DZone

THE DZONE GUIDE TO

Web Development

Frameworks & Responsive Design

VOLUME II

BROUGHT TO YOU IN PARTNERSHIP WITH

CRAFTER SOFTWARE

Perfecto

DEAR READER,

A few years ago, I put on a blindfold and made a prediction. I looked at the future of web development and pictured a few clear winners in the space, dominating the landscape and deciding how we all do web dev. Fact is, I was fundamentally wrong. As we're approaching 2018, we still see new technologies and frameworks emerging daily, and older ones are evolving to adapt to the most recent trends. The tech industry leaders are contributing back to the community with exciting new ideas and technology almost daily, and everyone seems pretty focused on delivering the best experience possible to the end user. It's a vibrant market, and while some argue that it's lost focus and generated a lot of noise, I tend to disagree.

I once read an analogy that made a lot of sense (I am deeply sorry, but I can't remember the source). Think of a hardware store and picture the hammers section. You will find hammers of all colors, shapes, and materials. There are hammers with plastic handles and wooden handles. Then you pick whether you want your hammer to be yellow, blue, big, small, double-sided, stainless steel, forged barrel, etc., etc. There's just a world of possibilities and combinations for a tool with a seemingly simple function: hit stuff!

The truth is, every tool out there is trying to solve a specific set of problems or embrace a different methodology or work model. It is the developer's job to figure out which one is the best for a particular task, and it is DZone's job to identify and bring you the best content about them. Welcome to the second edition of the *DZone Web Development Guide: Frameworks and Tools*.

We have some exciting research findings, proving that the web is evolving fast, and web developers are quick to adopt new technologies. We cover some ground around security and show you how to ensure your web application looks and behaves awesomely on every platform. We introduce you to Kotlin for web development using the Ktor asynchronous web framework, and to Vue.js, a JavaScript library that seems to be the perfect solution when all you need is to sprinkle some dynamism on an otherwise static webpage. We show you how to use TypeScript with Redux if you want to introduce type safety on your application. And much, much more.

I hope you enjoy reading the content as much as I did, and that you find our research useful. I'll just leave a final word of appreciation to our sponsors and contributors and let you dive deep into yet another excellent guide. Happy reading.

BY HERNANI CEQUEIRA
DISTINGUISHED ENGINEER, **DZONE**

TABLE OF CONTENTS

PRODUCTION
Chris Smith
DIRECTOR OF PRODUCTION

Andre Powell
SR. PRODUCTION COORDINATOR

G. Ryan Spain
PRODUCTION PUBLICATIONS EDITOR

Ashley Slate
DESIGN DIRECTOR

Billy Davis
PRODUCTION ASSISSTANT

MARKETING
Kellet Atkinson
DIRECTOR OF MARKETING

Lauren Curatola
MARKETING SPECIALIST

Kristen Pagàn
MARKETING SPECIALIST

Natalie Iannello
MARKETING SPECIALIST

Miranda Casey
MARKETING SPECIALIST

Julian Morris
MARKETING SPECIALIST

BUSINESS
Rick Ross
CEO

Matt Schmidt
PRESIDENT

Jesse Davis
EVP

SALES
Matt O'Brian
DIRECTOR OF BUSINESS DEV.

Alex Crafts
DIRECTOR OF MAJOR ACCOUNTS

Jim Howard
SR ACCOUNT EXECUTIVE

Jim Dyer
ACCOUNT EXECUTIVE

Andrew Barker
ACCOUNT EXECUTIVE

Brian Anderson
ACCOUNT EXECUTIVE

Chris Brumfield
SALES MANAGER

Ana Jones
ACCOUNT MANAGER

Tom Martin
ACCOUNT MANAGER

EDITORIAL
Caitlin Candelmo
DIRECTOR OF CONTENT AND COMMUNITY

Matt Werner
PUBLICATIONS COORDINATOR

Michael Tharrington
CONTENT AND COMMUNITY MANAGER

Kara Phelps
CONTENT AND COMMUNITY MANAGER

Mike Gates
SR. CONTENT COORDINATOR

Sarah Davis
CONTENT COORDINATOR

Tom Smith
RESEARCH ANALYST

Jordan Baker
CONTENT COORDINATOR

Anne Marie Glen
CONTENT COORDINATOR

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 guide a great success.

Executive Summary

BY JORDAN BAKER
CONTENT COORDINATOR, DZONE

As software continues to take over the world, every organization is creating web applications to interact with their user base. Take this rapid expansion of web development, and add to it a proliferation of languages, frameworks, and tools, and you get a thriving and growing community of web developers. We asked 406 members of this community for their thoughts on the field and the suite of technologies available.

ANGULAR IS THE DOMINANT CLIENT-SIDE FRAMEWORK

DATA Angular currently sits atop the frameworks hierarchy, with 79% of respondents using it to create applications, compared to 64% last year, and 62% of respondents said they're interested in using it further or learning more about this framework. Additionally, TypeScript is one of the most favored "flavors" of JavaScript right now, with a 52% use rate among respondents, up from 32% last year. This growth among Angular users sits in stark opposition to React, which has stayed static year-over-year in terms of user growth, coming in at 30% over our last two community surveys.

IMPLICATIONS While the Google team did a great job with Angular, the main reason for its resounding success could be that it is the only true framework designed to create full web applications. Out of its main competitors, React and Backbone (though often included under the "frameworks" category) are in fact libraries, and Vue.js, while it can be used as a framework, was designed to create User Interfaces. TypeScript seems to be growing in popularity due to its being a superset of JavaScript, allowing developers to make improvements, such as type safety, to the most widely-used web development language today.

RECOMMENDATIONS Become familiar with the Angular framework, and the TypeScript iteration of the JavaScript language. Backed by Google, this framework will only continue to improve and become easier to use, and developers should take advantage.

IS CHROME EATING THE WORLD?

DATA 95% of respondents stated they actively develop for Chrome (the same number as last year), while every other major browser saw a decrease in active development: Firefox (77% - 72%), IE (51% - 40%), Safari (38%- 32%).

IMPLICATIONS Chrome's and Firefox's developer tools, such as the super handy JavaScript console, make them browsers that are really geared toward helping web developers create better applications and sites. The fact that Chrome is developed for rather than on, however, also speaks to a growing community of Chrome users outside of the development industry. While reasons for preferences are hard to quantify, the fact that Chrome is backed by one of the biggest companies in the world, with huge marketing departments, could be one reason for its success.

RECOMMENDATIONS If you've never played around with the Chrome developer tools, check them out and become familiar with the ways they can help in the development process. Mozilla also has an excellent reputation for encouraging open source development, offering an expansive set of documentation and their own developer browser in which devs can edit and write HTML, CSS, and JavaScript code. But, while Chrome and Mozilla remain great options for developers, other browsers remain in wide use outside of IT, and should thus be tested against during the development process.

A SHIFT TOWARD NOSQL DATABASES

DATA MySQL and Oracle DB both saw decreased use since last year's survey, with MySQL falling from a 61% use rate in 2016 to 57% in 2017, and Oracle DB from 38% to 32%. In opposition to this decrease, NoSQL databases MongoDB and Redis increased in popularity, from a 42% - 46% and 19% - 24% use rate, respectively.

IMPLICATIONS As more companies and organizations, even those outside the development space, create web applications to interact with users, their data is growing. This requires a more easily scalable solution, such as NoSQL databases. NoSQL is also more agile due to its more dynamic, schema-less data model, and is probably seeing a boon from the growth of Agile development processes. Redis in particular is known as a light-weight solution, which can second as a messaging queue, a helpful feature for web developers working with UI/UX.

RECOMMENDATIONS While NoSQL databases are not perfect, they are currently the best option for web developers, as they allow for agility and scalability in development. If your team is not yet using a non-relational database, testing out options such as MongoDB and Redis should be on your radar.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

DEMOGRAPHICS

406 software professionals completed DZone's 2017 Web Development survey. Respondent demographics are as follows:

- 40% of respondents identify as developers or engineers, 19% identify as developer team leads, and 15% identify as software architects.
- The average respondent has 12 years of experience as an IT professional. 60% of respondents have 10 years of experience or more; 21% have 20 years or more.
- 37% of respondents work at companies headquartered in Europe; 29% work in companies headquartered in North America.
- 18% of respondents work at organizations with more than 10,000 employees; 18% work at organizations between 1,000 and 10,000 employees; and 22% work at organizations between 100 and 1,000 employees.
- 88% develop web applications or services; 46% develop enterprise business apps; and 24% develop native mobile applications.

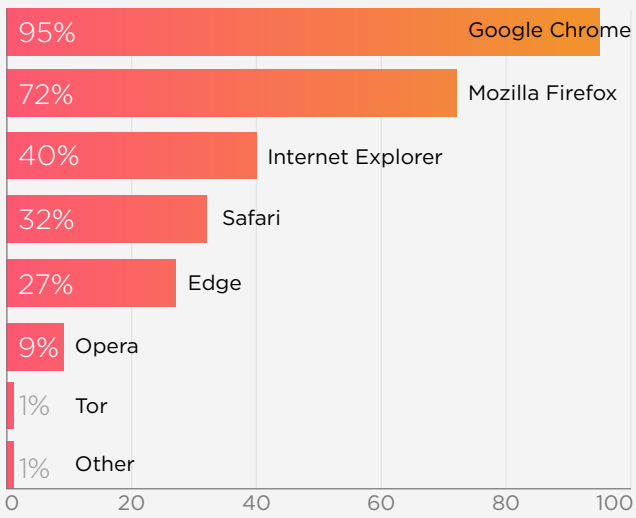
FRAMEWORKS

The Angular JavaScript framework was the most popular in our 2016 Web Development survey, and this year its popularity has only increased. While use of Angular 1 decreased slightly from 59% to 57%, use of Angular 2 increased from 33% to 46%. Overall, respondents who have used some form of Angular increased from 64% in 2016 to 79% this year. React, the next most popular framework behind the Angulars, remained unchanged year over year at 30%. Vue, on the other hand, increased dramatically from results in 2016, going from 3% to 10%. React, while its response was static from last year, is the framework respondents have the most interest in trying, with 58% of responses (up 8% from last year). Angular 2 had a minor boost in interest as well, increasing from 50% to 54% since last year's survey. On average, respondents have used 1.633 client-side frameworks for web development.

