DZone
A DEVADA MEDIA PROPERTY

THE 2018 DZONE GUIDE TO

Dynamic Web & Mobile Application Development

VOLUME V

BROUGHT TO YOU IN PARTNERSHIP WITH

Cloudinary

GrapeCity

outsystems

Perfecto

Dear Reader,

Welcome to the *2018 DZone Guide to Dynamic Web and Web Mobile Application Development*. Since February 2017, I have managed the Web Dev Zone here on DZone. In that time, the fields of web and mobile development have seen interesting changes, from backend runtimes to how developers create dynamic UIs and mobile applications.

And at the heart of this revolution sits JavaScript. A language originally created to only execute on the client-side of a web browser now has the ability to create servers and backend logic, run native mobile applications, and, of course, deliver expectational user experience on the front-end. Through the use of particular tech stacks, such as MEAN (Mongo-Express-Angular-Node) or MERN (Mongo-Express-React-Node), developers can now create full-stack applications using the JavaScript ecosystem. Additionally, the emergence of React Native and NativeScript gives JavaScript developers the ability to create fully native mobile applications.

But JavaScript is not the only actor on the stage of web development. Drawing upon other subfields of software development, such as DevSecOps, database development, and integration engineering, web and mobile application developers are being tasked with integrating authentication protocols into their code, integrating APIs and API calls, and sending database requests. Machine learning, big data, and AI are even making in-roads in web development, allowing sites and their developers to monitor user behavior for patterns. Web development is now, more than ever, a multi-faceted field.

In this guide, expert mobile and web developers from across software industry cover various topics that relate to web and mobile app development. We'll take a look at securing and managing APIs with access tokens, how mobile applications utilize APIs, testing applications, and more.

Given the astronomical amount of information, technologies, and tools available on web and mobile development, there's no way one developer can master them all. But we hope this guide helps you achieve your development goals and learn something interesting along the way.

Thank you again for reading and your continued support of DZone. We hope you enjoy the *2018 DZone Guide to Dynamic Web and Mobile Application Development*.

By Jordan Baker
CONTENT COORDINATOR, DEVADA

Table of Contents

DZone is...

BUSINESS & PRODUCT
MATT TORMOLLEN, CEO
MATT SCHMIDT, PRESIDENT
JESSE DAVIS, EVP, TECHNOLOGY
KELLET ATKINSON, MEDIA PRODUCT MANAGER

SALES
CHRIS BRUMFIELD, SALES MANAGER
FERAS ABDEL, SALES MANAGER
JIM DYER, SR. ACCOUNT EXECUTIVE
ANDREW BARKER, SR. ACCOUNT EXECUTIVE
BRETT SAYRE, ACCOUNT EXECUTIVE
ALEX CRAFTS, KEY ACCOUNT MANAGER
BRIAN ANDERSON, KEY ACCOUNT MANAGER
SEAN BUSWELL, SALES DEV. REP.
JORDAN SCALES, SALES DEV. REP.
DANIELA HERNANDEZ, SALES DEV. REP.

PRODUCTION
CHRIS SMITH, DIRECTOR OF PRODUCTION
ANDRE POWELL, SR. PRODUCTION COORD.
G. RYAN SPAIN, PRODUCTION COORD.
BILLY DAVIS, PRODUCTION COORD.
NAOMI KROMER, SR. CAMPAIGN SPECIALIST
JASON BUDDAY, CAMPAIGN SPECIALIST
MICHAELA LICARI, CAMPAIGN SPECIALIST

MARKETING
SUSAN WALL, CMO
AARON TULL, DIR. OF MARKETING
SARAH HUNTINGTON, DIR. OF MARKETING
WAYNETTE TUBBS, DIR. OF MARKETING COMM.
ASHLEY SLATE, SR. DESIGN SPECIALIST
KRISTEN PAGÀN, MARKETING SPECIALIST
COLIN BISH, MARKETING SPECIALIST
LINDSAY POPE, CUSTOMER MARKETING MAN.
SUHA SHIM, ACQUISITION MARKETING MAN.

EDITORIAL
SUSAN ARENDT, EDITOR-IN-CHIEF
MATT WERNER, PUBLICATIONS COORDINATOR
SARAH DAVIS, PUBLICATIONS ASSOCIATE
KARA PHELPS, CONTENT & COMMUNITY MANAGER
TOM SMITH, RESEARCH ANALYST
MIKE GATES, CONTENT TEAM LEAD
JORDAN BAKER, CONTENT COORDINATOR
ANNE MARIE GLEN, CONTENT COORDINATOR
ANDRE LEE-MOYE, CONTENT COORDINATOR
LAUREN FERRELL, CONTENT COORDINATOR
LINDSAY SMITH, CONTENT COORDINATOR

Executive Summary

KARA PHELPS - CONTENT AND COMMUNITY MANAGER, DEVADA

Web application development continues to skyrocket in demand, without a slowdown in sight. Every organization needs a web developer (or two, or 10) to help reach customers and interact with them, and the nature of the job is always in flux. New languages, tools, and frameworks are constantly overtaking each other in the race to solve the latest critical issue. It's a face-paced industry with a thriving community.

The overlap between the fields of web dev and mobile also keeps growing. It's no longer just "mobile-first" — mobile is a given. This year, we combined our web development survey and our mobile survey for the first time. We asked 818 professionals in web and mobile development to weigh in on the state of the industry and the range of technologies they use. Here are a few key trends we noticed.

ANGULAR STILL DOMINATES, BUT OTHER OPTIONS ARE GETTING LOUDER

DATA

55 percent of survey respondents have used the Angular framework; 52 percent have used AngularJS. React trails at 40 percent. 66 percent of respondents, however, are interested in using React, compared to 43 percent who are interested in using Angular, and 30 percent who are interested in using AngularJS.

IMPLICATIONS

Angular doesn't have backwards compatibility, so it's a bit surprising to see so much interest in its predecessor, AngularJS — although it makes sense that developers wouldn't want to refactor their existing code. Front-end web developers are still predominantly most familiar working with the Angular and AngularJS frameworks, but they are increasingly drawn toward React. While React is technically a library, not a framework, it offers a wide range of open-source integrations and maintains performance during asynchronous data requests.

RECOMMENDATIONS

New developers would be well-served to learn JavaScript, and it certainly wouldn't hurt to get familiar with Angular and TypeScript, the subset of JavaScript that Angular uses. They still dominate the current landscape. However, major companies rely on React, like Facebook (which created it) and AirBnB. React's popularity among developers is also significant, so this is a good time to study up.

YOU'RE PROBABLY BUILDING A WEB APP

DATA

85 percent of survey respondents are currently developing web applications and services (otherwise known as software-as-a-service, or SaaS). 50 percent of respondents are developing enterprise business apps, and 27 percent are developing native mobile apps.

IMPLICATIONS

As cloud adoption continues to increase and mature at the enterprise level, more organizations are creating web apps to meet the needs of cloud-based businesses — and those of consumers.

RECOMMENDATIONS

SaaS operates on a subscription model, asking users to purchase a license and typically support services. Organizations can then deliver software to users without asking them to install anything, and standardized APIs integrate customer data and various functionalities with on-premise software and other SaaS applications. When developing web apps like this, it's crucial to put processes in place to make sure they're consistently available, easily updated, and secure.

IT'S CHROME AND ANDROID'S WORLD; WE JUST LIVE IN IT

DATA

97 percent of survey respondents said they actively develop for Chrome, 64 percent for Mozilla Firefox, 41 percent for Internet Explorer; 30 percent for Safari, and 26 percent for Edge. As for mobile platforms, 82 percent of survey respondents said they develop apps for Android, 59 percent for iOS, and 25 percent for React Native.

