user credentials. Since we are only interested in our user's GitHub information, we don't want to allow signing up through Auth0. This can be specified as a parameter to Lock. You can read more about Lock in auth0.com/docs/libraries/lock.

Before we continue, we need to retrieve our **Client ID** and **Domain** from Auth0. Additionally, ensure that *localhost* is included in the list of "**Allowed Callback URLs**" and "**Allowed Origins (CORS)**" within the Auth0 dashboard. To be able to use GitHub as a Connection, you need to create a new application under **Settings > OAuth Applications** in GitHub, and pass both **Client ID** and **Client Secret** from the new registered application. Add Auth0's callback URL within the GitHub application, which should be in the form *https://<user>.auth0.com/login/callback*.

If everything has been done correctly, we should be able to wire our application with the Auth class with the button, and display the Auth0 modal inside.

We are now able to log in our user, but we still need to fetch its data. That's where JSON Web Tokens (JWT) come in. On a successful login, src/Auth.js will store the JWT of the authenticated request as an id_token inside our local storage. We can then use this token to request Auth0's profile information of the user. Since JWTs are encoded with a secret that only Auth0 knows, they are safe to send and receive in a client-side application. We will retrieve the JWT through an API class, and then perform the required queries to both Auth0 and GitHub. Our now src/Api.js looks like this:

```
import axios from 'axios'
import Auth from './Auth'

export default class Api {
  constructor (callback) {
    this.auth = new Auth('xRTGXVGR03uOlQMRds6ZpU0fx8OjLakE',
'jjperezaguinaga.auth0.com', callback)
  }

getRepos () {
    return this.isLoggedIn()
    ? this.getProfile()
    : Promise.reject(new Error('User is not authenticated'))
  }
```

Continued top right

```
  async getProfile () {
    const profile = await axios.post('https://jjperezaguinaga.
auth0.com/tokeninfo', {id_token: this.auth.getToken()})
    return Promise.all([
      profile.data,
      axios.get(`https://api.github.com/users/${profile.data.
nickname}/repos?per_page=100&sort=updated`)
    ])
  }

  isLoggedIn () {
    return this.auth.loggedIn()
  }
  login () {
    this.auth.login()
  }
  logout () {
    this.auth.logout()
  }
}
```

We are taking advantage of both Promise and Axios to perform asynchronous requests to our endpoints, and resolve them gracefully, respectively. Additionally, we use await and async to control the flow of our requests, and chain each response to then retrieve the user GitHub repositories. Finally, src/Api.js provides us with utilities to communicate with our src/Auth.js class and log out whenever we don't need the resources anymore.

CONCLUSION

The process we used to authenticate our user is known as the Implicit Grant flow defined by RFC 6749, better known as OAuth2. Due to the security restrictions of any client-side application, one can perform limited operations against the Resource Server. In order to use Auth0 to perform calls against an API (such as creating a new repository in GitHub, for instance), it's required to set up a server that is able to perform the access_token handshake with the Resource Server. However, even without a server, we are able to retrieve basic information about our user, such as their profile and repositories. A few years ago it would have been required to have a full-blown dedicated server and database to perform such tasks.

Although we picked Auth0 to showcase this flow, it's important to mention that Auth0 is not the only service in the authentication-as-a-service industry. Amazon Cognito and Stormpath provide similar solutions, both with their pros and cons. Particularly, Amazon Cognito interacts perfectly with the AWS ecosystem, and might be a better option over Auth0 if the consumers of your application are working with AWS-related resources.

JOSÉ AGUINAGA is a Web Engineer with multiple years of experience in JavaScript-related technologies. Having worked for different startups in various cities across the world like Zürich, San Francisco, México City, and Bali, José has developed an insider's understanding of the startup culture and the web development industry, especially within the Fintech ecosystem.

Diving Deeper

INTO WEB DEV

TOP #WEBDEV TWITTER FEEDS

To follow right away

 @jdalton

 @swizec

 @davidwalshblog

 @rachelandrew

 @addyosmani

 @collis

 @rauschma

 @rasmus

 @paul_irish

 @chriscoyier

TOP WEB DEV REFCARDZ

Node.js

dzone.com/refcardz/nodejs

Developing distributed, multi-threaded applications using traditionally synchronous languages can be complex and daunting, Node leverages JavaScript's asynchronous programming style via the use of event loops with callbacks to make applications naturally fast, efficient, and non-blocking.

Functional Programming in JavaScript

dzone.com/refcardz/functional-programming-with-javascript

Functional programming is a software paradigm that will radically change the way in which you approach any programming endeavor. Combining simple functions to create more meaningful programs is the central theme of functional programming.

AngularJS

dzone.com/refcardz/angularjs-essentials

Provides an essential reference to AngularJS, an MVW framework that enables web developers to build dynamic web applications easily.