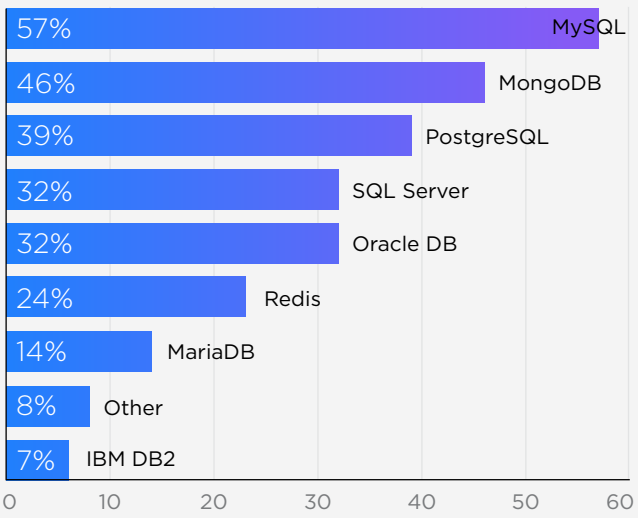
LANGUAGES

JavaScript, HTML/CSS, and Java were by far the most used languages among survey respondents. 84% of respondents

Which browsers do you actively develop for?



Which databases are you using with your web apps?



said they use JavaScript for web development, 78% said they use HTML/CSS, and 72% said they use Java. 54% of respondents said they use all three of these, and only 5% of respondents said they don't use any. Next up were PHP and Python, with 22% and 21% respectively. When asked about what languages respondents' organizations used, results were nearly identical, but PHP (29%) and Python (27%) were occasionally recognized as languages used for web development elsewhere in a respondent's organization. ASP.NET received some recognition as well, being used in 23% of respondents' companies, while only 15% of respondents said they use it themselves. Respondents said they use an average of 3.44 different languages when developing for the web, and said their organizations use an average of 3.77. For JavaScript, most respondents said they use ECMAScript (62%) or TypeScript (52%), with only 14% of respondents answering that they have used third-place CoffeeScript.

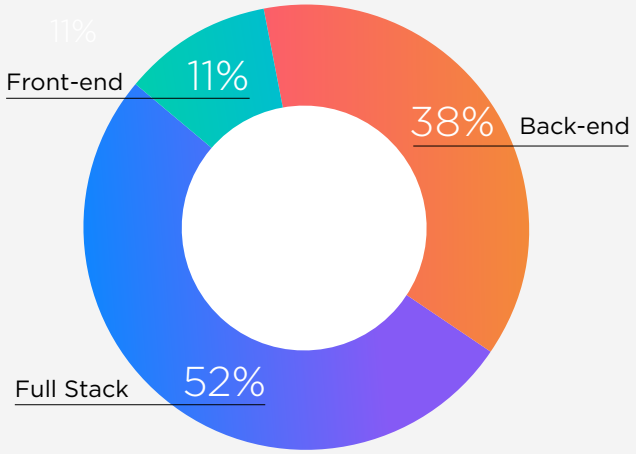
BROWSERS AND DEVICES

Google Chrome remains the browser most actively developed for—95% of respondents said they develop with Google's browser particularly in mind (compared to 96% last year). Mozilla Firefox and Internet Explorer, the next two most popular browsers to target, had more significant decreases from last year's responses, with Firefox dropping from 77% to 72%, and IE dropping from 51% to 40%. Even with fewer respondents actively developing for Internet Explorer, the increase in responses for Microsoft Edge was underwhelming, changing from 25% to 27%. For mobile devices, the most popular individual devices were the iPhone 7 (37%), the iPhone 6 and the Samsung Galaxy s7 (both 33%). 46% of respondents said they target at least one iPhone, and 42% target at least one Samsung phone; 53% said they target at least one mobile device.

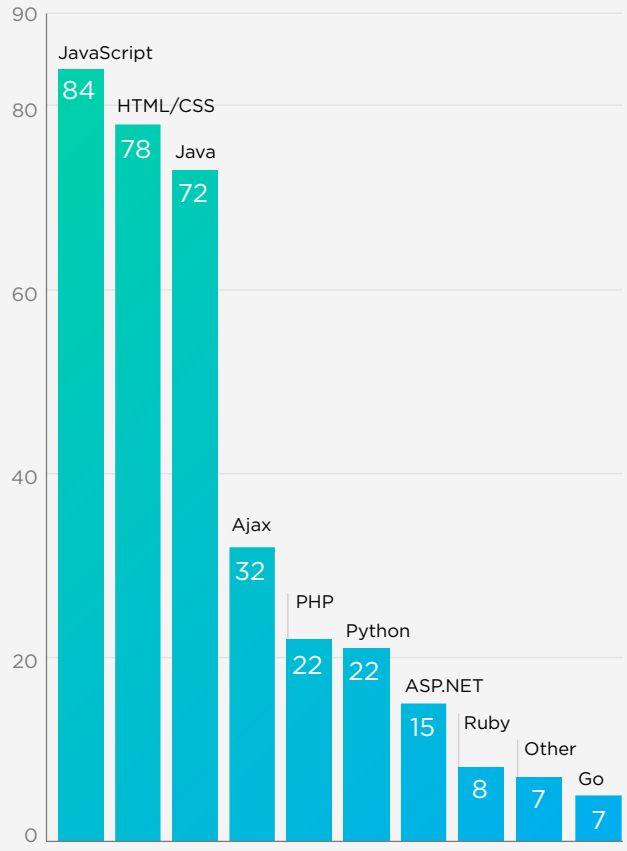
BACKEND AND BEHIND THE SCENES

The most popular databases behind web applications among respondents were MySQL (57%), MongoDB (46%), and PostgreSQL (39%). For web servers, respondents preferred Apache Tomcat (57%) and Apache Web Server (54%), with third place going to NGINX (41%). Respondents who need to push real-time/streaming data from their server overwhelmingly said they use the WebSocket API (71%) over other methods like HTTP streaming (35%) and polling (27%); HTTP streaming did gain a 9% increase from the 2016 results, while WebSocket dropped 6%. For web app testing, Selenium was heavily used (56%), followed by Jasmine (26%) and PhantomJS (25%). npm (63%) dominated over package managers Bower (29%—a 7% drop from 2016) and Webpack (28%—an 8% increase).

Do you work more in the front-end or back-end of your web application?



Which languages are you currently building for web apps?



Using TypeScript with Redux

QUICK VIEW

01 Even though Redux comes with its own annotations, adding your own can help restrict possible issues.

02 Besides preventing errors, adding detailed type annotations can help make your editor tooling more effective.

03 Establishing a consistent type model and applying it to your Redux reducers will help make multi-store setups type-safe and easy.

04 Using unions to define valid messages and annotating the `store.dispatch` function at the point of usage provides end-to-end type safety for Redux messages.

BY **ADAM BARD**
SENIOR DEVELOPER, **TAPSTREAM**

Redux is a library that provides a way to manage application state in one place. Not every application needs help with this, but for many applications, it can simplify access to this global state without causing incidental complexity by restricting the ways it can be updated and accessed.

Redux is all well and good by itself, but using TypeScript, we can add type annotations to greatly increase the type-safety of Redux-based React apps. In this article we will explore some sample annotations and develop a standard boilerplate for use with a single Redux store, and then expand this solution to cover a multi-store Redux configuration, demonstrating along the way the benefits of this added type safety when writing reducers and dispatching actions.

DUMMY-PROOFING REDUX

Let's look at Redux's "Getting Started" example:

```
import { createStore } from 'Redux'

function counter(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1
  case 'DECREMENT':
    return state - 1
  default:
    return state
  }
}

const store = createStore(counter)
```

The interesting part here is the reducer function counter. The signature of this function is critical: it accepts a state (of any shape), an action (which must have a "type" field), and returns a state of the same shape as the original. Get this wrong, and your program can break in subtle ways.

If only there was some way to enforce this signature to prevent this.

We can start by defining the shape of our state:

```
type AppState = number
```

Extremely boring stuff. Next, we'll define our action, which is a bit more interesting.

Redux considers any object with a "type" key a valid action, but we can do much better than that by restricting our actions down to only ones we expect:

```
interface IncrementAction {
  type: 'INCREMENT'
}

interface DecrementAction {
  type: 'DECREMENT'
}

type AppAction = IncrementAction | DecrementAction
```

That | is the secret sauce of this whole article, and represents one of my favorite TypeScript features: union types. You can read it as "or": The action is either an Increment or a Decrement.

Now, we need simply to apply these types to our function signature:

```
function counter(state: AppState = 0, action: AppAction):
AppState {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1
  case 'DECREMENT':
    return state - 1
  default:
    return state
  }
}
```

Besides annotating the arguments and return values, we've not had to change the function's implementation at all; TypeScript's type inference does an excellent job checking the above without requiring extra annotations. However, if we do something like misspell "INCREMENT", or forget to return the state in the default: case, the type checker will tell us right away, because we've provided it a list of the only acceptable values for action.type.

Unfortunately, we don't automatically get this benefit when calling `store.dispatch` — by default, it will (again) accept any object with a "type" key. There are two ways around this:

1) You can manually annotate store.dispatch when you call it:

```
store.dispatch<AppAction>({type: 'INCREMENT'})
```

2) You can create your own Dispatch type and apply it to store.dispatch via an intermediate variable:

```
type AppDispatch = (action: AppAction) => AppAction

const dispatch: AppDispatch = store.dispatch
dispatch({type: 'INCREMENT'})
```

This latter method is particularly useful when using the connect utility from react-Redux.

MULTI-STORE APPLICATIONS

The above is well and good, but eventually most applications will find themselves separating AppState and the reducer into multiple reducers. Each of these reducers will have to provide:

- Its own state
- Its own actions
- Its own reducer function

I also like to include a fourth item: a function returning a blank state. This will come in handy frequently.

Let's move our counter reducer into its own file (I usually put these all together in a reducers/directory). (Let's also change the state to store the count as a key, just to make it more typical of a real-life Redux state object).

One more detail: to be correctly typed, the reducer function should accept any action, not just the ones from its reducer. So, we'll need to import `AppAction` from the store. Luckily, TypeScript can deal with this sort of partial circular import.

```
// reducers/counter.ts
import { AppAction } from '../store'

export interface State {
  count: number
}

export const blankState = () => ({
  count: 0
})

interface IncrementAction {
  type: 'INCREMENT'
}

interface DecrementAction {
  type: 'DECREMENT'
}

export type Action = IncrementAction | DecrementAction

export function reducer(state: State = blankState(),
action: AppAction): State {
  switch (action.type) {
  case 'INCREMENT':
    return { count: state.count + 1}
  case 'DECREMENT':
    return { count: state.count - 1}
  default:
    return state
  }
}
Now, in store.ts, we'll connect this sub-reducer to our
global store:
import { createStore, combineReducers } from 'Redux'
import * as counter from './reducers/counter'

export type AppState = {
  counter: counter.State
}

export type AppAction = counter.Action

export type AppDispatch = (action: AppAction) => AppAction

const blankAppState = (): AppState => ({
  counter: counter.blankState()
})

const store = createStore(
  combineReducers({
    counter: counter.reducer
  }),
  blankAppState()
)
```

From this, it should be clear how we might extend our store with additional reducers. Imagine that we wrote a "todos" reducer as well. After connecting it to our store, our store.ts file might look like:

Redux is all well and good by itself, but using TypeScript, we can add type annotations to greatly increase the type-safety of Redux-based React apps.

```
import { createStore, combineReducers } from 'Redux'
import * as counter from './reducers/counter'
import * as todos from './reducers/todos'

type AppState = {
  counter: counter.State,
  todos: todos.State
}

type AppAction = counter.Action | todos.Action

const blankAppState = (): AppState => ({
  counter: counter.blankState(),
  todos: todos.blankState()
})

const store = createStore(
  combineReducers({
    counter: counter.reducer,
    todos: todos.reducer
  }),
  blankAppState()
)
```

So, what have we gained?