IMPLICATIONS

Chrome is the overwhelming favorite browser for developers, thanks in part to its excellent developer tools (although Firefox has a great set as well). Chrome is also widely used outside the tech community, helping to ensure a broad reach for Chrome-targeted web apps. When it comes to mobile, Android is still the top choice. Apps using this platform tend to be written in Java, and 82 percent of survey respondents (equal to those who said they develop for Android) reported that their company uses the Java ecosystem.

RECOMMENDATIONS

Chrome and Firefox use may be widespread, but you should still test your app in other browsers — people do use them. And, while iOS products dominate in North America and parts of Europe, Android is still the preferred platform for app development.

STRUGGLING TO GET A GRIP ON TEST RESULTS DATA?

WE KNOW THE CHALLENGE OF CLIMBING THROUGH MOUNTAINS OF TEST DATA AND HAVE DEVELOPED **DIGITALZOOM™** TO CONQUER DATA WITH SPEED AND EFFICIENCY. IT'S LIKE HAVING YOUR VERY OWN SHERPA.

DIGITALZOOM™

INSIGHTS INTO CI PIPELINE
RISK/FOCUS AREA MAPPING
VISUAL VALIDATION
SUMMARY RESULTS LIST
SINGLE TEST REPORT

Perfecto

MORE INFO AT PERFECTO.IO/DIGITALZOOM

Conquering Test Results Data With Visual Analytics

"In three years, every product [we make] will be obsolete. The only question is whether we will make them obsolete or somebody else will."
– ***Bill Gates***

If this quote doesn't make you feel nervous about continuously innovating in your apps, you are one cool cucumber.

Most of you are practicing Agile; some are on the path towards a full Continuous Integration/DevOps environment. If you buy into Bill Gates's assertion, we are all in a perpetual state of innovation. It's challenging to have a career in technology and not embrace constant change!

Testing and validation are crucial to keeping DevOps teams on track and innovating new features and capabilities. If your builds/commits don't pass the regression test suite, then you're taking 2 steps forward only to take one step back – and that's just a regression suite.

Clients who scale up test automation discover that increased test cases yield tons of test results data. It's critical to harness technology to crunch the results and identify real issues.

Perfecto has solved the pain of test automation results with:

- Insights into the CI pipeline
- Risk/focus heat mapping
- Summary results list
- Single test report
- Visual validation

Open-source solutions like TestNG or JUnit have their place but come with delays, security issues, and don't scale well.

THE BOTTOM LINE

DigitalZoom[tm] is the DevOps team's best tool to conquer mountains of results data. Our visual analytics tool connects to any CI, offers a heatmap showing where risk areas lie, and includes drill-downs to test artifacts with speed and efficiency.

There are many ways to look at test results data; as you scale, you need a tool like DigitalZoom[tm] that cuts through the noise and gets straight to what's important — getting back to work.

WRITTEN BY TZVIKA SHAHAF
DIRECTOR OF PRODUCT MANAGEMENT, PERFECTO

PARTNER SPOTLIGHT

Perfecto DigitalZoom™ Visual Analytics

Holistic test results visibility in one lighting fast, easy-to-use interface

CATEGORY
DigitalZoom™Visual Analytics for Test Results Data

RELEASE SCHEDULE
Continuous Deployment

OPEN SOURCE?
No

CASE STUDY

DigitalZoom™ puts a powerful **management dashboard** at your fingertips, giving you a quick, yet comprehensive overview of the risk areas of your app, across all platforms. Fast defect triaging allows you to pinpoint and resolve issues early. You can also generate customized test reports based on tags, platforms, and other key data to assess test coverage, understand current quality risks, and report feedback to every member of the DevOps team. Whatever your role in an organization, DigitalZoom™ gives you the tools you need to streamline your development pipeline.

STRENGTHS

Getting a handle on massive test results data translates into efficiency and productivity gains by:

1. **Keeping track of application health** - On-demand access to digital quality dashboard and custom reports - enables quick go/no-go decisions
2. **Addressing issues quickly** - allowing more time to focus on feature development and validation, less on analysis
3. **Eliminating flaky tests** for faster cycles
4. **Bridging communication gaps** across the DevOps team and fostering ongoing communication across Dev, QA, and Biz

NOTABLE USERS

- AT&T
- GE
- Ulta Beauty
- Toyota
- National Australia Bank

WEBSITE perfecto.io | **TWITTER** @perfectomobile | **BLOG** blog.perfecto.io

Key Research Findings

BY JORDAN BAKER - CONTENT COORDINATOR, DEVADA

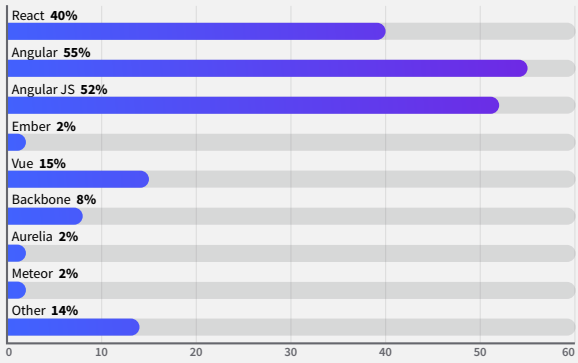
DEMOGRAPHICS

For this year's DZone Guide to Dynamic Web and Mobile Application Development, we surveyed our community of software professionals to get their thoughts on the state of the field. We received 1,202 responses, with a 64% completion rating. Based on the number of responses, we have calculated the margin of error for this survey at 4%.

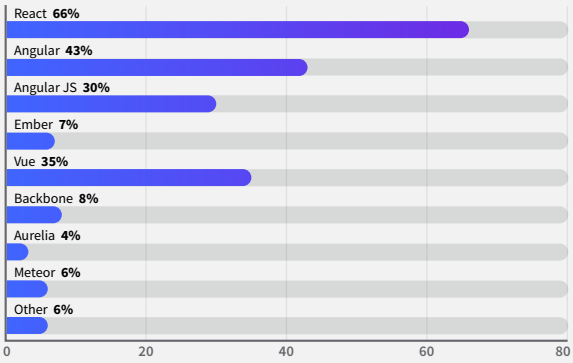
Below is a quick snapshot of the demographics of those surveyed.