WEB DEV PODCASTS

The Shop Talk Show

Front End Happy Hour

Three Devs and a Maybe

WEB DEV ZONES

Learn more & engage your peers in our Security-related topic portals

Web Dev

dzone.com/webdev

Web professionals make up one of the largest sections of IT audiences; we are collecting content that helps web professionals navigate in a world of quickly changing language protocols, trending frameworks, and new standards for user experience. The Web Dev Zone is devoted to all things web development—and that includes everything from front-end user experience to back-end optimization, JavaScript frameworks, and web design. Popular web technology news and releases will be covered alongside mainstay web languages.

Java

dzone.com/java

The largest, most active Java developer community on the web. With news and tutorials on Java tools, performance tricks, and new standards and strategies that keep your skills razor-sharp.

Mobile

dzone.com/mobile

The Mobile Zone features the most current content for mobile developers. Here you'll find expert opinions on the latest mobile platforms, including Android, iOS, and Windows Phone. You can find in-depth code tutorials, editorials spotlighting the latest development trends, and insight on upcoming OS releases. The Mobile Zone delivers unparalleled information to developers using any framework or platform.

WEB DEV BOOKS

Secrets of the JavaScript Ninja
by John Resig and Bear Bibeault

Programming Ruby 1.9 & 2.0: The Pragmatic Programmers' Guide
by Dave Thomas, Andy Hunt, and Chad Fowler

Python Cookbook
by David Beazley, and Brian K. Jones

Qlik Playground™

```
try {
 QlikPlayground.learn();
} catch (Exception e) {
 System.out.println("never called");
} finally {
 me.setGenius(true);
}
```

Create genius data-driven apps with built-in features like search and associative modeling.

Qlik Playground is a free, one-of-a-kind programming environment that lets developers get hands-on with their data—without all the hassle. Built for developers by developers, Playground makes it easy to import your data to start building, playing and exploring right away. Backed by an open source community, it works with any stack, architecture or platform so you can get ramped-up fast. Skip the boring tasks like writing queries and creating tables and start building the stuff you love.

Come play at **playground.qlik.com**

Qlik

How Reactive Programming Can Elevate Your Data-Driven Development

Over 2.5 quintillion bytes of data are generated each day across the internet—a never-ending onslaught of information that requires a level of programming that can capture, manipulate, and transform that data into actionable and relevant knowledge. Reactive programming is this new paradigm: a powerful evolution in data-driven application and website development.

With an unprecedented number of internet users browsing existing data, and creating more data, it's critical that websites and applications can respond to user inquiries with quick, accurate information. This means applications must be scalable, resilient, and built on a foundation that is event-based and message-driven.

Application scalability and resiliency are paramount in order to meet expanded user expectations, minimize downtime, and stay ahead of competitors. A message-driven architecture provides an asynchronous boundary needed to decouple from the strict time-and-space boundaries that monolithic applications must conform to. This decoupling can endow the application with elasticity, or the ability to easily scale out on demand. These attributes—message-driven architecture and elasticity—are the cornerstones of reactive programming.

A reactive approach to data-driven development also allows for more flexibility, making websites and applications easier to develop and to change, as well as more tolerant of failure, which makes releasing patches on highly-available infrastructure possible.

At Qlik, we offer a complete set of reactive tools and an engine through a responsive user interface that utilizes WebSockets and real-time communication to quickly and easily associate, analyze, and update data across servers and applications. Gone are the days when developers have to manually and tediously update their code for each new data source or change in the data. As any CIO can tell you, the ongoing management of all of the different data sources and processing engines can be a huge amount of overhead: overhead that opens the door to mistakes and closes the door to time for new projects and developments.

Application scalability and resiliency are paramount in order to meet expanded user expectations, minimize downtime, and stay ahead of competitors.

The reactive way of developing with the Qlik Engine gives developers the tools to set up the data from the onset to automatically react to changes. Using tools like ReactJS, Redux, Flux, and asynchronous programming, developers can use Qlik to quickly build an event-driven application that incorporates visualizations and analytics on huge data sets. Once your dashboard is built out and your organization's data is displayed in reactive charts and other UI elements, they are hooked up to the engine. Every time the user makes a selection or drills down to explore a particular use case, the engine automatically recalculates and updates each chart. No extra coding is necessary.

To try this out for yourself, Qlik has developed an open sandbox architecture for developers. Qlik PlaygroundTM is a free programming environment that allows developers the chance to check out Qlik's engine without commitment. Once an account is created, just load in your external data and begin to quickly and effortlessly create your data-driven application.

Sources: *https://medium.com/reactive-programming/what-is-reactive-programming-bc9fa7f4a7fc#.ah2e144gi*

WRITTEN BY DAVE NUGENT
DEVELOPER RELATIONS ENGINEER, QLIK

Executive Insights on Web Application Development

BY **TOM SMITH**
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01 Keys to developing web applications are planning, UX, and using appropriate tools and technologies to optimize developer productivity.