Firstly, our reducers are now strongly typed, right down to their accepted actions — no sign of an any type anywhere, nor is it possible to forget to return the state.

In addition, dispatch functions will only accept valid actions (provided that they have been annotated as described above). I find that this obviates the need for a separate action-generating function as many Redux guides recommend. We could, of course, provide such a function:

```
const incrementAction = (): IncrementAction => ({
  type: 'INCREMENT'
}
```

...but, since actions are type-checked for validity anyhow thanks to our efforts, it becomes little extra effort to construct the action at the time that you dispatch it. Your editor will catch any errors as you code!

TYPING REACT-REDUX

With our state and dispatch types, we can make a React integration that much more type-safe as well.

When writing a component with providers — as we do when we use react-Redux's connect — I like to split its props into parts based on who provides them.

```
import * as React from 'react'

interface StateProps {
  count: number
}

interface DispatchProps {
  increment: () => void
  decrement: () => void
}

class MyComponent extends React.PureComponent<StateProps &
DispatchProps, {}> {
  // ...
}
This will let us annotate the functions that we'll be
providing to connect:
import { connect } from 'react-Redux'

import { AppState, AppDispatch } from './store'

const MyComponentWrapper = connect(
  (state: AppState): StateProps => ({
    count: state.counter.count
  }),
  (dispatch: AppDispatch): DispatchProps => ({
    increment: () => dispatch({type: 'INCREMENT'}),
    decrement: () => dispatch({type: 'DECREMENT'}),
  })
)(MyComponent)
```

Let's dissect the above:

- The state provider is annotated to accept AppState and return StateProps, giving us that extra type-checking.
- The dispatch provider is similarly annotated to accept AppDispatch and return DispatchProps. Using AppDispatch ensures that our calls to dispatch can only be made with valid messages — if we misspell a type or miss an argument, the compiler (or our editor tooling) will catch it and tell us.

SUMMARY

Although Redux comes with its own types for the above, they lean heavily on TypeScript's any type. By providing our own implementations of the State, Action, and Dispatch types we can gain vastly increased type safety.

ADAM BARD is a software developer and author from Victoria, BC, Canada, with decades of experience making websites (and other applications). He presently develops for Tapstream, freelances from adambard.com, and is currently working on laterforreddit.com, among others. You can find him on Twitter @adambard.

Implementing Effective User Password Management: Do's and Don'ts

BY GABRIELE TOMASSETTI
FREELANCE SOFTWARE ENGINEER

MINIMUM LENGTH REQUIREMENT...

8 characters. Longer passwords are generally better, but forcing your users to have a password longer than 8 characters is dangerous. If it is too long users will bypass them with repeatable patterns like *toolongtoolong*. Furthermore, most users will choose a password of the minimum length, making easier to crack them.

...AND A MAXIMUM LENGTH.

...Is at least 64 characters. You should allow users to use long passwords, if they want. The NIST suggests a length of at least 64 characters. Also, be clear about it, do not silently truncate passwords that are longer than your limit.

DO NOT PUT RESTRICTIONS ON CHARACTERS CLASSES.

You should not require passwords to have different classes (e.g., an uppercase letter, a digit, etc.). People use predictable substitution rules and patterns such as *P@ssw0rD*. However, you should also not prohibit their use.

BLOCK PASSWORD REUSE AND COMMON PASSWORDS.

Attackers will try to use passwords exposed in data breaches with their associated emails. Basically, if a breach contains an account with email@example.com and password *shd30a@s#89*, they will test whether that combination works on your service. The simplest way to check exposed passwords is with the Pwned Password API of Have I Been Pwned.

ALLOW USERS TO PASTE PASSWORDS.

Even in the *confirm password* field. Some developers block pasting based on the idea that practicing typing a password facilitates remembering it. It does not work and it is annoying to people that use password managers.

BE CAREFUL WHEN ALLOWING TO BYPASS A PASSWORD.

You must provide a method to bypass a password, in the case a user forgets their own password. The issue is that attackers can always take advantage of it. So, if you have a weak alternative to a password, you are always exposed. For example, avoid security questions, instead use another authenticated channel, like an email address or a mobile phone.

USE GRADUAL AUTHENTICATION.

There is no foolproof way to protect against an attacker that has the right password, but is effectively the wrong person. However, you can contain the damage by limiting the actions of a user identified by an anomalous authentication. An anomalous authentication could be one coming from a different country than usual. A limitation could be forcing the user to pass a CAPTCHA. You should also warn the user through an authenticated channel (e.g. email) in the case of an anomalous authentication.

ENCOURAGE TWO-FACTOR AUTHENTICATION.

Two-factor authentication is effective in protecting user accounts. It is also considered cumbersome by many and that is why it is not popular. A middle ground could be to encourage the use of two-factor authentication only when doing important actions, such as buying an item or publishing an article.

33

IS THE ANSWER.

OR IS IT?

Thirty-three is the number of web platforms required to achieve 80% market coverage.

But if you consider previous, current and beta browsers in a responsive web environment and factor in the number of possible OS versions maybe that's not enough? Read on to find out!

Perfecto

perfectomobile.com/is33theanswer

Why is 33 the Answer?

The strategy pillars for testing responsive websites are clear.

1. Automate (Duh)
2. Use the open source Selenium framework, or one based on it
3. Use traffic analytics to determine appropriate lab size
4. Extend functional test automation to include visual verification across break points
5. Embrace TDD and then progressively test every build, every merge, every night
6. Deliver fast feedback

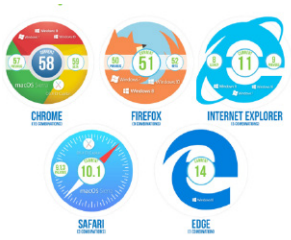
While much has been written about the need to automate and the de facto automation framework of choice — Selenium — more work is needed to define guidelines on building the right strategy.

Most cite the key advantages of a single code base and consistent user experience across platforms. These advantages also create a multiplier effect on testing:

1. Platform matrix (web and mobile)
2. Visual validation across CSS break points, and
3. Differing web and mobile navigation options

Selenium is well suited to address most of these challenges – high volume parallel execution. Most teams look for extensions to visual validation and a new strategy for enabling efficient analysis of test results able to connect to the defect management tool of choice. The Achilles heel undermining confidence is a reliable lab offering supporting real mobile devices and web browsers matched to user traffic.

So why is 33 the answer? Thirty-three is the number of web platforms required to achieve 80% market coverage. But think about applying a "current, beta, last" approach across five browser platforms, and don't forget about mobile devices. So, full coverage requires adding in thirty-two mobile devices. Turns out the answer isn't 33, it is 55!



WRITTEN BY CARLO CADET
DIRECTOR OF PRODUCT MARKETING, **PERFECTO**

PARTNER SPOTLIGHT

Continuous Quality Lab

Perfecto

Beauty Retailer Focuses on FLAWLESS User Experiences for Doubling EComm Revenue

CATEGORY
Automated Testing Platform

NEW RELEASES
Every three weeks

OPEN SOURCE
No

CASE STUDY

One leading beauty retailer is revamping their strategy to double eCommerce revenue by focusing on delivering flawless experiences.

Software development retooled their app quality strategy to accelerate delivering new features faster. The team partnered with Perfecto to accelerate development velocity without more budget.

The team grew test automation by 50%. By implementing more automation and parallel execution, regression duration dropped from 160 hours to less than 12 hours, a 94% acceleration.

Next up was scaling browser coverage. Testing coverage went from the latest version of two browsers using one Windows OS to supporting all five-leading browsers running the current, beta, and one prior version, now totaling 36 browsers, a 17x improvement.

This acceleration enabled shorter sprints and shifted testing earlier. Regression testing was previously executed once, maybe twice per sprint. Today, the team is receiving faster feedback with smoke tests executed every build and full nightly regression runs.

Partnering with Perfecto resulted in a faster time-to-market and better quality at the same budget.

The team is well on its way towards delivering flawless user experiences.

STRENGTHS

- **Automate More**: The Forrester Wave™ Mobile Front-End Test Automation Tools LEADER
- **Optimize DevOps pipelines** by enabling Continuous Testing of web, mobile, and IoT apps
- **One cloud-based lab** to test every client, embedded within the delivery toolchain
- **Analyze fast, fix fast:** provide developers the artifacts to reproduce and trouble shoot issues
- **Continuous testing in production:** Proactively monitor the real customer experience

NOTABLE CUSTOMERS

- Elevate
- R&V
- Verizon
- Paychex
- ING
- Kaiser Permanente

WEBSITE perfectomobile.com

TWITTER @perfectomobile

BLOG blog.perfectomobile.com

A View to Vue.js

QUICK VIEW

01 Vue is an easy to use, lightweight framework for working with JavaScript.

02 Vue provides a simple and intuitive syntax for data binding and dynamic expressions.

03 While easy to use, it scales up from simple use cases to more complex single page applications.

BY **RAYMOND CAMDEN**
DEVELOPER ADVOCATE, **IBM**

If you're like me, you've likely been bombarded over the past few years by Angular and React content. Every other blog entry, conference session, or keynote seemed to be focused on either of the two. With that context in mind, I can still remember the first time I saw a "Vue.js" talk listed on a conference schedule. While I certainly do not pretend to know everything, at least when it comes to buzzwords and the such, I was pretty sure I had heard of most of the popular front-end frameworks out there and this was one I had decidedly not heard of.

Vue.js is not new. It was first released in 2014 by its creator, Evan You. But it's almost as if talk about the framework has been drowned out by all the attention to Angular, and even more so, React. If you've avoided looking at Vue.js because of this, you've made a mistake, and I'd like to explain why.

SO WHAT IS IT?

Vue is an incredibly simple framework that can scale up with your needs. If you aren't building an app but instead simply need a bit of JavaScript on a page, Vue is the perfect tool to make that JavaScript easier to manage and more powerful. I like Angular, but sometimes I don't want an "app" — I don't want a large set of dependencies and a build "process" just for one simple page. While Vue projects can certainly grow into complex applications, the default is to start small and simple and I really appreciate that.