- The average respondent has 14 years of experience in the industry.
- Respondents reported four main geographical areas where their companies are located:
  - 32%: USA
  - 25%: Europe
  - 12%: South America
  - 11%: South-Central Asia
- Respondents reported working in one of three main verticals:
  - 20% work for a software vendor
  - 19% are employed in the finance/banking industry
  - 10% work on e-commerce platforms
- Most survey-takers work in enterprise-sized organizations:
  - 24% for organizations sized 100-999
  - 21% for organizations sized 1,000-9,999
  - 21% for organizations sized 10,000+
- A majority of respondents work on immediate teams of ten people or less:
  - 33% work on a team of 6-10 people
  - 32% work on a team of 2-5 people
  - 14% work on a team of 11-15 people
- Most respondents work in one of three mains roles:
  - 41% are developers/engineers
  - 21% identify as developer team leads
  - 16% are employed as architects
- Respondents reported developing three main types of applications:
  - 85% develop web applications/services (SaaS)
  - 50% develop enterprise business apps
  - 27% develop native mobile applications
- Several important programming language ecosystems were reported:
  - 82% reported working in the Java ecosystem
  - 77% reported working in the client-side JavaScript ecosystem
  - 42% reported working in the Node.js ecosystem
  - 33% reported working in the Python ecosystem
  - 32% reported working in the C# ecosystem

1. WHICH OF THESE CLIENT-SIDE JAVASCRIPT FRAMEWORKS *HAVE* YOU USED?



2. WHICH OF THESE CLIENT-SIDE JAVASCRIPT FRAMEWORKS ARE YOU *INTERESTED* IN USING?



- Java, however, dominated the main programming languages used:
  - 60% use Java
  - 11% use C#
  - 11% use JavaScript (client- and/or server-side)
  - 7% use Python

FULL-STACK DEVELOPMENT

There exist three main forms of web application development: full-stack, front-end, and back-end. Among respondents, 48% work more with full-stack applications, 44% work with back-end apps, and only 7% concentrate on front-end applications. Additionally, there proved three main languages used among respondents: JavaScript (84%), HTML/CSS (75%), and Java (71%). The dominance of these three languages leads to the conclusion that respondents who develop full-stack applications mostly use Java as their backend language (though, as we'll see, the use of Node.js is increasing), JavaScript for DOM manipulation and other front-end logic, and HTML/CSS for styling.

For the rest of this section, we'll divide full-stack development into its constituent parts: front-end and back-end. We'll examine how the developers in our response population create these two types of applications.

FRONT-END WEB DEVELOPMENT: JAVASCRIPT FLAVORS AND FRAMEWORKS

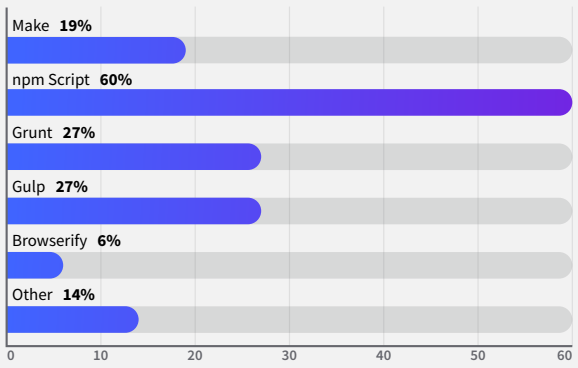
As noted above, JavaScript proved the most popular language for web application development among respondents. JavaScript, however, differs from many other popular development languages in that there is no one, true JavaScript. While the ECMAScript standard has been established, there exist several supersets under the main JavaScript umbrella other than ECMAScript, such as TypeScript, CoffeeScript, and Elm.

Of all these "flavors" of JavaScript, though, two established themselves as the most popular among our respondents: TypeScript (64%) and ES6 (58%). Interestingly, when we asked which flavor of JavaScript respondents are interested in (rather than which ones they've used), the percentages around TypeScript stayed rather static, but the percentages for ES6 fell by 10%, going from 58% who use it to 48% who are interested in it. And, though its adoption rate is low, CoffeeScript garnered a fair amount of interest as compared to its usage statistics. 12% of respondents reported using CoffeeScript, whereas 19% reported an interest in learning more about the language.

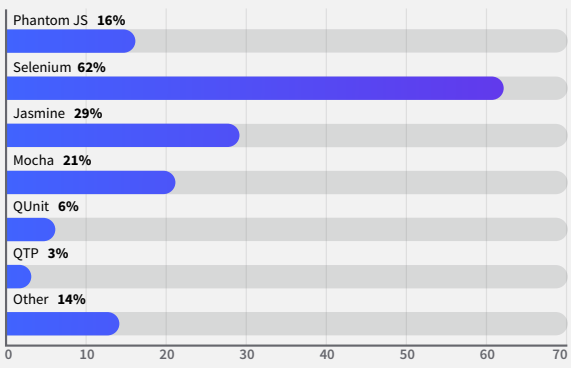
When setting up a front-end development environment, languages are only one part of the equation. Frameworks play an extremely important role in the creation of front-end apps, and three have come to dominate the landscape: React, Vue, and Angular. When we asked which of these frameworks respondents have used, 55% told us Angular, 52% reported AngularJS (i.e. the 1.x version of the Angular framework), 40% said React, and 15% said Vue. Much like we saw with the flavors of JavaScript, however, the frameworks developers are interested in differs from those they have used. When asked which client-side JavaScript framework they're interested in using, 66% reported React, 43% said Angular, 35% told us Vue, and 30% said AngularJS.

The steep drop-off in AngularJS is not surprising given that the framework is now on version seven and has since switched to using TypeScript as its primary language. The dramatic increase in React and Vue, though, is of interest. React saw a 26% increase (40% reporting to have used it and 66% reported to be interested) and Vue went up by 20% (15% used vs. 35% interested in). Interestingly, current versions of Angular dropped 12% between these two categories. While the unpopularity of Angular as compared to React at first seems surprising given the wide

3. WHAT BUILD SCRIPT TOOLS DO YOU USE?



4. WHAT DO YOU USE TO TEST YOUR WEB APPS?



spread adoption of Angular, this actually adheres to trends in the wider developer community. Earlier in 2018, Stack Overflow released a survey report that took into account the responses of over 100,000 developers and technologists. In this survey, React finished as the most loved web development framework and garnered the second most votes of any development framework (only behind TensorFlow). Additionally, React finished as the most wanted framework in this poll.

When we compare the popular flavors of JavaScript to the main JavaScript-based frameworks used in front-end development, we see some interesting trends. Among respondents who code in React, 74% use ES6 and 74% use TypeScript. Among Vue developers, 83% use ES6 and 64% use TypeScript. While both of these frameworks were originally designed to be used with ES6-style JavaScript, the popularity of TypeScript has caused support for the language to appear in both of these frameworks. Docs on TypeScript support for Vue.js can be found here and for React.js here. It is interesting to note how significantly more popular TypeScript has proven thus far among React developers than Vue developers. This difference could simply be due to the fact that React is backed by Facebook and has had more resources to put into their support of TypeScript, from both a coding and community marketing perspective.

BACKEND DEVELOPMENT

THE RISE OF NODE.JS

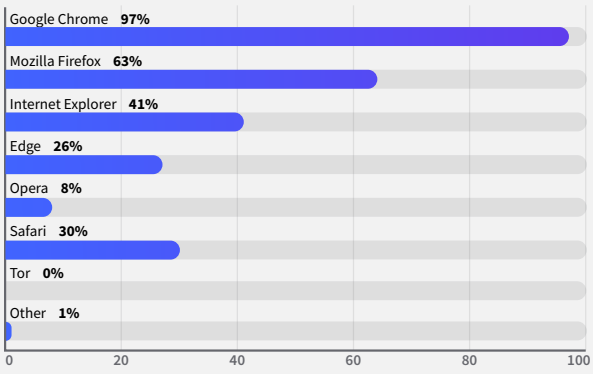
90% of respondents reported using JavaScript on the client-side; as discussed above, this is to be expected. Interestingly, 41% of respondents use JavaScript on the server-side, up from the 36% of respondents who targeted the server-side with their JavaScript in our *2017 DZone Guide to Web Development*. This high adoption rate of JavaScript on the back-end correlates with the increase in the usage rates for the Node.js runtime. In this year's survey, 42% of survey takers reported that their organization uses the Node.js ecosystem. This is up from 35% in 2017, nearly mirroring the growth rate of server-side JavaScript over the past year.