02 The most significant changes to the development of web apps are platforms, tools, and the speed with which they have enabled developers.

03 Web apps enable speed to market securely and automatically, making customers' lives simpler and easier across a variety of industries.

To gather insights on the state of web application development today, we spoke to 13 executives from 12 companies developing web applications or providing web application development tools to their clients. Specifically, we spoke to:

SAMER FALLOUH, Vice President, Engineering and
ANDREW TURNER, Senior Solution Engineer, Dialexa
ANDERS WALLGREN, CTO, ElectricCloud
BRENT SANDERS, CEO, Fulton Works
CHARLES KENDRICK, CTO, Isomorphic Software
ILYA PUPKO, V.P. of Product Management, Jitterbit
FAISAL MEMOM, Product Marketing, NGINX
BRUNO CORREA, IT Coordinator, Ranstad (Brazil)
CRAIG GERING, Vice President, Engineering, Sencha
JOACHIM WESTER, Founder, Starcounter
MICHAEL MORRIS, CEO, Topcoder
GREG LAW, CEO, Undo
ALEXEY AYLAROV, CEO, Voximplant

KEY FINDINGS

01 The keys to developing sound web applications are **planning, user experience (UX), and using the appropriate tools and technology to optimize developer productivity**. It's important to think through the requirements up front in order to choose the most appropriate platform, technology, and tools to deliver the project in the smartest and most efficient way. In order to do this, you must understand who and what you are building for and the definition of a "successful" application. Your application must provide a seamless UX across any browser and any device—including IoT—the right experience, on the right device, at the right time. Automate the process, including testing, with DevOps/CI/CD. Anything that improves the quality of life for developers will improve the quality of applications.

02 The most significant changes to the development of web applications are **platforms, tools, and the speed with which they have enabled developers** to develop applications. Frameworks have matured, build systems have evolved, there are new architectural patterns, and movement to the cloud is enabling the standardization of advanced technology stacks. There's an increasing set of tools for all budgets. Tools have become extremely easy to use and many are almost like 20th century magic that "write code for you." Applications are developed and brought to the end user more quickly with design sprints that enable the development team to work more quickly.

03 The technical solutions used most frequently to develop web apps are **JavaScript, AngularJS, ReactJS and Native, Node.js, and HTML5**. Other solutions mentioned include: AWS, CSS, Docker, Ember, Java, NGINX, open source, Selenium, TypeScript, and Webpack.

04 Real-world problems solved by web applications are diverse with **speed to market—securely and automatically—and making customers' lives simpler and easier across a variety of industries** most frequently mentioned. Applications with APIs integrate with other applications automatically. A manufacturer of telecom chips uses process controls and predictive analytics to ensure chips will meet FCC regulations. Apps are empowered with voice

and video calls. An app for NASA is helping astronauts keep up with their food intake. A staffing company in Brazil used an app to move from 75,000 job applicants, to 7,000 candidates who self-scheduled interviews picking the time and location most convenient for them. Airport dashboards, fleet tracking, car management, lot management, moving services, oil and gas taxing, device tracking, social media ranking, class pass for kids, kids healthy eating, LA metro patrons able to update fare balances, and students completing all of the steps in their enrollment—are just a few ways applications are improving life.

05 The most common issues affecting the development of web applications are the **increased complexity of the platforms, tools, and languages and the lack of planning that goes into ensuring you are using the best solution for the problem at hand.** The most prominent issue is the number of tools and languages available to accomplish the same thing and to be able to discern which solution to invest in. New JavaScript frameworks, versions of .NET, and Java are emerging every week. Things tend to get too technical and too hard to integrate. Teams need to support a wide variety of device types, browsers, and operating systems, and the landscape is always changing with new versions and updates being released. In addition, people don't think through what's needed to solve the problem. Many people with inadequate backgrounds and conscientiousness are releasing applications that are neither secure nor user friendly. Instead of doing some basic pre-planning, authors choose to just jump in and develop what they think is the solution to the problem rather than determining the end-user need and problem to be solved.

06 The future of web applications will **ultimately integrate across all devices providing a great user experience (UX).** Virtual tools will continue to make web application development faster, simpler, and easier. Applications will integrate across devices with APIs providing a more integrated ecosystem. They will also be more integrated into our daily lives with IoT, wearables, and practical applications that make our lives easier and more manageable. Progressive web apps will provide the performance and functionality of native apps combined with the convenience of the web. In the end, the web wins due to its openness, flexibility, and convenience. WebAssembly technology will allow developers to use any programming language to create web apps. Rapid application development will kill complexity while increasing development speed.

07 The biggest concerns around the development of web applications are **security and complexity—which can be addressed with planning.** Security is still a big issue, with many websites not defaulting to SSL, and we're replacing more "offline" apps with their web counterparts. Security needs to be done correctly with web apps and IoT since this will affect our homes, our cars, and our medical devices. There's still a lack of planning upfront and this becomes critical as complexity increases. We need more standardization across browsers. Breaking changes introduced by new versions of consumed frameworks can be a nightmare. This increases the need to choose the right technology platform for the job.