At the basic level, Vue handles two-way binding between your view (form fields and the such) and your code as well as the rendering of data. In the abstract, that's almost too vague to make sense of, so let's look at a simple example.

First, a bit of HTML. (The HTML has been simplified to save space, including removing the script tag to load Vue itself.)

```
<div id="app">
  <input v-model="name"><br/>
  My name is {{name}}
</div>
```

And now the JavaScript:

```
let app = new Vue({
  el:'#app',
  data:{
    name:'Ray'
  }
})
```

Let's tackle the JavaScript first. It begins by creating a new instance of Vue. You can basically think of this as a Vue app, but rather simplified. I bind it to the DOM with the el value and then specify some default data. In this case a name with the value of Ray.

On the HTML side, note that I've used a div with "app" as way of connecting it to the Vue instance made earlier. There are two special things going on here. The attribute v-model in the input tag binds the value in the form to the variable name defined in the Vue instance. Changing the input field will update the data and vice versa. The next line shows a simple example of the template syntax. In this case, `{{name}}` is automatically replaced with the value of name.

Run this code (which you can do via the CodePen here and you'll see that as you type, the text after "My name is" is updated automatically.

TAKING IT TO THE NEXT LEVEL...

Ok, that's possibly not that exciting, but let's kick it up a notch. Let's build a simple application that searches the iTunes API. It

will have a search field, a button to initiate searching, and then a place to render the results. We'll start with the HTML:

```
<div id="app">
  <input v-model="term" type="search">
  <button @click="search">Search</button>
  </p>
  <div v-for="result in results">
    <img :src="result.artworkUrl100">
    <b>Artist:</b> {{result.artistName}}<br/>
    <b>Track:</b> {{result.trackName}}<br/>
    <br clear="left">
  </div>
</div>
```

The first input field uses a similar syntax to the first example, using v-model to tell Vue, "Connect this value to a variable, 'term'." The button tag uses a shortcut "@click" to assign a click handler to the button. You'll see the code assigned to that in a moment.

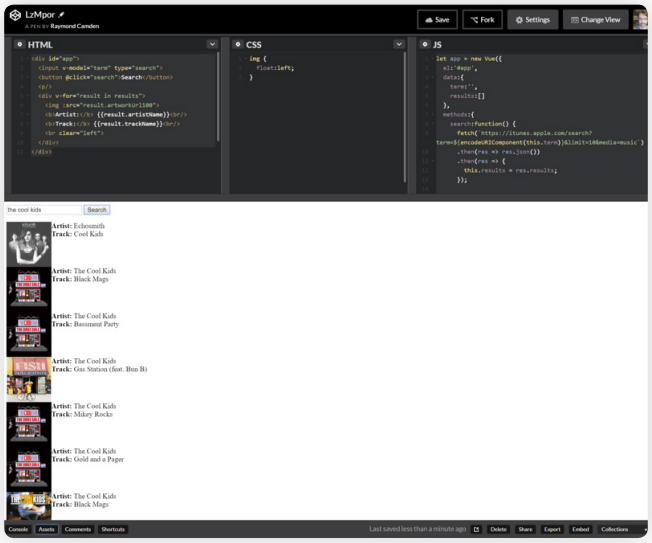
The fun part is the result render. By using `v-for` in the div, we've said, "Repeat this div for every item in the results array and assign a variable, result, to the individual item." Inside the div, we use `{{ … }}` tokens to render the data. The image uses another shorthand, `:src`, to assign the src value to the result. Now let's switch to the code.

```
let app = new Vue({
  el:'#app',
  data:{
    term:'',
    results:[]
  },
  methods:{
    search:function() {
      fetch(`https://itunes.apple.com/
      search?term=${encodeURIComponent(this.
      term)}&limit=10&media=music`)
      .then(res => res.json())
      .then(res => {
          this.results = res.results;
      });

    }
  }
})
```

It starts off much the same as the last example, but note the new data item: results. We set the initial result to an empty array. Next, we've defined a new block, methods, that will store functions related to our app. In this case, a function called search. It then uses the Fetch API (not part of Vue, it's a modern XHR replacement) to hit the iTunes API. The result is in JSON, which is parsed and then simply assigned back to the array defined earlier.

And that's it. No creating long strings of HTML to inject into the HTML, no event handlers defined by hand. Vue basically made shortcuts for all of the things we commonly do when working with JavaScript and HTML and made it as easy as pie. You can run this online yourself here. Here's an example of it running:



Of course, there's a lot more to the template rendering syntax, conditionals for example. The general form would look like this:

```
<div v-if="somethingIsCool">Cool.</div>
```

Another way to conditionally add something to DOM is via CSS binding:

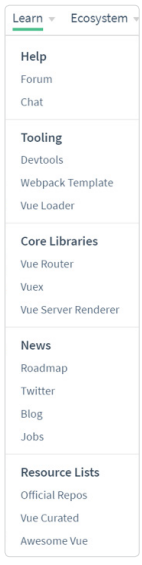
```
<div v-bind:class="{ coolItem:somethingIsCool }"></div>
```

In this example, when the value of somethingIsCool is truthy, the coolItem is added as a class value for the div.

Again, the idea here is to provide simple shortcuts for common use cases. To me, it feels like I've got the power of Angular with nearly zero overhead.

WHERE NEXT?

Obviously, the first place you'll want to start is the Vue homepage. Once there, start looking at the Guide as it walks you through the basics into the more complex tutorials. Then follow up with this incredible five-part guide to Vue by Sarah Drasner. Chrome and Firefox uses can also find a devtools extension to help development. Mouse over that "Ecosystem" dropdown on the Vue page for a good list of other related resources as well (see right).



My suggestion is to simply give it a whirl. At the end of the day, actually using a framework tells you pretty quickly if it's something that's going to work well for you. The primary reason I don't use React is that it simply didn't gel with me — it certainly didn't lack for features. However, Vue.js did, and I think it might with you too!

RAYMOND CAMDEN is a developer advocate for IBM. His work focuses on LoopBack, serverless, hybrid mobile development, Node.js, HTML5, and web standards in general. He's a published author and presents at conferences and user groups on a variety of topics. Raymond can be reached at his blog, on Twitter, or via email.

5 Dev Practices to Ensure Stellar Website Introduction

QUICK VIEW

01 Website development now has to accommodate various digital channels, forcing teams to shift more activities including testing left.

02 Agile and DevOps maturity requires adoption of new tools and practices (dealing with Big Data, ability to debug quickly, BDD, and ATDD).

03 Coverage optimization for digital platforms is an ongoing challenge that requires strategic thinking.

BY **AMIR ROZENBERG**
DIRECTOR OF PRODUCT MANAGEMENT, **PERFECTO MOBILE**

People have been creating websites for the longest time. Some may say "HTML is HTML". Well, not exactly: websites are now rendered on many different types of devices, browsers, form factors etc. New technologies enable HTML5 to offer content and services according to the user's context (location, time etc.). At the same time, development cycles accelerate and sprints are shrinking as organizations adopt agile practices. This article will offer best practices to improve team efficiency and deliver high-quality products, while maintaining velocity and competitive edge.

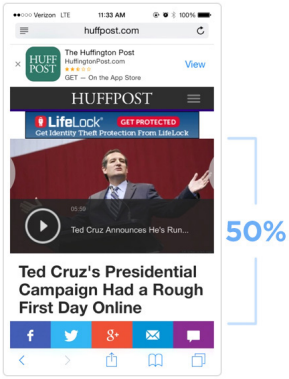


***Figure 1**: Suboptimal rendering on a mobile screen*

EMBRACE QUALITY ACTIVITIES (AS PART OF DEVELOPMENT)

We're increasingly observing the fallacy of the separation between dev and quality activities. Defects are often found late, anytime between mid-sprint and production. Developers then spend expensive hours undoing code to correct the issue, creating more QA work, increasing risk, and eventually jeopardizing the app's success.

The way to address this expensive cycle is to expand the "definition of done" to include some level of code verification. In other words, developers need to run a few tests pre-commit and post-commit to get fast feedback. This is the best, cheapest, and safest time to find a defect: the code is still fresh in the developers' head, it won't take long to fix, and there's no added risk by undoing code that was built on top of this recent code. A high-quality website is the end goal of the entire team, regardless if you are part of the dev, design, test, or ops team.

It's not to say traditional QA activities would all move into the hands of developers. Load testing, for example, may not be suitable for a development environment, but the more agile the quality activities can realistically be, the better.

To make this change, a few things need to happen:

- Small scale test activities need to be automated and provide rapid value: if a developer committed code, some tests need to be run to provide fast feedback. That process needs to be automated.
- Developers can adopt developer-oriented tests and tools: for example, unit tests with XCTest/Espresso, getting network performance insight using HAR files, or conducting single-user performance tests.
- Developers need to learn and see the value delivered, otherwise they won't do it.
- Taking advantage of the test automation pyramid and implementing more API tests at the expense of UI tests is a way to reduce testing time while practicing agile testing. Another modern approach for web dev and testing is to leverage headless browsers for unit testing (i.e. Google's puppeteer project).
- Don't underestimate the importance of accessibility for your website. This is becoming a big deal for many reasons,

including site SEO, adoption, and compliance. Testing for site accessibility is quite hard to automate on mobile devices compared to the more mature web testing tools. However, the complexity of doing so shouldn't be an excuse not to do it at all.

FOCUS VIGILENTLY

Assuming we agree that developers pick some quality activities, the question becomes, "What's relevant and vital for developers to work on?" Their time is expensive. One area where they should not be working is in a lab (maintaining emulators, browsers, or devices). The effort to maintain a lab can become very time-consuming very quickly: the lab needs updates, care, and attention. A proper lab should provide all the means for coverage, automation, fast reporting, etc. The developer should only need to describe the user flows that are important to test (matching the code they are developing now) and which devices and browsers to test those on. Everything else should be automated.

FAIL FAST, FAIL EARLY

In the spirit of agility, developers will realize that testing (and failing) early beats failing later. As mentioned earlier, it does require investment: (stable) tests need to be written alongside the code, and it will take some time for those to run and create a perspective for the quality of the code committed. However, the investment is well worth it. Knowing within an hour (or less) about a new defect is much better than knowing about it weeks or months later. Even iterating a few times, failing until the right solution is found, is better.

ADOPT COVERAGE APPROACH (INCLUDE MOBILE) AND PRIORITIZE

One of the things to remember is that the website will most likely be rendered on all kinds of devices, browsers, and form factors. The common approach to handle all of these variations is to adopt a responsive web design approach: one line of code serves many screens, but that's not easy either. With the same code, one needs to offer the right rendering for different device combinations and manage network connectivity to optimize the experience for small and large screens.