Of those respondents who work with the Node.js ecosystem, 54% do so on full-stack development projects and 40% on backend development projects. When we correlate our data on respondents who work in the Node.js ecosystem with our data on databases used for web applications, non-relational (or NoSQL) databases have higher rates of adoption among Node.js developers. For respondents whose organizations use Node.js, 56% use MongoDB. For those who report using the Java ecosystem, 43% use MongoDB; among Python ecosystem users, 49% use MongoDB; and with C# ecosystem adopters, 37% choose MongoDB as their database. This higher rate of adoption for MongoDB among Node developers versus those who work in other backend environments such as Java, Python, and C# could well be due to the asynchronicity of both technologies. Additionally, many developers use Mongoose, an open-source data schema solution, with MongoDB (source). Mongoose utilizes the JavaScript language, thus allowing Node.js developers to write their database logic in the same language as their application logic. For a more detailed discussion on databases, see the *2018 DZone Guide to Databsaes: Relational and Beyond*.

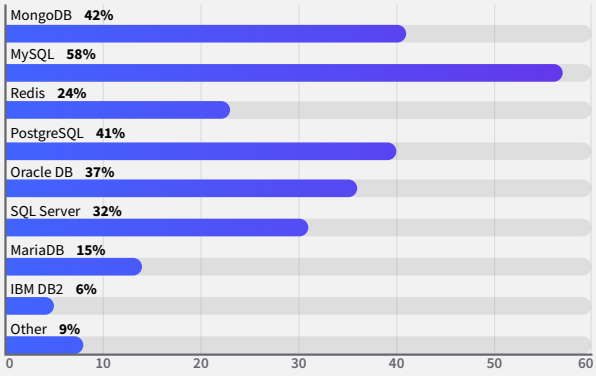
SERVER-SIDE OPERATIONS

The back-end of an application is a complicated place, full of API and database calls, logic, and more. When we asked how respondents typically divide their work between client and server, 76% told us the server-side exposes APIs, 70% have the server-side perform business logic, and 66% use the server-side to integrate systems such as databases, message queues, and EIS. Despite the growth of Node.js noted above, Java remains the dominant language for performing such operations on the backend. Thus, for the rest of this section, we'll use the

5. WHICH BROWSERS DO YOU ACTIVELY DEVELOP FOR?



6. WHAT DATABASES ARE YOU USING WITH YOUR WEB APPS?



statistics gathered from respondents who use Java to build web applications as our means of comparative analysis.

Despite the popularity of MongoDB among Node.js developers noted above, web developers working in Java seem to prefer traditional SQL databases. Among those respondents who told us they build web apps with Java, 61% use MySQL databases, 46% use Oracle DB, 45% use MongoDB, and 43% use PostgreSQL. Comparing these numbers to the adoption rates of these databases among the general survey population, Oracle DB proved more popular among Java-based web developers. Among the general survey population, 58% use MySQL, 42% use MongoDB, 41% use PostreSQL, and 37% use Oracle DB for their database needs. One potential explanation for Oracle DB's higher than average popularity among Java-based web developers is that both the Java language and Oracle DB are developed by the same organization and would thus be made to work well together.

When it comes to pushing data to the server, 67% of the general survey population use the WebSocket API, 34% use HTTP streaming, 25% use webhooks, another 25% use polling, and 19% reported using server-sent events. When we compare these numbers to our Java-based web developers, these percentages all dramatically fall. Among Java web developers, 37% use the WebSocket API, 19% use HTTP streaming, 14% use polling, 12% use webhooks, and 11% use server-sent events.

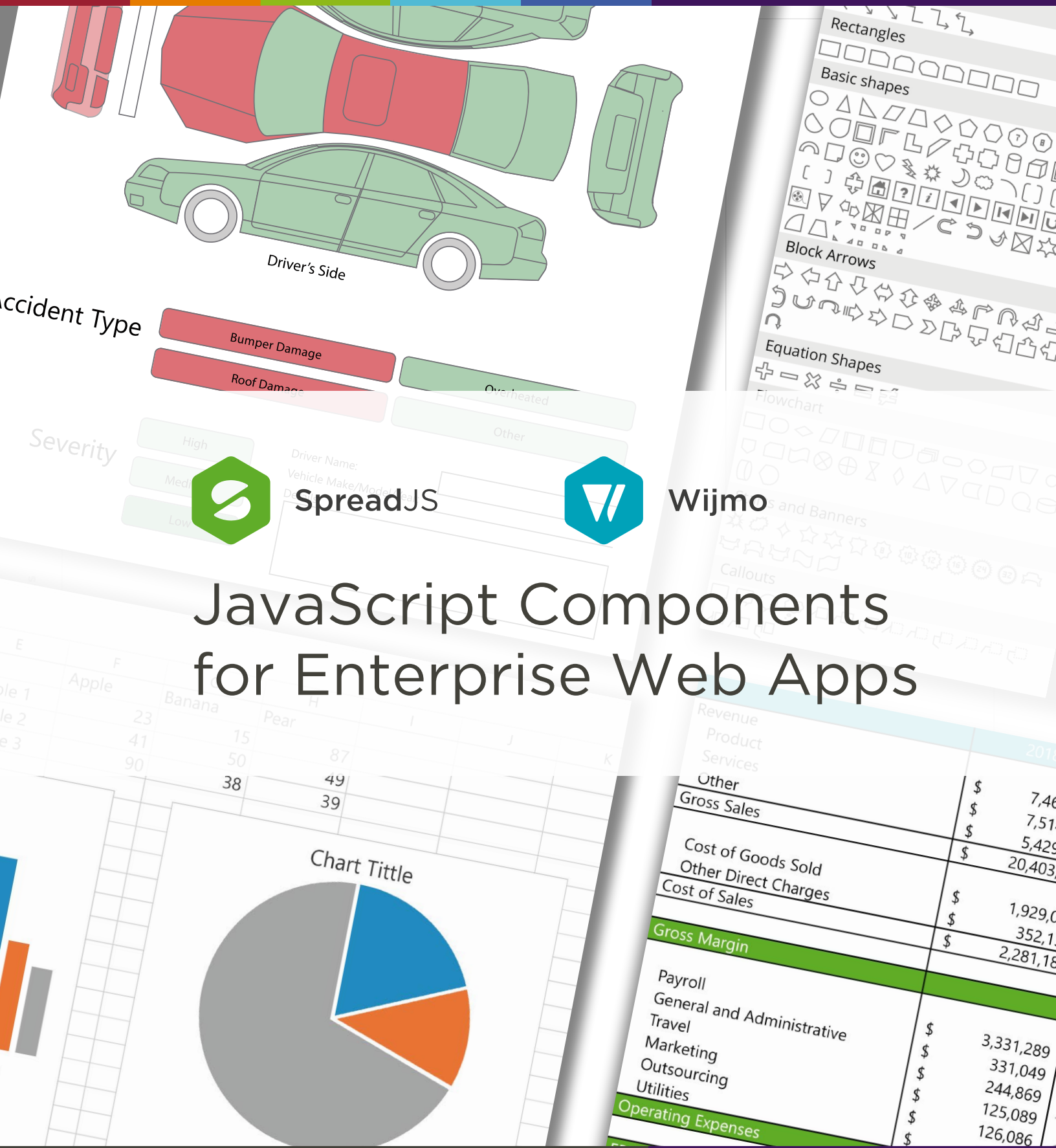
The adoption rate of the three most popular web servers among respondents (Apache Tomcat, Apache Web Server, and NGINX), also differed between the general survey population and Java web developers, though not as dramatically as in the case of the means of pushing data to the server. Among the general population, 62% reported using Apache Tomcat as their web server, 50% said they use the Apache Web Server, and 55% reported using NGINX. Among those respondents who use Java to build web apps, 75% use Apache Tomcat, 52% Apache Web Server, and 44% NGINX.