08 The skills developers need to develop effective web applications are: **agility, knowledge of the fundamentals, and knowing themselves.** The technical landscape continues to rapidly evolve, and developers must be able to adapt to the changes. Great developers will freely abandon their own solutions when better ones come along. The best developers don't always have the strongest technical experience, but they produce great work because they listen to, and empathize with, the end users and can honestly and objectively judge their own work to ensure they're meeting the needs of the end user. Know the components of the application and the database frameworks. Understand the glue code, but don't get too involved with it. The attitude of developers needs to scale with deployment and DevOps to bridge the gap. Developers need to become responsible for the operation of their code. Lastly, developers need to focus on what they are good at and what they're passionate about. Don't be afraid to say, "no, I'm not the best fit for this." Focus on your strengths.

09 Additional consideration with regard to developing web applications were wide ranging:

- Productivity is about to see a dramatic change with **artificial intelligence**.
- **What are the big conferences for JavaScript**? Is there a JavaOne equivalent?
- **Lack of thinking through requirements up front** is killing a lot of applications today.
- **What's your strategy for getting access to the IT talent you need?** How do you plan to leverage the on-demand workforce? How does crowdsourcing and virtual labor fit into your company?
- **Does the business have clear visibility about how web development will help them achieve their goals?**
- **What are the dynamics of an effective team?** Are people working as a team to build something together?
- **How does a corporate culture impact software development and teamwork?** Are DevOps/CI/CD driving application development or is application development driving the move to DevOps?
- To ensure companies make the right technology choices, **IT managers, developers, and executives need to work together** and trust each other to solve business problems. Forming cross-functional teams is important for building relationships, so each group understands the value and perspective of the others.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Embracing Simplicity with Static Site Generators

BY **RAYMOND CAMDEN**
DEVELOPER ADVOCATE AT **IBM**

QUICK VIEW

01 Static sites can be beneficial by providing an incredibly simple output that can be deployed anywhere quickly and cheaply.

02 By acting like an app server locally, the developer gets the benefit of a dynamic language on their local machine with the benefit of static output when done.

03 Even after moving to a static solution, many dynamic aspects, like forms and comments, can be added in via third-party services.

For many years developers have made use of various application servers to create dynamic websites. From PHP to ASP to ColdFusion and more recently Node.js, app servers let you integrate databases and other dynamic features into your websites. But more recently, many developers are beginning to realize that app servers may be more power than they need. Static site generators, a relatively recent innovation, provide developers with the nimbleness of an app server while working locally and outputting simple static files for easy deployment.

Given that an app server can be so powerful, why would a developer willingly give that up?

1. One of the biggest reasons is that an app server can go down. Every app server needs to be updated, monitored, and generally babysat to ensure it's running well and secured. On top of that, app servers are typically paired with database servers, which require the same time and care.
2. Many times app servers provide way more power than necessary. Do you really need PHP for a site getting less than a thousand hits per month? Do you need MySQL for data that changes rarely?
3. Another big reason to avoid app servers is that our websites are doing less and less now that so many people have moved to apps on their mobile devices. Your website may have turned into simply an announcement telling people to download your app.
4. Finally—browsers are much more powerful now than they were years ago. Things that app servers had to do because the browsers couldn't handle it may be better handled on the client side.

For this reason and more, static site generators provide a great alternative. A typical generator will do the following for you:

1. They will give you a template language. While not a real "programming" language per se, these template languages let you embed tokens in your code that will be replaced on the fly.
2. To replace those tokens, they provide a way for you to store data. This is outside a database server and is generally something like a simple JSON file.
3. They provide a way to preview your site on your local machine, running much like a traditional app server.
4. Then they provide a way to output to simple, static files.

Sound interesting? The good news is that you have more than a few options. The list at staticsitegenerators.net currently has nearly 450 different options. Every static site generator will have its own way of doing things while generally following the process listed above. For this article, let's take a look at a rather simple generator, Harp (harpjs.com).

WORKING WITH HARP

Harp isn't the best generator out there, and it isn't even my personal favorite, but it was the first one I encountered and was responsible for starting my love for generators in general. It's also one of the simplest ones to get started with, and that makes it a good option to discuss. Keep in mind, though, that if you don't like the particulars of how it works, you still have over four hundred other ones to try. (I'll discuss some other options at the of this article as well.) To begin, simply `npm install -g harp`. This will add the Harp command line to your system. To confirm, just run `harp` at the command line.

```
trash$ harp

  Usage: harp [options] [command]

  Commands:

    init [options] [path]  Initialize a new Harp project in current
directory
    server [options] [path] Start a Harp server in current
directory
    multihost [options] [path] Start a Harp server to host a
directory of Harp projects
    compile [options] [projectPath] [outputPath] Compile project
to static assets (HTML, JS and CSS)

  Options:

    -h, --help     output usage information
    -V, --version  output the version number

  Use 'harp <command> --help' to get more information or visit
http://harpjs.com/ to learn more.
```