The same thinking should be applied to quality activities: they need to consider a prioritized set of browsers, devices, user flows, and environments in which the user is likely to experience. The lab should support all of these so the developer can very easily create test scripts to exercise the code in as many scenarios as possible. The top browsers developers should currently consider in their testing are listed in the table on the top right.

SHIFT LEFT, SHIFT RIGHT: QUALITY ACROSS THE LIFECYCLE

Last but not least is the realization that app quality is something that needs to be a point of focus throughout the life stages of the app: from design to production. Truly agile dev teams are driving

ESSENTIAL WEB FAMILY	ENHANCED WEB FAMILY	EXTENDED WEB FAMILY
Chrome Latest/Win 10	Chrome Beta/Win 7	Chrome Latest/Win 8.1
Chrome Latest/Mac OS Latest	Chrome -1/Win 7	Chrome Beta/Win 8.1
Firefox Latest/Win 10	Firefox Beta/Win 10	Chrome -1/Win 8.1
IE 11/Win 10	Firefox Latest/Win 7	Firefox Latest/Win 8.1
Safari Latest/MacOS Latest	Firefox -1/Win 10	Firefox -1/Win 8.1
Edge Latest/Win 10	Safari Latest/MacOS -1	Firefox Beta/Win 8.1
Chrome Beta/Win 10	IE 11/Win 7	Safari -1/MacOS Latest
Chrome Beta/Win 7	IE 1/Win 8.1	Safari -1/MacOS -1
Firefox - 1/Win 7	IE 9/Win 7	Firefox Beta/MacOS Latest
Chrome -1/Win 10	Chrome -1/MacOS -1	Chrome Beta/MacOS Latest
	Chrome Latest/MacOS -1	IE 8/WIN 7
	Firefox Beta/WIN 7	
Total 10 Browser Permutations	**Total 12 Browser Permutations**	**Total 11 Browser Permutations**

the application behavior in production. There are several reasons/motivations to do so, to name a few:

- With the proliferation of client-side capabilities, the client side has an increasingly bigger role in real time incidents, and developers are being called to assist. If they are being called anyway, they might as well own the experience and fix incidents quickly.
- It's easy to do: with a proper test automation capability, it is very easy to apply the same or similar test framework to assets in production.
- It's valuable: Not only do coders know (and take pride) or their code in production, but they can also benchmark the new build performance against production and improve accordingly.

Modern websites are developed and displayed very differently compared to ones developed years ago. Technology (such as HTML5) and processes (agile) present new opportunities to deliver new functionality earlier and at higher quality and functionality to the market. With that comes the challenge of reducing defects in production, which risks the end user experience. With proper measures, tools, and perspective, developers can extend their area of responsibility to embed quality activities into their daily activities in a way that not only doesn't slow them down, but allows them to develop faster, find defects early, and fix them without undoing code and design. For more advanced reading, please refer to Earn Kinsbruner's ebook, *10 Test Automation Frameworks for Cross-Browser Testing – A Comparison Guide for 2017*.

AMIR ROZENBERG is the director of product management, responsible for core product strategy at Perfecto Mobile. Amir has extensive experience in the digital industry with expertise in areas including application development, testing, delivery, and monitoring. Prior to working at Perfecto Mobile, Amir led the mobile monitoring practice at Gomez/Compuware, led the founding of Adva Mobile, and held leadership positions at Groove Mobile, Nextcode Corp. etc, Amir commonly writes for DZone, CIO Magazine, and other publications.

The Flavors of JavaScript

Building web applications used to be relatively simple. You had HTML and CSS — maybe you used Python or vanilla JavaScript — and probably had MySQL and Apache HTTP on the backend. Now, however, you not only have a plethora of databases, hosting providers, and servers, but an enormous collection of JavaScript libraries and frameworks. Add to that the choice of testing tools, CDNs, build script tools...the sheer number of options can be confusing. To sort it out, we surveyed our audience to see what they used the most, and what use cases they were best suited for. Take a look at the menu and make your own perfect JavaScript sundae.

Recommended Combinations

MEAN STACK (MongoDB, ExpressJS, Angular, node.js)

LAMP STACK (Linux, Apache, MySQL, Python)

JAVA + SPRING

Package Management Tools

NPM
With 63% of members using it, npm is the most popular package management tool. It is also the default manager for node.js.

CDNs

JQUERY CDN
Part of the jQuery library is the CDN hosted by Stackpath. Given jQuery's popularity, 50% of survey respondents use it.

GOOGLE HOSTED LIBRARIES
A collection of CDN libraries that can be added to several web frameworks. About 35% of DZone members use them.

Build Script Tools

NPM SCRIPTS
Even though it's not built to be a build tool, npm can act as one thanks to npm Scripts,which are used by 55% of respondents.

GULP.JS
Gulp automates several repetitive web development tasks such as concatenation, and unit testing. 31% of DZone users automate tasks using Gulp.

Frameworks

ANGULAR
Angular has evolved from its origins as just a framework to a full platform for building HTML or JavaScript web apps. It includes templates, dependency injection, end-to-end tooling, and more. A whopping 79% of respondents have used some version of Angular.

BACKBONE.JS
Backbone was designed for single-page application. It has a reputation for being lightweight, and it's only dependent on the underscore.js library. 10% of developers use Backbone.js.

REACT
React was built by Facebook to build large web apps that can change over time or in response to a user's action without reloading the page. 30% of DZone members have used React.

VUE
Vue was specifically designed to build user interfaces, and is easy to integrate into projects with other JavaScript frameworks. 10% of survey respondents use Vue.js.

Testing Tools

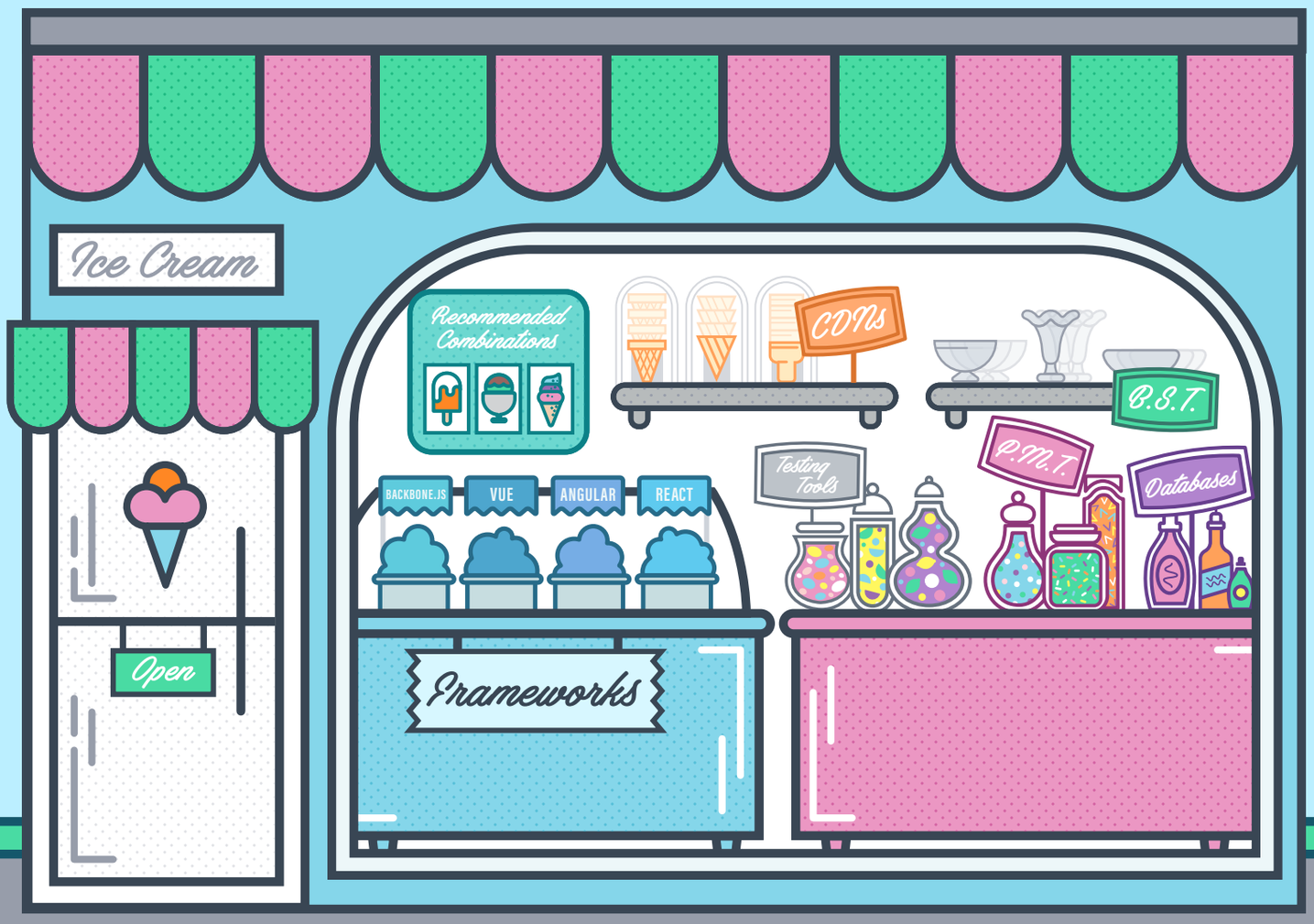
JASMINE
a lightweight behavior-driven development framework for testing JavaScript. 26% of DZone members use it.

SELENIUM
a tool for automating web browsers for the purposes of automated testing. 56% of respondents use it.

Databases

MYSQL
an open source relational database, used by 57% of respondents. It's a core piece of the LAMP stack and other –AMP stacks.

MONGODB
is a NoSQL document database that stores data in JSON, BSON, or XML files, used by 46% of DZone members.



Defenses vs. Injection Attacks

QUICK VIEW

01 Injection attacks are one of the most common ways to exploit systems.

02 Not enough developers understand what the common defenses against these attacks actually do.

03 Learn how filtering, synchronizer tokens, double submit cookies, and encrypted tokens are all used to stop XSS and CSRF attacks.

BY **CHRIS LAMB**
CYBER-SECURITY RESEARCH SCIENTIST, **SANDIA NATIONAL LABORATORIES**

Injection attacks are remarkably common. In fact, they're the backbone of most exploits; after all, in order to compromise a system, an attacker needs to inject something into the system that executes. We see this same pattern in remote code execution exploits, SQL injection, cross-site scripting, request forgery attacks, and so on. Most modern development platforms support defenses against these kinds of techniques out of the box simply by setting a few configuration parameters. And, the vast majority of developers leave it at that without really understanding what those defenses do.

Well, we're going to fix that today.