ENVIRONMENTS TARGETED: JAVASCRIPT AS A LANGUAGE FOR MOBILE DEVELOPMENT

When it comes to the environments targeted by web developers, one browser proved almost unanimous among respondents. 97% of survey takers reported actively developing for Google's Chrome browser. The runner up, Mozilla Firefox, is targeted by 64% of developers, followed by Internet Explorer at 41%, Safari at 30%, and Edge at 26%. Clearly, Chrome dominates the landscape of desktop browser-based development efforts. Interestingly, the results prove much the same when we look at the browsers targeted by hybrid application developers. 99% of respondents who develop hybrid applications develop for Chrome, 68% for Mozilla Firefox, 40% for Internet Explorer, 37% for Safari, and 28% for Edge.

Hybrid and native application developers target more than just the browser; they also have to worry about how their applications perform on various operating systems. 82% of respondents who develop native or hybrid applications do so for Android and 59% for iOS. Another 25% of native and hybrid app developers focus their efforts on React Native. While the prominence of Android and iOS is unsurprising, the proliferation of React Native as a popular mobile application development framework has led to an increase in the use of JavaScript as a language for mobile development.

When we asked which languages respondents are currently using for building native or hybrid mobile apps, 72% reported using JavaScript and 68% said Java, while there were only 24 write-in responses for Swift (the official language for developing iOS software) and 5 or Objective-C (a popular language for iOS development). When we compare the two main languages, JavaScript and Java, to our data on mobile platforms, an interesting pattern emerges. Among those respondents who develop for Android, 74% use Java and 65% use JavaScript. Among those respondents who develop for iOS, 77% use JavaScript and 65% use Java. This high percentage of JavaScript in both Android and iOS development seems due largely to frameworks such as React Native. According to React Native's documentation, it allows developers to "build a real mobile app that's indistinguishable from an app built using Objective-C or Java." This most assuredly accounts for the remarkably low use of Swift and Objective-C among this year's respondents. Diving into this data a little further, though, reveals that JavaScript may be more popular as a language for hybrid application development. Among respondents who develop hybrid applications, 87% use JavaScript and 63% use Java. For survey takers who create native mobile apps, 75% use Java and 61% use JavaScript.



SpreadJS

Wijmo

JavaScript Components for Enterprise Web Apps

Download your free trial at **GrapeCity.com**

GrapeCity

Expanding Developers' Capabilities With Complete JavaScript Components

SpreadJS 12 brings a major feature enhancement to your favorite JavaScript spreadsheet component: shapes! What will you do with data-driven shapes?

Comprised of both built-in and custom shapes, this new feature gives you the power to enhance your spreadsheets and applications with data-driven graphics and flowcharts. With 60+ built-in Excel-like shapes, you can:

- Import and export shapes-based Excel documents seamlessly
- Draw flowcharts with connectors
- Create annotations with arrows
- Add action buttons like play or skip to your apps
- Generate Gantt charts from a spreadsheet schedule

SpreadJS custom shapes can be implemented in many different ways. You can draw shapes to your specifications and add interactions like highlights, information callouts, or take some database action.

The shapes feature can be used to make different kinds of interactive diagrams. You can create a visual, interactive floor plan that allows users to see who sits at a desk or add information about amenities. Construct a production floor plan for a manufacturing facility, and highlight areas experiencing slowdowns or problems. You can also design an interactive, touch-based map of a car so users can highlight damage for insurance claims.

Power up your application with both built-in and custom shapes that enhance your spreadsheets with data-driven graphics, flowcharts, Gantt charts, and annotations.

GrapeCity's family of products provides developers, designers, and architects with the ultimate collection of easy-to-use tools for building sleek, high-performing, feature-complete applications. In addition to SpreadJS, GrapeCity's JavaScript offerings include Wijmo, high-performance HTML5/JavaScript UI controls for building enterprise-grade applications. Wijmo provides a complete collection of time-saving HTML5/JavaScript UI controls for building touch-first and lightweight applications. Wijmo's high-speed UI controls include FlexGrid, the industry's best Angular data grid.

WRITTEN BY JODY HANDLEY
PRODUCT MARKETING MANAGER AT GRAPECITY

Mobile Apps, APIs, and Secrets

QUICK VIEW

01. Secrets hidden in mobile apps, whether static or dynamically computed at runtime, can too easily be reverse-engineered.

02. Don't access third-party services from within a mobile app; delegate them to the API server instead.

03. Protect the API server behind an API Gateway and use a mobile app attestation service to control access.

BY PAULO RENATO
DEVELOPER ADVOCATE

With consumers now preferring mobile apps on their portable devices rather than websites on their laptops/desktops, mobile app development is an essential component in any enterprise software product lifecycle. While websites don't necessarily need APIs to serve their content, APIs are needed in the majority of the use cases for mobile apps. This adds a new layer of complexity in terms of software development, deployment, maintenance, and, as will be explained in this article, security.

Before APIs existed, in order to get data from third-party websites, the approach was to scrape the website HTML — producing unstructured data through a process that was both complex and prone to error. Helpfully, for the scrapers, modern APIs have a set of endpoints that return structured data, making its unauthorized collection and reuse so much easier.

Let's now take a look at how to keep a mobile app secure and free of risky secrets while guaranteeing that the API backend server knows with certainty that it is communicating with a genuine instance of the mobile app, even when the app is running in compromised environments, such as on rooted and jailbroken devices.

SECURING THE COMMUNICATION CHANNEL

First up, the mobile app must only communicate over an HTTPS secure channel to protect the confidentiality, integrity, and authentication of the requests from man-in-the-middle-attacks.

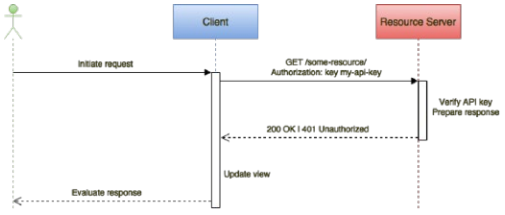
While the inexperienced developer may imagine that this is sufficient to secure communications between a mobile and an API server to keep all sensitive information away from attackers, experience tells us that these protocols are vulnerable. Older SSL or TLS 1.0 implementations have been deprecated due to recent attacks against them, such as the Poodle attack, that in the end allows an attacker to gain access to supposedly protected data.

Even with the use of more secure protocols, an attacker can perform man-in-the-middle attacks against the API server and a mobile app installed in a device he controls in order to be able to extract all secrets and understand how the app queries the API server and what responses he gets back.

For further security, Certificate Pinning should be used so that the mobile app and the API server know exactly which certificate public keys to expect from each other, protecting themselves from rogue certificates provided by man in the middle attacks used to decrypt the data in transit. Unfortunately, even Certificate Pinning can be bypassed with the use of instrumentation frameworks, so we must look further for a more complete solution.

MOBILE APP IDENTIFICATION

The most common way for a mobile app to identify itself to the API server is with the use of an API Key, a unique ID string present in the headers of each API call or as a query parameter in the URL. The latter method is not recommended since it may be logged in any server that handles the URL and then extracted.