Harp's command line mainly revolves around three options—making a new site, running the server, and outputting static flat files. Let's begin by creating a new site: `harp init demo1`.

```
trash$ harp init demo1
Downloading boilerplate: https://github.com/harp-boilerplates/
default
Initialized project at /Users/raymondcamden/Desktop/trash/demo1
```

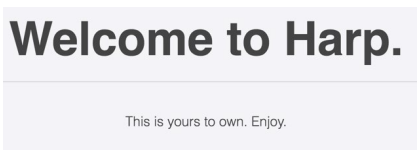
This will create a simple site with four files:

- _layout.jade handles creating the layout for the site.
- 404.jade is a 404 handler for the site.
- Index.jade is the home page.
- main.less is a style sheet written using Less

For rendering HTML, Harp supports Markdown, EJS, and Jade. For rendering CSS, Harp supports Less, Stylus, and Sass. You can even use CoffeeScript as well. Harp automatically converts the particular language into the right output. Let's see this in action. At the command line again, run `harp server`.

```
demo1$ harp server
------------
Harp v0.23.0 - Chloi Inc. 2012-2015
Your server is listening at http://localhost:9000/
Press Ctl+C to stop the server
------------
```

If you open this up in your browser, you'll see the following:



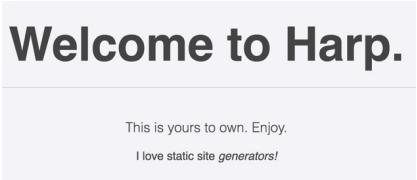
So how did this work? By default, Harp's initial files are all using the Jade template language. Jade was recently renamed to Pug, and you can find documentation about all it supports at its website: pugjs.org. But for now, open up index.jade:

```
h1 Welcome to Harp.
h3 This is yours to own. Enjoy.
```

Even if you've never seen Jade before, you can probably guess as to how it is working. The first line is creating an h1 block with text inside it, and the second line is creating an h3 block. Modify the template like so:

```
h1 Welcome to Harp.
h3 This is yours to own. Enjoy.
p I love static site
    i generators!
```

All we've done is added a new paragraph with a bit of styled text. Reload the page in your browser and you'll see the new output:

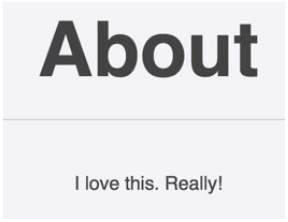


If you view the source, you'll see HTML, not Jade. This is—pretty much—like every other old app server out there. PHP acts the same way.

Now let's create another file, about.md:

```
About
===
I love this. Really!
```

Harp let's you freely mix and match different template engines within one site, although probably you'll stick to one for your project. You can open this new file by requesting about.html in the browser. Harp is smart enough to know this maps to the Markdown file you just created.



Where did the site's rather simple layout come from? The _layout.jade file:

```
doctype
html
  head
    link(rel="stylesheet" href="/main.css")
  body
    != yield
```

In the Jade template above, the critical part is `yield`. Harp automatically runs the layout file for every file you request—that's how it worked for both index.jade and about.md. It takes the content from those files and places it in a variable called yield. This then allows your template to put the page content anywhere you want. Here's a modified version that adds a footer.

```
doctype
html
    head
        link(rel="stylesheet" href="/main.css")
    body
        != yield
    footer This site is copyright #{ new Date().getFullYear() }
```

Note the user of a bit of code at the end to include a dynamic year. Now when you run the home page, you'll see (below, left):

Now let's talk a bit about data. We mentioned that all static site generators have a way to let you define data for use in the site, and Harp is no different. Harp provides two different ways of defining data, but for now, we'll look at the simpler of the two options—global data. Add a new file to your site named _harp.json. Harp will not render any file that begins with an underscore. In this file, include the following:

```
{
    "globals":{
        "email":"raymondcamden@gmail.com",
        "name":"Awesome Site!"
    }
}
```

In order to use global data, you must use the file named _harp.json (there's an exception to this, but don't worry about that for now) and an object defined with the name `globals`. Outside of that, you can then use whatever variables you want. In the example above, two variables, email and name, are defined, but that's completely arbitrary. You can include any valid JSON you want here.

Once defined, these variables will be available in all your files. Here's an updated _layout.jade file that includes the name value as the title:

```
doctype
html
    head
        title #{ name }
        link(rel="stylesheet" href="/main.css")
    body
        != yield
    footer This site is copyright #{ new Date().getFullYear() }
```

And here's a modified home page using the email:

```
h1 Welcome to Harp.
h3 This is yours to own. Enjoy.
p I love static site
    i generators!
p Email me at
    a(href="mailto:#{email}") #{ email }
```

While not as powerful as a proper database, we now have one place where we can define dynamic values and have them automatically updated throughout the site.