INPUT FILTERING AND CONTENT MARKING

We can look at these kinds of defenses as falling into two distinct categories: input filtering or content marking. Content filtering involves, well, filtering content. We've been using these approaches since, roughly, the beginning of time. Essentially, this involves scanning all submitted content, and ensuring that the content contains no potentially harmful strings, characters, symbols, or other input. We've used this technique to scan submitted data in web applications for SQL characters, for example, as well as for submitted JavaScript syntax. The idea is that we can capture submitted SQL or JavaScript, and by capturing those symbols, we can stop them from being stored and potentially executed. Content marking takes a different approach. Here, we somehow flag all known good content and then check for that marker prior to execution. This is a common cross-site request forgery defense. Synchronizer tokens are a common defense against this type of attack today, in many different application delivery platforms.

These approaches are common, but still have significant problems. Content filtering is difficult to implement well, and leads to both false positive and negative issues. Granted, many of these implementations today are well designed and effective, but they're not free. Implementing the correct filters takes time, thought, and experience; and processing every submitted input string has a significant computational cost, especially at scale. Tagging approaches fortunately tend to scale and can be implemented at the application server level, but they also lead to more complexity, require additional processing, and can be difficult to implement.

With that in mind, let's take a look at the different types of injection attacks and their corresponding attack models to understand how to best craft a defense for each approach.

SQL INJECTION AND XSS DEFENSES

Today, XSS and SQL Injection attacks are highly focused on filtering. This makes a certain amount of sense—both attack models are the same, though the targets differ.

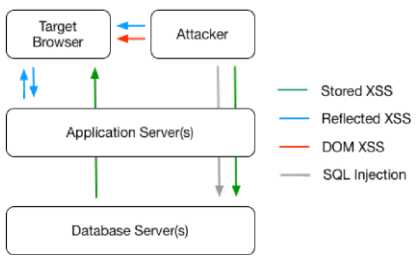


***Figure 1**: shows the attack models of various XSS and SQL injection attacks.*

Figure 1 shows the attack models of various XSS and SQL Injection attacks. Stored XSS attacks involve injecting content through the application server into the underlying data store, where the injected script is eventually sent to the target browser. Reflected XSS attacks are usually instigated via some external program, like an email client, chat program, or SMS client that coerces the user to click on a specially-crafted URL with a script payload, which is reflected from an application server back to the client with the trust properties of the reflecting server. DOM XSS attacks inject a script payload into the browser DOM, where it is extracted and executed by the browser. Finally, SQL Injection attacks send specially-crafted data to the application server, which then forwards that data to the underlying data store where that data is interpreted as valid SQL directives. Note that the first stages of stored XSS and SQL Injection attacks are the same - injecting content into the data store.

All mainstream defenses against these attacks use filtering in one form or another, but in very particular orders. OWASP goes into detail with respect to what kind of filtering happens in what order to defend against which types of attacks—suffice to say that it's complex, and you see developers messing the order up all the time. (Ever see "&" in a dynamic section of a web site? This is why.) Escaping content is straightforward, even if the ways you need to combine the escaping isn't, but it ends up being pretty obtuse. CSRF defenses are much more interesting and more difficult to understand.

CSRF DEFENSES

First, in order for any of these defenses to work, you can't have any XSS vulnerabilities. XSS isn't required to make CSRF work, but it does give attackers the ability to circumvent every CSRF defense. The simplest way to detect CSRF attacks is to just confirm that the target and origin URLs are the same. If they are, there's no cross-site activity—nothing to see here, go ahead and fulfill the request. We do this by checking the origin or referrer HTTP header values (or both) and comparing them to the target HTTP header value. While this approach can give you insight into the validity of a request, these headers frequently don't exist and are modifiable in a variety of ways. This isn't really sufficient to defend against CSRF, and may not even be necessary, as it's so easy to alter.

The other three CSRF specific defenses include synchronizer tokens, double submit cookies, and encrypted tokens.

Synchronizer tokens are randomly generated values that an application server inserts into sensitive elements as a hidden field or within a URL (think forms, images, and any other tag that will initiate an HTTP request). The server will then extract this token and compare it to the token saved in the user session, rejecting the request if the two don't match. Usually, this is maintained for an entire user session, though it can be implemented at the request level as well (though this approach leads to problems, like using the back button or bookmarking a site).

Furthermore, including the token in the URL can lead to token leaking. URLs are frequently saved in logs and browser histories, where they can be accessed by attackers and used to circumvent CSRF protections.

Double submit cookies take advantage of the fact that attackers can't access cookies when crafting CSRF requests due to same-origin policies baked into browsers. Basically, a server submits a random value as a cookie separate from the session ID cookie. The client then inserts this random value into a hidden field (or similar) where the random value can be validated by the receiving server. This way, the server need not maintain any state; the cookie and the hidden field are delivered by the client on resource requests, and the server has access to both pieces of data, allowing it to validate that they are in fact both equal. Keep in mind though, cookies aren't always available, and with cookies you can run into subdomain issues as well.

Encrypted tokens are the final CSRF defense in use today. This is similar to the double submit cookie approach, in that some combination of a user ID, a timestamp, and a nonce are inserted into a data structure that is then encrypted by the server, using a stored key. The token is submitted to the client in a hidden field in generated HTML. The client then uses this encrypted value with any HTTP requests sent to the server, including the value in either HTTP attributes or hidden fields. When the sever receives this token, it will decrypt the token with its stored key and compare the extracted values, using the timestamp to protect against replay attacks.

This is the state of the art today with respect to injection attack defenses—either SQL Injection, XSS, or CSRF attacks. When looking at them in order like this, the evolution of synchronizer tokens into double submit cookies, which then branched out into encrypted tokens, becomes clearer. Note that none of these defenses are perfect, some have significant scalability problems, and standard browser use cases (e.g. saving page or using the back button) can cause the defenses to fail and deny service to legitimate users. Nevertheless, you need to protect against these kinds of attacks, and these are the best ways to do it today.

DR. CHRIS LAMB currently serves as a cyber-security research scientist with Sandia National Laboratories. He is also a Research Assistant Professor affiliated with the Electrical and Computer Engineering department at the University of New Mexico. Currently, his research interests center around industrial control system cybersecurity, machine learning, artificial intelligence, and their intersections. He is a TOGAF 9 Certified Enterprise Architect and a Certified Information Systems Security Professional (CISSP) through the International Information Systems Security Certification Consortium.


CRAFTER
SOFTWARE
Content Management
Reinvented for Developers
Finally, an Open Source CMS that is:
Amazing for Developers
Easy for Editors
Simple for Operations
The first and only enterprise CMS based on Git, enabling content and code to flow seamlessly throughout development and deployment cycles.
READ E-BOOK
DOWNLOAD
CRAFTER CMS
craftersoftware.com

CMS Reinvented for Modern Web Dev

Modern web apps frequently contain a lot of content, including rich digital media. The most engaging apps keep their content and look/feel updated regularly, and they benefit from content targeting and personalization to better address the end user needs.

As a result, developers must consider incorporating a content management system (CMS) into their development, allowing business users to edit and manage content and optimize the digital experience. However, traditional CMS platforms frustrate web developers, slowing down development and deployment of code. Legacy CMS solutions are monolithic, database-driven, with incomplete APIs. And they usually constrain developers to using old and restrictive UI tools and development processes.

What web developers need is a new approach to content management! A CMS built from the ground up for modern web (and mobile) app development. A CMS built upon microservices and built with modern robust enterprise tools (Spring, Groovy scripting). A headless CMS with REST APIs that deliver JSON (or XML) to any UI development framework (React, Angular, Node, vanilla JavaScript, really any UI technology).

A CMS based on the most popular source code control system (Git) that allows both content and code to seamlessly flow back and forth through the development and deployment cycles. A CMS that doesn't rely on a single database architecture, but a distributed and decentralized data tier that scales out on the elastic web. A CMS that enables and actually accelerates agile development. And a CMS that lets business users edit content and create personalized digital experiences without developer support. Developers and users working cohesively, but independently, together.

Now that would revolutionize modern day web development, wouldn't you say? Learn more at craftersoftware.com.

WRITTEN BY RUSS DANNER
VICE PRESIDENT, **PRODUCTS AT CRAFTER SOFTWARE**

PARTNER SPOTLIGHT

Crafter CMS

Crafter CMS enables rapid and agile development of innovative, high performance, omni-channel digital experiences.

CATEGORY
CMS, SaaS, PaaS

NEW RELEASES
Continuous

OPEN SOURCE
Yes

A major restaurant franchise that operates globally needed to revamp its digital presence for its e-commerce website and mobile app. Due to its modern approach to content management, Crafter CMS was selected as the content platform on which to build all their digital experiences.

With a large percentage of their revenue coming from online ordering, the restaurant brand's first priority was to build a modern, user-friendly web experience that allowed customers to quickly browse, find specials, and order food for delivery. Crafter CMS supports any UI framework, including all modern JavaScript frameworks such as React and Angular, so a fresh responsive user experience was easily created. In addition, a native mobile app was built, leveraging Crafter's Content-as-a-Service capabilities to deliver rich content through the mobile experience.

Crafter's backend infrastructure based on Spring and its native support for Groovy scripting allowed easy integration to the restaurant's ERP and e-commerce backend systems, allowing individual franchise owners to customize local promotions based on, among many factors, real-time inventory of ingredients. Because this restaurant chain is a major sponsor of nationwide (even global) sporting events during which large scale TV and social media advertising was in play, Crafter's high performance architecture and its ability to rapidly scale out elastically enables the handling of very large surges of web traffic and keep business running smoothly during these peak periods.

STRENGTHS

- The first and only enterprise CMS based on Git, enabling content and code to flow seamlessly throughout development and deployment cycles
- Headless CMS (Content-as-a-Service) with strong APIs and support for any UI framework
- User friendly in-context editing tools and multi-channel preview for content editors
- Shared-nothing architecture (No JCR, No Database) built for elastic scalability
- Very high performance content delivery

NOTABLE CUSTOMERS

- AT&T
- Mastercard
- Marriott
- Papa John's
- PGA Tour
- AppNexus
- Bank of New York Mellon
- Banca Mediolanum

WEBSITE craftersoftware.com

TWITTER @craftersofware

BLOG blog.craftersoftware.com

Using Kotlin with Ktor to Create Web Apps

QUICK VIEW

01 Learn how Kotlin can be used for developing web service applications in a simple and elegant way.

02 See what makes ktor, itself written in Kotlin, a very interesting library for the purpose of writing web apps with Kotlin.

03 Take a look at how Kotlin's powerful features like type-safe builders, coroutines and extension functions are utilized inside ktor.

04 Finally, learn how a ktor app can be tested with very little effort.

BY **SIMON WIRTZ**
SOFTWARE DEVELOPER, **NDESIGN**

When Google made Kotlin an official language for Android a few months ago at Google I/O, the language quickly gained a lot of popularity in the Android world. On the server side though, Kotlin is not as broadly adopted, and some people still seem to be cautious when backend services are involved. Other developers are convinced that Kotlin is mature enough and can safely be used for any server application in which Java could play a role otherwise.