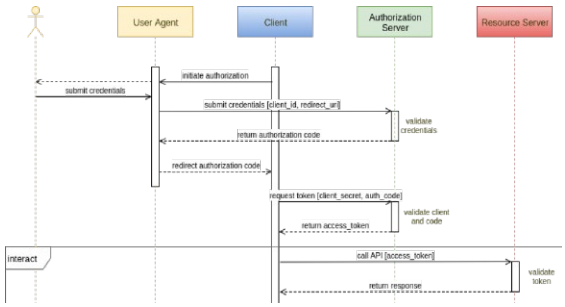


Note: Read the client as the mobile app and the resource server as the API server.

API keys help identify the origin of a request and can be used to rate-limit the frequency of API calls. They provide all users of the app with the same set of access permissions and call statistics gathering. Typically, though, API keys are static secrets within an app which are vulnerable to discovery and abuse, and blacklisting a compromised key will disable not just one but all installed app instances.

USER AUTHENTICATION

The most common user identification approach is to use the OAUTH 2 authorization flow that provides an access token, also known as a bearer token, in the form of an JWT token, consisting of a JSON payload containing a set of claims with an expiration time. The token must be signed by the OAUTH 2 provider with a secret that is shared with the API server. Per best practices, it should be passed in the header of each HTTPS request so that the API server can validate the signature and check that the claims have not expired.



Note: Read user agent as a browser, client as the mobile app, authorization server as the OAUTH2 provider, and resource server as the API server.

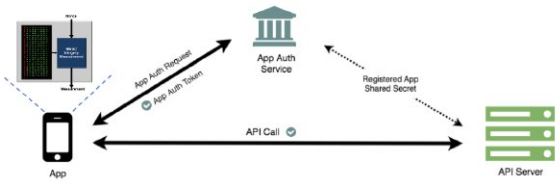
User access tokens often have long lifetimes — hours, days, or even weeks in order to improve the user experience. This makes life easier for attackers to steal and reuse them frequently, until they are detected and revoked.

Though using a user OAUTH 2 access token appears to be an improvement over using an API key, it still leaves a security risk within the mobile app, API, and backend server ecosystem. Simply put, it does not resolve the problem of the API server responding to requests from a cloned or tampered mobile app or from an automated script or bot. Additional techniques such as mobile app attestation are needed.

MOBILE APP ATTESTATION

A mobile app attestation service uses unique characteristics of the mobile app to attest its integrity and authenticity. An example of a unique characteristic might be a simple hash of the application package. The integrity of this simple attestation depends on the integrity of the hashing computation. Such a simple scheme might be fairly easy to spoof.

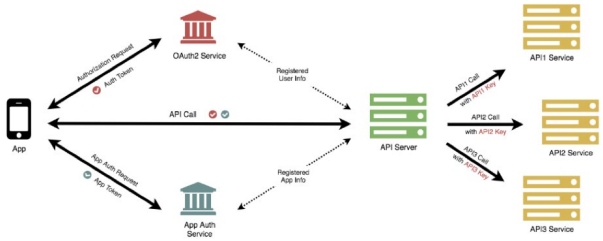
A more robust integrity attestation service might use a random set of high-coverage challenges to detect any mobile app replacement, tampering, or signature replay. If the responses satisfy the challenges, the mobile app attestation service returns to the mobile app a very time-limited integrity token signed by a secret that is known only by the attestation service and the API server. On failure, the mobile app attestation service will sign the integrity token with a random secret that is not known to the API server so that the server can distinguish between an integrity token that passed the challenges from one that failed to pass them or from one that was forged by an attacker while trying to reverse engineer the mobile app.



This means that with every API call, both the mobile app attestation token and the OAUTH 2 access token must be sent in the request headers in order for the API server to be able to validate both tokens, before handling the request.

Unlike the user authentication flow, the mobile app attestation service doesn't require user interaction. Thus, the token lifetimes can be extremely short, and no refresh tokens are needed as in a user authentication flow.

Crucially, once the mobile app attestation service has provided the integrity token that makes it possible for the API server to distinguish requests coming from a genuine mobile app, we have now effectively protected the OAUTH 2 access token from unauthorized reuse. We could also go further and remove all secrets in the mobile app and instead delegate to the API server the authentication of all calls to third-party APIs.



MOBILE APP SECRETS

When the need to use secrets in a mobile app to access an API server or third-party service arises, those secrets commonly appear as plain text in the source code. More advanced developers use code obfuscation techniques to hide them while others calculate the required secrets dynamically at runtime and apply code obfuscation techniques to hide the related source code.

It should come as no surprise to anyone that any secrets or decision-making code that lives on the mobile device is at risk. In fact, any secret stored statically or calculated dynamically in a mobile app can

be reverse-engineered by using techniques ranging from proxy tools to instrumentation frameworks to code decompilers.

While iOS developers may use the Keychain Services API to store the secrets in the encrypted database, Android developers will encrypt the secrets and save them in SharedPreferences, using at the same time the Android Keystore to keep the encryption keys safe. However, as described above, both iOS and Android solutions can be reverse-engineered at runtime.

A good approach is to delegate access management to an API Gateway of all third-party services that a mobile app needs to work with, thus removing the need for the app to handle all these different secrets. However, this still leaves the mobile app needing to store and present the required secret expected by the API Gateway; thus, an obvious exploit remains for attackers.

Considering the mobile app attestation service described earlier, it should be clear that employing such a technique will remove the need to have secrets in the mobile app source code or, if they are necessary for other reasons, at least to protect them more effectively.

REVERSE-ENGINEERING

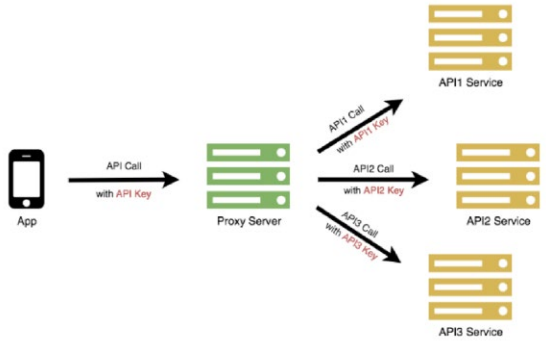
The best way to know how to properly secure a mobile app is to think like an attacker would. The first step in that direction is to know what tools exist and how they can be used to reverse-engineer a mobile app to gain valuable insights on what the app's security strengths and weaknesses are.

Reverse-engineering may seem like a daunting task, one that is only achievable by a few, but, in fact, it is readily achievable by "normal" engineers due to the proliferation of open-source tools which were initially aimed at helping security researchers, pentesters, and white hats to automate their tasks.

The use of a set of combined tools (like Apktool, Dex2Jar, and JD-GUI for Android or Clutch in conjunction with IPA or a Hopper disassembler for iOS) will allow an attacker to extract the source code from any mobile app binary for further analysis using a tool like TruffleHog to identify all secrets in the source code.

With a proxy tool like MiTMproxy, an attacker can intercept, observe, and manipulate the traffic between the mobile app and the API server in order to see how requests and responses are made, how they are structured, and what secrets are used.

If using an MitM proxy is not enough, then an instrumentation framework like Frida or Xposed will allow an attacker to run the mobile app on a device or emulator to intercept system or method calls and/or manipulate the memory of the device in order to understand how the code works, bypass security checks, or extract whatever sensitive data the attacker is interested in.



An all-in-one solution for reverse-engineering exists by the name of Mobile Security Framework. It only requires the mobile app binary to be uploaded — and then the process of extracting the source code, finding secrets, and potential security vulnerabilities is automated, and a very detailed report is given in the end. This tool can also be used while the mobile app is running to perform a dynamic analysis of what the code is doing and what traffic is going in and out, just like you would do with a discrete set of tools.