For the coup de grace, let's now kill off this server and output in static files. That takes all of one command: `harp compile -o ../demo1_output`

This tells Harp to compile the current directory and output it one level higher in a folder called demo1_output. This will create four files: 404.html, about.html, index.html, and main.css. If you open index.html you'll notice the layout and variables have all been rendered. You can now take these files and deploy them as a simple static site.

Harp has more features than we've covered here so be sure to check the harpjs.com/docs for more information!

ADDING DYNAMIC BACK

So obviously once you've gone static you end up losing a few simple things you may miss. One example of that is form processing. How do you add, for example, a basic contact form to your site so visitors can contact you? Luckily there's a few options for this. One of the best, and easiest, to use is Formspree (formspree.io). With Formspree, you can simply point your form to their server, include your email address, and they will forward along the results to you. Consider the following example:

```
Contact Me
===
<form action="https://formspree.io/raymondcamden@gmail.com"
method="POST">
    Your Name: <input type="text" name="name"><br/>
    Your Email: <input type="email" name="_replyto"><br/>
    Your Comments: <textarea name="comments"></textarea><br/>
    <input type="submit" value="Send Comments">
</form>
```

In this rather simple (and kinda ugly) form, the action points to Formspree and includes my email address. Now when the form is submitted, Formspree will take the contents and email them to me. The very first time this happens Formspree will require you to confirm that you want the form to work, but after that it's automatic. You can even tell Formspree where to send the user next by including a hidden form field: `<input type="hidden" name="_next" value="some url on your site telling the user thank you">`.

Formspree is a great service with a very generous free tier. Check the site for current costs and what's supported at that level.

On top of forms, you can also find services for calendars, comments, and other dynamic aspects.

WRAP UP

While obviously static site generators aren't going to be a good option for every site, it's amazing how many places they can be used. As much as I consider myself a decent developer, I absolutely love the idea of my production websites being as simple, and stupid, as possible. The fewer things I have to worry about breaking down is always a good thing!

RAYMOND CAMDEN is a developer advocate for IBM. His work focuses on APIs, Bluemix, hybrid mobile development, Node.js, HTML5, and web standards in general. Along with Brian Rinaldi, he is the author of the soon-to-be released book on Static Sites from O'Reilly—(Working with Static Sites). Raymond can be reached at his blog (raymondcamden.com), @raymondcamden on Twitter, or via email at raymondcamden@gmail.com.

Solutions Directory

This directory contains solutions for debugging platforms, platforms as a service, device detection, various web dev frameworks, web testing, IDEs, and more. It provides feature data and product category information gathered from vendor websites and project pages. Solutions are selected for inclusion based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY NAME	PRODUCT	TYPE	FREE TRIAL	WEBSITE
ActiveState	Komodo IDE	PaaS	21 Days	activestate.com/komodo-ide
Afilias Technologies	DeviceAtlas	Device Detection	30 Days	deviceatlas.com
Amazon	AWS Elastic Beanstalk	PaaS	Free Tier Available	aws.amazon.com/elasticbeanstalk
Aryaka	SmartCDN	Web App Acceleration	Available by request	aryaka.com/services/web-app-acceleration
Automattic	Wordpress	CMS	Free Tier Available	wordpress.com
Backtrace	Backtrace	Debugging Platform	Available by request	backtrace.io
Bithound	Bithound	Node.js Debugging	Free Tier Available	bithound.io
Blue Spire	Aurelia	JavaScript Framework	Open Source	aurelia.io
Browserling	Browserify	Build Script Tool,ÊPackage Manager	Open Source	browserify.org
BrowserStack	BrowserStack	Web Testing	Available by request	browserstack.com
Cake Foundation	CakePHP	PHP Web Framework	Open Source	cakephp.org
ClojureScript	ClojureScript	Compiler	Open Source	clojurescript.org
CoffeeScript	CoffeeScript	Language	Open Source	coffeescript.org
CommonJS	CommonJS	Package Manager	Open Source	commonjs.org
Django Software Foundation	Django	Python Web Framework	Open Source	djangoproject.com
DocumentCloud	Backbone.js	JavaScript Framework	Open Source	backbonejs.org
DocumentCloud	Underscore.js	JavaScript Library	Open Source	underscorejs.org
Dropbox	Dropbox platform	PaaS	Available by request	dropbox.com/developers
Drupal	Drupal	CMS	Open Source	drupal.org