If you want to develop web apps with Kotlin, you can choose from various web frameworks like Spring MVC/WebFlux, Vert.x, Vaadin, and basically everything available for the JVM. Besides the aforementioned frameworks, there's also a Kotlin specific library available for creating web applications, called ktor. After reading this article, you'll know what ktor is, its advantages are, and how you can quickly develop a web application in Kotlin.

KTOR

The web application framework ktor, written in Kotlin, is meant to provide a tool for quickly creating web applications with Kotlin. The resulting software may be hosted in common servlet containers, like Tomcat, or standalone in a Netty, for example. Whatever kind of hosting you choose, ktor is making heavy use of Kotlin coroutines, so that it's implemented 100% asynchronously and mainly non-blocking. ktor does not dictate which frameworks or tools you use, so you can choose whatever logging, DI, or templating engine you like. The library is pretty light-weight in general, but is still very extensible through a plugin mechanism.

One of the greatest advantages attributed to Kotlin is its ability to provide type-safe builders, also known as Domain Specific Languages (DSL). Many libraries already provide DSLs as an alternative to common APIs, such as the Android library Anko, the Kotlin html builder, or the freshly released Spring 5 framework. As we will see in the upcoming examples, ktor also makes use of such DSLs, which enable the user to define the web app's endpoints in a very declarative way.

EXAMPLE APPLICATION

In the following example, a small RESTful web service will be developed with ktor. The service will use an in-memory repository with simple Person resources, which it exposes through a JSON API. Let's look at the components.

PERSON RESOURCE AND REPOSITORY

According to the app's requirements, a resource "`Person`" is defined as a `data class`, and the corresponding repository is defined as an `object`, Kotlin's way of applying the Singleton pattern to a class.

```
data class Person(val name: String, val age: Int){
    var id: Int? = null
}
```

The resource has two simple properties which need to be defined when a new object is constructed, whereas the `id` property is set later when stored in the repository.

The `Person` repository is rather unspectacular and not worth observing. It uses an internal data store and provides common CRUD operations.

ENDPOINTS

The most important part of the web application is the configuration of its endpoints, exposed as a REST API. The following endpoints will be implemented:

The application won't support an update operator via `PUT`.

ENDPOINT	HTTP METHOD	DESCRIPTION
/person	GET	Requests all Person resources
/persons/{id}	GET	Requests a specific Person resource by its ID
/persons	DELETE	Requests all Person resources to be removed
/persons/{ID}	DELETE	Requests all Person resources to be removed by its ID
/persons	POST	Requests to store a new Person resource
/	GET	Delivers a simple HTML page welcoming the client

ROUTING WITH KTOR

Now ktor comes into play with its structured DSL, which will be used for defining the previously shown endpoints, a process often referred to as "routing." Let's see how it works:

```
fun Application.main() {
    install(DefaultHeaders)
    install(CORS) {
        maxAge = Duration.ofDays(1)
    }
    install(ContentNegotiation){
        register(ContentType.Application.Json,
        GsonConverter())
    }

    routing {
        get("/persons") {
            LOG.debug("Get all Person entities")
            call.respond(PersonRepo.getAll())
        }
    }
    // more routings
}
```

> ktor does not dictate which frameworks or tools you use, so you can choose whatever logging, DI, or templating engine you like.

The fundamental class in the ktor library is `Application`, which represents a configured and, eventually, running web service instance. In the snippet, an extension function `main()` is defined on `Application`, in which it's possible to call functions defined in `Application` directly, without additional qualifiers. This is done by invoking `install()` multiple times, a function for adding certain `ApplicationFeatures` into the request processing pipeline of the application. These features are optional and the user can choose from various types. The only interesting feature in the shown example is `ContentNegotiation`, which is used for the (de-)serialization of Kotlin objects to and from JSON, respectively. The Gson library is used as a backing technology.

The routing itself is demonstrated by defining the GET endpoint for retrieving all `Person` resources here. It's actually straight-forward since the result of the repository call `getAll()` is just delegated back to the client. The `call.respond()` invocation will eventually reach the `GsonSupport` feature, taking care of transforming the `Person` into its JSON representation. The same happens for the remaining four REST endpoints, which will not be shown explicitly here.

THE HTML BUILDER

Another cool feature of ktor is its integration with other Kotlin DSLs like the HTML builder, which can be found on GitHub. By adding a single additional dependency to a ktor app, this builder can be used with the extension function `call.respondHtml()`, which expects type-safe HTML code to be provided. In the following example, a simple greeting to the readers is included, which the web service exposes as its index page.

One of the greatest advantages attributed to Kotlin is its ability to provide type-safe builders, also known as Domain Specific Languages (DSL).

```
get("/") {
    call.respondHtml {
        head {
            title("ktor Example Application")
        }
        body {
            h1 { +"Hello DZone Readers" }
            p {
                +"How are you doing?"
            }
        }
    }
}
```

STARTING THE APPLICATION

After having done the configuration of a ktor application, there's still a need to start it through a regular `main` method. In the demo, a Netty server is started standalone by doing the following:

```
fun main(args: Array<String>) {
    embeddedServer(Netty, 8080, module =
    Application::main).start(wait = true)
}
```

The starting is made pretty simple by just calling the `embeddedServer()` method with a Netty environment, a port and the module to be started, which happens to be the thing defined in the `Application.main()` extension function from the previous example.

TESTING

A web framework or library is only applicable if it comes with integrated testing means, which ktor actually does. Since the `GET` routing for retrieving all `Person` resources was already shown, let's have a look at how the endpoint can be tested.

```
fun Application.main() {
@Test
    fun getAllPersonsTest() =
    withTestApplication(Application::main) {
        val person = savePerson(gson.
        toJson(Person("Bert", 40)))
        val person2 = savePerson(gson.
        toJson(Person("Alice", 25)))
        handleRequest(HttpMethod.Get, "/persons") {
            addHeader("Accept", json)
        }.response.let {
            assertEquals(HttpStatusCode.OK,
            it.status())
            val response = gson.fromJson(it.content,
            Array<Person>::class.java)
            response.forEach { println(it) }
            response.find { it.name == person.name } ?:
            fail()
            response.find { it.name == person2.name } ?:
            fail()
        }
        assertEquals(2, PersonRepo.getAll().size)
    }
```

This one is quite simple: Two resource objects are added to the repository, then the `GET` request is executed and some assertions are done against the web service response. The important part is `withTestApplication()`, which ktor offers through its `testing` module and makes it possible to directly test the `Application`.

The article presented basically everything worth knowing in order to get started with ktor web applications. For more details, I recommend the ktor homepage or the samples included in the ktor repository.

The complete source code of the developed web application can be found on GitHub, it's backed by a Gradle build.

SIMON WIRTZ has been having fun as a Software Developer with a focus on Java (web-)applications for almost 6 years now. Currently he works for a German software company in very different areas, mostly involved in Spring and OSGi development with Java. He started supporting Kotlin in January 2017 by blogging and speaking about it, and he uses the language whenever possible.

DIVING DEEPER INTO WEB DEVELOPMENT

TOP #WEBDEV TWITTER ACCOUNTS TO FOLLOW

@JOHN_PAPA
John Papa

@RAUSCHMA
Axel Rauschmayer

@UMAAR
Umar Hansa

@IREADERINOKUN
Ire Aderinokun

@ADDYOSMANI
Addy Osmani

@CHRISCOYIER
Chris Coyier

@PAUL_IRISH
Paul Irish

@SWIZEC
Swizec

@SARASOUEIDAN
Sara Soueidan

@RACHELANDREW
Rachel Andrew

BEST WEB DEVELOPMENT ZONES

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 UX.

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.

PERFORMANCE DZONE.COM/PERFORMANCE

Scalability and optimization are constant concerns for the Developer and Operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection to tweaks to keep your code as efficient as possible.

TOP WEB DEVELOPMENT REFCARDZ

NODE.JS BY DAVE WHITELEY

Node.js is a JavaScript runtime that runs on top of Google's open-source JavaScript engine called V8. Pairing JavaScript's naturally event-driven, asynchronous coding style with non-blocking I/O libraries makes Node.js fast, lightweight, and efficient.

ANGULARJS BY GIL FINK

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

REACT.JS ESSENTIALS BY HEMANTH H.M.

Gives you the essentials of React.js, from the architecture to the virtual DOM, from building components to styling them.

TOP WEB DEVELOPMENT BOOKS

LEARN JAVASCRIPT VISUALLY_BY IVELIN DEMIROV

The OWASP Top 10 is a list of the top ten most critical web application security risks. These are the most important things for security experts to focus on.

A SMARTER WAY TO LEARN JQUERY _BY MARK MYERS

This GitHub page contains an awesome variety of information security resources, providing you with everything from training material to online courses to books.

THE PRINCIPLES OF OBJECT-ORIENTED JAVASCRIPT BY NICOLAS C. ZAKAS

Learn what makes JavaScript functions unique, how to define your own constructors, inheritance patterns for types and objects, and more.

TOP WEB DEVELOPMENT PODCASTS

ADVENTURES IN ANGULAR

This podcast includes overviews of new technologies in the web dev world, AngularJS tutorials, and interviews with the Angular community.

THE BIG WEB SHOW

Whether you're a web designer or a web developer, there's something in The Big Web Show for you. The podcast includes episodes about the mysteries of UX, practical design discovery, and code.

JAVASCRIPT JABBER

This weekly podcast features in-depth talks with web developers about using JavaScript, career advice, and personal projects.

Executive Insights on the Current and Future State of Web Application Development

QUICK VIEW

01 Choose the right technology to ensure you deliver a good user interface, user experience, and customer experience across all platforms.

02 Ultimately, we will have collaborative tools that help develop web apps that work seamlessly across devices and form factors.

03 Developers need to: 1) be empathetic; 2) know the fundamentals of common languages; and, 3) have a desire to learn and take on more responsibility.

BY **TOM SMITH**
RESEARCH ANALYST, **DZONE**

To gather insights on the state of web application development today, we spoke with 12 executives who are familiar with the current state of the industry. Here's who we spoke to:

MATT CHOTIN SR. DIR. OF DEVELOPER INITIATIVES, APPDYNAMICS

MICHAEL BECKLEY CTO, APPIAN

GIL SERVER CEO, APPLITOOLS

MIKE KAIL CTO, CYBRIC

KEVIN BRIDGES CTO, DRUD

ANDERS WALLGREN CTO, ELECTRIC CLOUD

JIM MCKEETH DEVELOPER ADVOCATE, EMBARCADERO

LUCAS VOGEL FOUNDER, ENDPOINT SYSTEMS

CHARLES KENDRICK CTO, ISOMORPHIC SOFTWARE

MARK BROCATO ENGINEERING DIRECTOR, SENCHA

COLE FURFARO-STRODE LEAD SOFTWARE ENGINEER, SPARKPOST

PETE CHESTNA DIR. OF DEVELOPER ENGAGEMENT, VERACODE

KEY FINDINGS

01 The keys to developing web applications are **choosing the right technology and ensuring you deliver a good user interface (UI), user experience (UX), and customer experience (CX)**.