SUMMARY

With the increase in mobile app deployment and the common practice of embedding secrets in the source code to access in-house and third-party APIs, mobile apps and their APIs have become an attractive target for attackers and a liability for the businesses that they represent in terms of security and attack surface.

The use of a secure communication channel in conjunction with mobile app attestation is a solid and complete solution to reduce the attack surface and strengthen the security between a mobile app and an API server, because if you're using only a secure communication channel with an API key, user authentication and code obfuscation techniques will not prevent the API server from being abused by other sources.

So, the takeaway is to develop mobile apps with security in mind from day one, minimizing the use of embedded secrets and delegating access management to an API Gateway as needed. While mobile attestation techniques significantly improve app and API security, the developer should continuously conduct reverse engineering tests to probe the security and resilience of the code he writes following practices recommended in the OWASP Mobile Security Testing Guide.

PAULO RENATO is known more often than not as paranoid about security. He strongly believes that all software should be secure by default. He thinks security should always be opt-out instead of opt-in and be treated as a first-class citizen in the software development cycle instead of an afterthought when the product is about to be finished or released.

diving deeper

INTO DYNAMIC WEB & MOBILE APPLICATION 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.

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.

Security dzone.com/security
The Security Zone covers topics related to securing applications and the machines that run them. The goal of the Zone is to help prepare the people who make and support those applications stay up to date on industry best practices, remain aware of new and omnipresent threats, and help them to think "security-first."

twitter


@umaar


@AddyOsmani


@SaraSoueidan


@Swizec


@sophiebits


@plavookac


@sarah_edo


@leisa


@rauschma


@rachelandrew

refcardz

Essential Liferay
Includes web content management, workflow, administration, hot tips, and more.

Node.js
This updated Refcard introduces Node, explains how it works, and dives into its architecture. Learn how to install and use Node to its full potential and access a list of the most commonly used APIs.

Introduction to Web Components
Download this Refcard to learn more about web components. Java Champion Kito Mann walks you through the process from set up to using helpful libraries.

podcasts

JS Party
Recording live every Thursday, JS Party explores JavaScript and the web in general.

Shop Talk Show
This self-proclaimed "internet show about the internet" will teach you about various web development topics through interviews with people working in web dev.

CodePen Radio
Learn about a wide range of topics related to web dev with episodes about debugging, feature testing, serverless, and more.

books

PHP and MySQL Web Development
Learn how to use PHP and MySQL to produce web applications, how to work with a MySQL database, and how to use PHP to interact with your database and server.

JavaScript and JQuery: Interactive Front-End Web Development
Learn about basic programming concepts, core elements of the JavaScript language, JQuery, and more.

Android Programming: The Big Nerd Ranch Guide
Learn how to use hands-on example apps in Android studio to create apps that can integrate with other apps, download and display pictures from the web play sounds, and more.

Don't Let Poor UI Leave Your Customers in the Dark.

outsystems

Light Their Way
With a Modern UI.

If opening an app sends users back to the stone age, it doesn't matter how useful your application is. Putting your users first is one small step for you, one giant leap for them.

See it work at
outsystems.com/ui

SPOTLIGHT: User Interface and Experience

Someone might hand me the baseball cap of my dreams, but if it's too tight and it hurts to wear it, it'll just end up at the back of my closet. The same principle applies to app dev. Even if it's the most beautiful app I've ever seen, if the functionality isn't there, what's the point?

What I'm trying to say is that a modern, functional user experience is key to keeping customers happy when they're using your app. Plus, I'll let you in on a little secret. We're all hardwired to want to be comfortable. So, if an app is familiar and easy to use, you're offering users comfort.

That means thinking about what your users want to achieve, how they will interact with your app while you're designing it, and creating a set of screens and functions that provide them with the best and fastest possible way to do so.

As you can imagine or even know first hand, this can mean a lot of work (JavaScript frameworks! CSS!) and a lot to track. Wouldn't it be easier if someone handled all the elements like menus and camera functionality for you, and all you needed to do was focus on building the best user experience possible?

With low-code, it is that easy. Developing on a low-code platform takes care of the nuts and bolts, like navigation and external hardware integration, so you can focus on delivering an excellent UX instead. What's more, at OutSystems, we took the time to analyze hundreds of popular apps, found out what UI features made them great, and built UI templates right into our platform, saving you time and stress.

WRITTEN BY MIKE HUGHES
PRODUCT MARKETING, OUTSYSTEMS

PARTNER SPOTLIGHT

OutSystems Low-Code Development Platform

OutSystems is the #1 low-code platform for digital transformation—build mobile apps, web portals, mission-critical systems, and more.

outsystems

CATEGORY

High productivity, low-code application development and delivery platforms.

RELEASE SCHEDULE

Annual major releases and monthly feature releases.

OPEN SOURCE?

No, but we offer a free personal edition

CASE STUDY

The City of Las Vegas wanted to rapidly develop user-centric applications for its citizens and staff. They restructured and chose OutSystems to maximize the value of their modernized ecosystem. Two developers completed the first proof of concept, a Mobile Inspector App with offline capability, in three months. It seamlessly integrates with the back end and everything on screen is in real time. Because OutSystems streamlines development of features, like offline capability, the city could focus on the business API layer to support more apps in the future. The new app reduces inspection times and eliminates the need for frequent trips back into the office, and the city is now poised to build a variety of internal and citizen-facing applications.

STRENGTHS

1. Visual, low-code full-stack web and mobile application development
2. Integration with everything: Pre-built connectors to leading AI/ML and IoT platforms
3. Over 100 built-in UI patterns and screens
4. One-click deployment and continuous delivery for even the most complex apps
5. High availability, scalability, and enterprise-grade security for running apps