COMPANY NAME	PRODUCT	TYPE	FREE TRIAL	WEBSITE
Ecma International	ECMAScript	Scripting Language	Open Source	github.com/tc39/ecma262
Elm	Elm	Language	Open Source	elm-lang.org
Engine Yard	Engine Yard Cloud	PaaS	21 Days	engineyard.com
Facebook	React.js	JavaScript Library	Open Source	facebook.github.io/react/
Facebook	Flow	Static Type Checker	Open Source	flowtype.org
Fujitsu	Fujitsu S5 Cloud Service	PaaS	No Free Trial	welcome.globalcloud.global.fujitsu.com
GNU	Make	Build Script Tool	Open Source	gnu.org/software/make
Google	AngularJS	JavaScript Framework	Open Source	angularjs.org
Google	Google Cloud Platform	PaaS	60 Days	cloud.google.com
Grails	Grails	Java Web Framework	Open Source	grails.io
Grunt	Grunt	Build Script Tool,ĖPackage Manager	Open Source	gruntjs.com
Hewlett Packard Enterprise	HPE Helion	PaaS	Available by request	hpe.com/us/en/solutions/cloud.html
Hewlett Packard Enterprise	HPE Unified Functional Testing	Web Testing	Open Source	www8.hp.com/us/en/software-solutions/unified-functional-automated-testing/
IBM	Bluemix	PaaS	30 Days	ibm.com/bluemix
IBM	LoopBack	Node.js API Framework	Open Source	loopback.io/
Intel	Crosswalk	Web Runtime	Free Solution	crosswalk-project.org
Jelastic	Jelastic	PaaS	Varies by hosting provider	jelastic.com
JetBrains	WebStorm	IDE	30 Days	jetbrains.com/webstorm
JetBrains	PyCharm	IDE	Free Tier Available	jetbrains.com/pycharm
JetBrains	RubyMine	IDE	30 Days	jetbrains.com/ruby
JetBrains	PHPStorm	IDE	30 Days	jetbrains.com/phpstorm
JetBrains	Rider	IDE	Available through early access	jetbrains.com/rider/
JetBrains	Resharper	Debugging Platform	30 Days	jetbrains.com/resharper
Langa	Trails	Node.js Web Framework	Open Source	trailsjs.io
Laravel	Laravel	PHP Web Framework	Open Source	laravel.com
LeaseWeb	LeaseWeb	IaaS	Available by request	leaseweb.com/cloud
Less	Less	CSS Preprocessor	Open Source	lesscss.org

COMPANY NAME	PRODUCT	TYPE	FREE TRIAL	WEBSITE
Lightbend	Play	Java Web Framework	Open Source	playframework.com
Mendix	Mendix Platform	PaaS	Up to 10 Users	mendix.com
Meteor Development Group	Meteor	JavaScript Framework	Open Source	meteor.com
Microsoft	Visual Studio Code	IDE	Free Solution	code.visualstudio.com
Microsoft	ASP.NET	Web Framework	Free Solution	asp.net
Microsoft	Microsoft Azure	PaaS	Free Tier Available	azure.microsoft.com
MochaJS	Mocha	Web Testing	Open Source	mochajs.org
NearForm	Hapi	JavaScript Framework	Open Source	hapijs.com
Node.js Foundation	Node.js	JavaScript Environment	Open Source	nodejs.org/en
Node.js Foundation	Koa	Web Framework	Open Source	koajs.com
Node.js Foundation	Express	Web Framework	Open Source	expressjs.com
Nodesource	NISolid	Node.js Runtime	Available by request	nodesource.com
npm, inc.	npm	Package Manager	Open Source	npmjs.com
Open Source Matters	Joomla!	CMS	Open Source	joomla.org
OpenQA	Selenium	Web Testing	Open Source	seleniumhq.org
Optimizely	Optimizely	Web Testing and Experimentation	30 Days	optimizely.com
Oracle	Oracle Cloud	PaaS	Available by request	cloud.oracle.com
OutSystems	OutSystems	PaaS	Free Tier Available	outsystems.com
PhantomJS	PhantomJS	Web Testing	Open Source	phantomjs.org
Pivotal Labs	Jasmine	Web Testing	Open Source	jasmine.github.io
Pivotal Software	Spring	Java Web Framework	Open Source	spring.io
Progress Software	OpenEdge	PaaS	60 days	progress.com/openedge
Qlik	Qlik Playground	Web App Testing Environment	Open Source	playground.qlik.com
QuickBase, Inc.	QuickBase	PaaS	30 days	quickbase.com
Raygun	Pulse	Real User Monitoring	30 days	raygun.com/products/real-user-monitoring
Red Hat	OpenShift	PaaS	Free Tier Available	openshift.com
RequireJS	RequireJS	Package Manager	Open Source	requirejs.org
RisingStack	Trace	Node.js Debugging	Free Tier Available	trace.risingstack.com