The number one key is technology choice. Customers go to development shops with a half-built app because they picked the wrong technology for the job. Avoid the outdated, overblown frameworks from the largest vendors and find the stable vendors with a proven track record, using up-to-date approaches that actually work everywhere they need to.

Understand what the customer is doing and what device they are using so you are able to provide a great UX that empowers users and customers and leads to business growth. Provide a common UX across all platforms, operating systems, and form factors.

DevOps platform upgrades, patches, security, and scaling should be built-in so you can focus on collaborating with your business to deliver high-quality apps. Automation will help you scale and produce better, more secure apps quickly.

02 **The most significant changes to the development of web applications have been the rising popularity of JavaScript and microservices**. JavaScript's explosion and the number of frameworks that are evolving are breathtaking. While they are great at providing a common experience across all screens, it can be challenging to keep up with them.

Companies are either implementing microservices or exploring ways to use them. We see this in push notifications and the need for services to provide features and functionalities end users have come to expect. Web application developers need to think more about collaborating with business operations to ensure alignment of data, process, and business rules with new CX.

03 Three industries were specifically mentioned for web development use cases: healthcare, defense, and technology, but there were plenty of technical use cases with automation, and these were the only industry examples mentioned multiple times. **This shows the breadth and depth of possibilities with web apps**.

Web applications allow the analysis of chemical assays and animal tests, as well as the analysis of contracts with partners and other functions. Another pharmaceutical is using web apps to improve patient adherence to and compliance with prescribed medicine. Web apps can also be used to accelerate clinical trials.

Web applications can be used to track and itemize costs involved in government bids through a network of suppliers and contractors so a company can provide the necessary transparency for bids that can be in the billions of dollars.

Chip manufacturers can use web applications to collaboratively analyze performance and defects in batches of chips to refine their manufacturing process, which results in higher quality chips at lower cost and less waste.

Other technical applications include management of structured data, providing DevSecOps, ensuring cross-platform compatibility, and providing tools that help make developers of web apps more productive by reducing two weeks of work to a half-day.

04 The most common **hurdles affecting the development of web applications are: 1) culture; 2) legacy systems; and, 3) complexity** on multiple fronts. There's a mismatch between developers, DevOps, and SecOps. Testing is still being done manually, and environment management is a constant struggle. It creates inefficiencies that lead to higher costs, lower job satisfaction, and degraded quality applications. Speed to market is bogged down by how hard it is to release software.

The modernization of legacy systems is challenging. Companies need to help their employees with modern development and technology. Legacy systems teams don't want to consider security, passwords, or international standards because of their inability to see the possibilities of expanded adoption of their app. They are taking a narrow view, and their app will not scale. Bad choices early on and a lack of flexibility can lead to an expensive and time-consuming re-work of the application.

It is also a challenge to provide a consistent UX and beautiful UIs, across multiple modalities. All of the changes are around tooling. Some developers don't even know what version of JavaScript they are transpiling to. How many versions of clients or your API are you going to support? Apps are no longer self-contained. You don't know which browser your user is using, you cannot control when browsers are updated, operating systems and devices are always changing and this affects the UX. You need to be able to see what the end user is experiencing.

05 The future of web application are **collaborative tools that help develop web apps that work seamlessly across devices and form factors**. These applications will use AI and machine learning to understand the patterns of users to improve relevance and usefulness, enhancing the UX which will result in the growth of the app. We'll have the ability to coordinate all of the different variables that can affect web applications to ensure they are fully synchronized before they are presented to the end user.

06 **Different aspects of security tend to be the biggest concerns** around web applications. The explosion of JavaScript is opening companies up to potential security vulnerabilities. We need to identify bad actors more quickly and get them off the system. We also need to be aware of the growth in insider threats and the opportunity for human mistakes.

We need to test application frontends and backends in challenging environments. Some apps and sites are still not running SSL by default. Open source is no more or less secure than first-party code; however, it does not age well. If a vulnerability is announced it needs to be fixed in a timely manner. The OWASP Top 10 are also still an ongoing concern, and, unfortunately, developers are not held accountable for secure code.

Additionally, we continue to see too much reliance on technology and not enough on business outcomes. Inherent in this is that we need to keep the business, and its data, secure.

07 Three skills mentioned most frequently that developers need to build effective web apps are: **1) empathy; 2) knowledge of the fundamentals of common languages; and, 3) a desire to learn and take on more responsibility**.

Empathize with your users and those you are working with. Ask yourself what the end user is trying to accomplish with your app. Walk through the steps your users are taking to understand their pain points and their workflow. Have a good grasp on the hot new technologies, but empathy for the user so you can create the best UX.

Know the fundamentals of the most popular languages, whether you're building with JavaScript, HTML, or CSS. Know the right CSS frameworks, JavaScript frameworks, and layout requirements so you can build a good UI/UX to manage the middleware layer.

You need to have a desire to continuously improve yourself, your apps, and the development process. Begin by using the principles of Continuous Delivery and DevOps. Measure what matters, collaborate with your team, automate your pipeline, and improve. A full-spectrum, well-rounded engineer is responsible for quality, unit tests, security, and containerization.

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.

Solutions Directory

This directory contains platforms, frameworks, and libraries to build and maintain web applications. It provides free trial 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	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
ActiveState	Komodo IDE	PaaS	21 Days	activestate.com/komodo-ide
Adobe	PhoneGap	Hybrid Web Development Platform	Open Source	phonegap.com
Afilias Technologies	DeviceAtlas	Device Detection	30 Days	deviceatlas.com
Amazon	AWS Elastic Beanstalk	PaaS	Free tier available	aws.amazon.com/elasticbeanstalk
Apache	Cordova	Hybrid Web Development Platform	Open Source	cordova.apache.org
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, Package 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

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
CloudFlare	CloudFlare	JavaScript CDN	Free tier available	cloudflare.com/cdn
CoffeeScript	CoffeeScript	Language	Open Source	coffeescript.org
CommonJS	CommonJS	Package Manager	Open Source	commonjs.org
Crafter Software	Crafter CMS	CMS, Development Platform	Available by request	craftersoftware.com/products/overview
CSS Crush	CSS Crush	CSS Preprocessor	Open Source	the-echoplex.net/csscrush
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
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
Google	Google Hosted Libraries	JavaScript CDN	Open Source	developers.google.com/speed/libraries
Grails	Grails	Java Web Framework	Open Source	grails.io

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Grunt	Grunt	Build Script Tool, Package Manager	Open Source	gruntjs.com
Gulp	Gulp	Build Script Tool	Open Source	gulpjs.com
Hewlett Packard Enterprise	HPE Helion	PaaS	Available by request	hpe.com/us/en/solutions/cloud.html
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
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

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Micro Focus	Unified Functional Testing	Web Testing	60 Days	software.microfocus.com/en-us/software/uft
Microsoft	Visual Studio Core	IDE	Free Solution	asp.net
Microsoft	ASP.NET	Web Framework	Free Solution	asp.net
Microsoft	Microsoft Azure	PaaS	Free tier available	azure.microsoft.com
Microsoft	TypeScript	JavaScript Superset	Open Source	TypeScriptlang.org
Microsoft	Microsoft Ajax CDN	JavaScript CDN	Open Source	docs.microsoft.com/en-us/aspnet/ajax/cdn/overview
MochaJS	Mocha	Web Testing	Open Source	mochajs.org
Myth	Myth	CSS Preprocessor	Open Source	github.com/segmentio/myth
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
Perfecto Mobile	CQ Lab	Automated web and mobile testing	Available by request	perfectomobile.com/solutions/perfecto-test-automation
PhantomJS	PhantomJS	Web Testing	Open Source	phantomjs.org
Pivotal Labs	Jasmine	Web Testing	Open Source	jasmine.github.io

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Pivotal Software	Spring	Java Web Framework	Open Source	spring.io
Polymer Authors	Polymer	Web Components Library	Open Source	polymer-project.org
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
Rainforest	Rainforest	Automated Mobile and Web Testing	Available by request	rainforestqa.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
Rogue Wve 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/platform
Sencha	Ext JS	JavaScript Framework	30 Days	sencha.com/products/extjs
Sencha	Sencha GXT	JavaScript Framework	30 Days	sencha.com/products/gxt

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
SensioLabs	Symfony	PHP Web Framework	Open Source	symfony.com
StackPath	SecureCDN	JavaScript CDN	15 Days	stackpath.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
The jQuery Foundation	jQuery CDN	JavaScript CDN	Open Source	code.jquery.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
webcomponents.org	Web Components	Web Platform API Hosting Service	Open Source	webcomponents.org
Webpack	Webpack	Package Manager	Open Source	webpack.github.io
WebRTC	WebRTC	Real Time Communication Through APIs	Open Source	webrtc.org
wzrd.in	wzrd.in	Browserify-as-a-Service	Open Source	wzrd.in

GLOSSARY

ANGULAR
A platform for building HTML and JavaScript web applications, led by Google and based on TypeScript. It is a rewrite of the AngularJS JavaScript framework.

COROUTINE
Coroutines simplify asynchronous programming by putting the complications into libraries. The logic of the program can be expressed sequentially in a coroutine, and the underlying library will figure out the asynchrony.

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.

INJECTION ATTACK
A scenario where attackers relay malicious code through an application to another system for malicious manipulation of the application. These attacks can target an operating system via system calls, external programs via shell commands, or databases via query language (SQL) injection.

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.

KOTLIN
A language that runs on the JVM, developed by JetBrains, provided under the Apache 2.0 License, offering both object-oriented and functional features.

MICROSERVICES ARCHITECTURE
An architecture for an application that is built with several modular pieces, which are deployed separately and communicate with each other, rather than deploying one single piece of software.

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 serverside. It is event-based, nonblocking, and asynchronous.

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

SINGLE-PAGE APPLICATION
A web application that responds to user actions and changes over time by rewriting the app, rather than loading new web pages.

TYPESCRIPT
A superset of JavaScript developed by Microsoft that adds static typing.

USER INTERFACE (UI)
A term to describe the ways in which the end user directly interacts with a device or application.

USER EXPERIENCE (UX)
A term to describe all aspects of the end user's interaction with an application.

VUE.JS
A JavaScript framework built for designing user interfaces and single-page applications, designed to be adopted incrementally.