COMPANY NAME	PRODUCT	TYPE	FREE TRIAL	WEBSITE
Rogue Wave Software	Zend Framework	PHP Web Framework	Open Source	framework.zend.com
Ruby on Rails	Ruby on Rails	Web Framework	Open Source	rubyonrails.org
Salesforce	Heroku Platform	PaaS	Free Tier Available	heroku.com
SAP	OpenUI5	JavaScript UI Library	Open Source	openui5.org
SAP	SAP HANA Cloud Platform	PaaS	Free Tier Available	hcp.sap.com
Sass	Sass	CSS Preprocessor	Open Source	sass-lang.com
Sauce Labs	Sauce	Web Testing	14 Days	saucelabs.com
Sencha	Sencha Platform	Web App Platform	30 Days	sencha.com/web-application-lifecycle-management-sencha-platform
Sencha	Ext JS	JavaScript Framework	30 Days	sencha.com/products/extjs
Sencha	Sencha GXT	Java Web Framework	30 Days	sencha.com/products/gxt
SensioLabs	Symfony	PHP Web Framework	Open Source	symfony.com
Stylus	Stylus	CSS Preprocessor	Open Source	stylus-lang.com
Swisscom	Swisscom Application Cloud	PaaS	Available by request	swisscom.ch/en/business/enterprise/offer/cloud-data-center-services/paas/application-cloud.html
Telerik	Kendo UI	HTML and JavaScript Framework	Available by request	telerik.com/kendo-ui-html-framework-opt
The jQuery Foundation	jQuery	JavaScript Library	Open Source	jquery.com
The jQuery Foundation	QUnit	Web Testing	Open Source	qunitjs.com
Tilde	Ember.js	JavaScript Framework	Open Source	emberjs.com/
Tsuru	Tsuru	PaaS	Open Source	tsuru.io
Twitter	Bootstrap	Web Framework	Open Source	getbootstrap.com/
Twitter	Bower	Package Manager	Open Source	bower.io
Vaadin	Vaadin	UI Framework	Open Source	vaadin.com
Verizon Digital Media Services	Edgecast CDN	CDN	Available by request	verizondigitalmedia.com/platform/edgecast-cdn
Vue.js	Vue.js	JavaScript Framework	Open Source	vuejs.org
Walmart Labs	Joi	Validation System	Open Source	github.com/hapijs/joi
Walmart Labs	Lazo	Web Framework	Open Source	github.com/lazojs/lazo
WaveMaker	WaveMaker	PaaS	30 Days	wavemaker.com
Webpack	Webpack	Package Manager	Open Source	webpack.github.io

GLOSSARY

AJAX: A method of using asynchronous data exchange between the client and server in order to create interactive websites.

API (APPLICATION PROGRAM INTERFACE): Code that allows for communication between different software programs.

ASP: Active Server Page; a feature of the Microsoft Internet Information Server and an HTML page where scripts are processed before a page is sent to the user.

ASYNCHRONOUS MODULE DEFINITIONS (AMD) API: Defines modules so that their dependencies can be loaded at different times. It is ideal in browser environments.

AUTH0: A platform that helps simplify application and API authentication.

BLACK BOX: A device whose internal workings either are not known or don't have to be understood in order to understand its input, output, and transfer characteristics.

COLDFUSION: A program used to build websites and deliver web pages that allows developers to build websites in pieces that can be stored in a database and reassembled.

COMMONJS: A project that moves JavaScript to outside of the browser.

DEPENDENCY INJECTION: A process that occurs in object-oriented programming in which a resource that a piece of code requires is supplied.

DOM (DOCUMENT OBJECT MODEL): Allows for the creation and modification of HTML and XML documents as program objects to help control who can modify the document.

HARP: A static web server written in Node.js that uses built-in preprocessing.

JADE: A feature-rich Node.js template.

JAVASCRIPT: An interpreted script language from Netscape that is used in website development for both client-side and server-side scripting. It is easier and faster to code than compiled languages, but takes longer to run.

JSON: JavaScript Object Notation; a language-independent textual data interchange format that is used in browser-based code to represent simple data structures and objects.

JSON WEB TOKENS (JWT): A method of transferring claims in a compact and URL-safe way.

MODULAR PROGRAMMING: Making a program being written more efficient, understandable, and modifiable by enforcing a certain logical structure; also known as structured programming.

MODULE: A component of software or hardware that is portable and interoperable.

MODULE BUNDLER: A tool that eases the process of using modules by bundling them with necessary dependencies.

MVC (MODEL-VIEW-CONTROLLER): A way of relating the UI to underlying data models that lets developers reuse object code and reduce the time spent developing applications with UIs.

NODE: The most basic element that is used to build data structure.

NODE.JS: An I/O framework that develops applications that are highly dependent on JavaScript on both the client-side and server-side. It is event-based, non-blocking, and asynchronous.

PHP: A script programming language and interpreter used in web development that interprets and performs operations before a page is sent to the user.

PRETTYPRINT: Stylistic formatting conventions that make text, source code, etc. easier to read and understand.

REACT: A Library in JavaScript that is used to build user interfaces.

VIRTUAL DOM: A data structure that describes how the DOM should look.