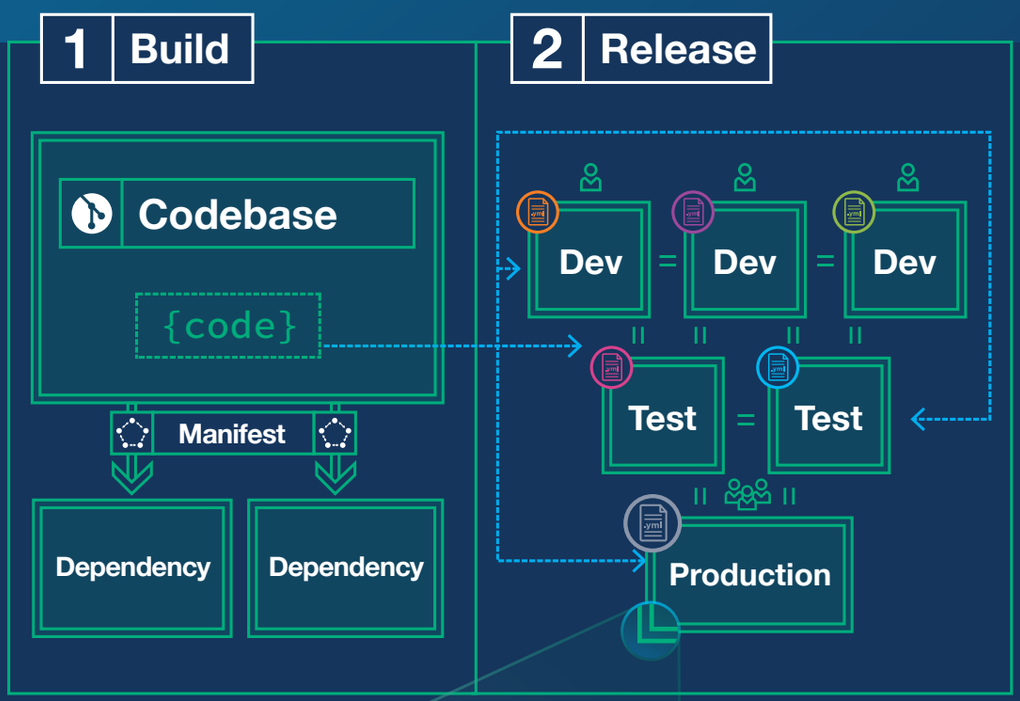
NOTABLE USERS

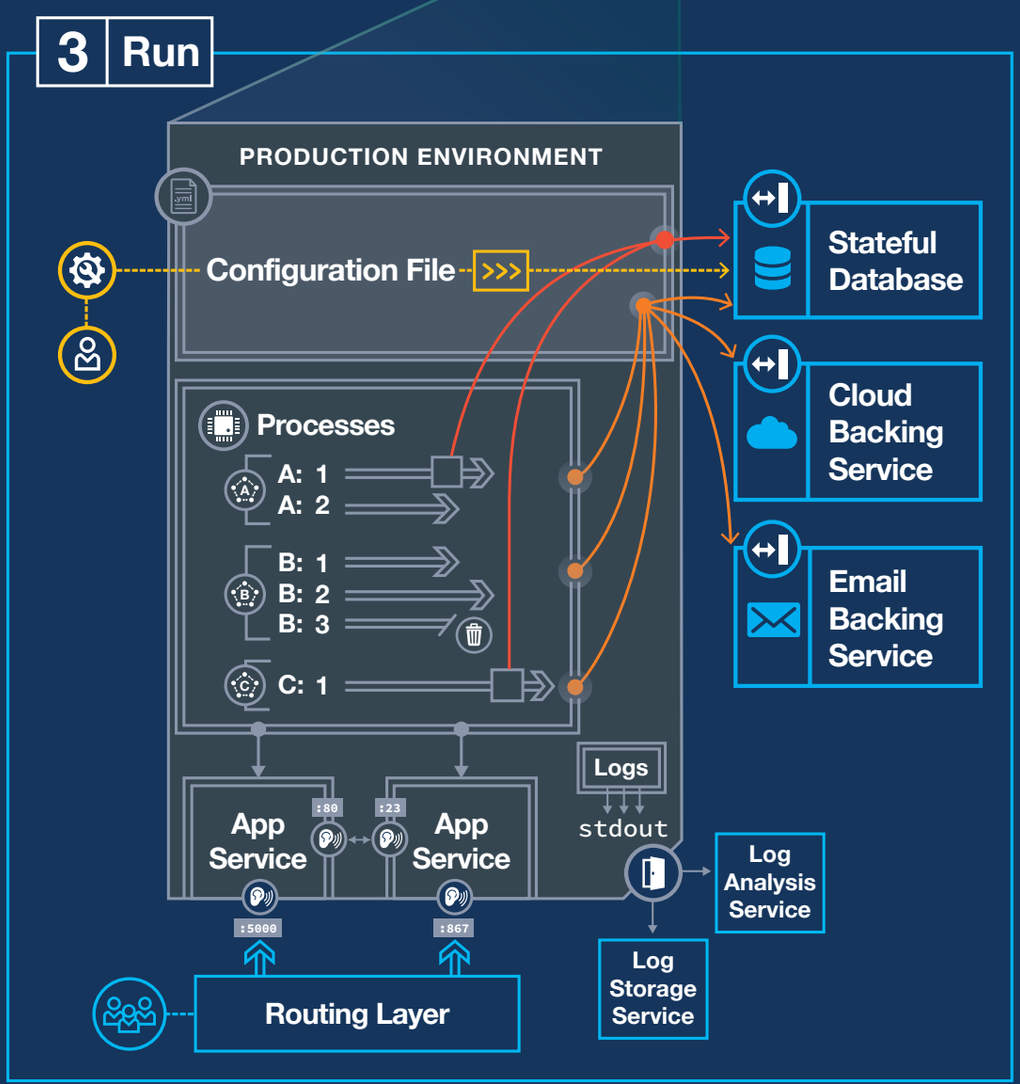
- FICO
- AXA Insurance
- Vodafone
- Toyota
- Randstad

WEBSITE outsystems.com | **TWITTER** @ OutSystems | **BLOG** outsystems.com/blog

The 12-Factor App

Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.





the 12 factors

Codebase
One codebase, many deploys, strict version control

Dependencies
Explicitly declare and isolate dependencies

Configuration
Store config in each deploy environment, preferably using environmental variables

Backing Services
Treat backing services as resources (neutral as to local vs. third-party) located via config

Build, Release, Run
Strictly separate build, release, and run; never change code at runtime

Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service

Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection

Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently

Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)

Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)

Logs
Log by writing all output streams to stdout; rout streams using non-app services

Admin Processes
Run one-off/admin processes (db migration, REPL , one-time scripts) in same environment as normal processes

The Pain of Creating Web-Ready Media Files

We live in a highly visual, easily-distractible world, and expect high-performance, media-rich content to be delivered to us on a variety of channels and devices. This presents a number of complex challenges for developers charged with taking raw media files and delivering the fast and engaging experiences consumers have come to expect.

Delivering "web-ready" files is time-consuming and cumbersome, involving numerous variables and multiple stages in the process. Key considerations for delivering web-ready media files include:

1. Ensuring the proper layout and resolution of images and videos are delivered to users' devices, which vary considerably in form factor and device-pixel ratios.
2. Optimizing files to achieve the smallest file size possible without degrading perceived quality.
3. Delivering content to a global audience with efficient caching rules to match local performance challenges.

Many developers address these challenges through a multi-step manual process that involves uploading a file, cropping and resizing it to fit various resolutions, saving each cropped version as its own file, transcoding those into multiple formats, and implementing backend logic to assign the right image to the right end-user based on their device, screen size, and browser. Running through this workflow for each image or video might be considered an inconvenient annoyance at first, and quickly becomes agony at scale.

However, effective and efficient solutions, like Cloudinary, automate the end-to-end media management process and streamline the entire production workflow. Learn more about how Cloudinary helped Fairfax Media drive digital transformation and simplify their media management workflow.

WRITTEN BY DORON SHERMAN
VP DEVELOPER RELATIONS, CLOUDINARY

PARTNER SPOTLIGHT

Cloudinary

The Media Full Stack: Heavy-duty Image & Video Platform

Cloudinary

CATEGORY
Image and Video Management

RELEASE SCHEDULE
Continuous

OPEN SOURCE?
No

CASE STUDY

UA Record is Under Armour's health and fitness app. It enables users to sync a wide array of data, access original content, and share user generated content, such as videos, photos, and workout stories. The team sought an easy-to-use solution that allowed them to host a large quantity of images and videos and simplify image sizing and transformation.

After implementing Cloudinary, Under Armour was able to quickly take the solution into production, helping them speed up development cycles and improve user experience.

"Cloudinary technology is meeting Under Armour's needs to deliver a great user experience, while hosting an increasingly large number of images and scaling to meet the explosive demand for its fitness apps." - Hanifen, Product Manager at Under Armour

STRENGTHS

1. Automates responsive & optimized image creation
2. Dynamic media manipulation via URL
3. Multi-CDN delivery options
4. Robust APIs, integrations & partner ecosystem

NOTABLE USERS

- AMC
- Forbes
- StubHub
- Under Armour
- Whole Foods Market

WEBSITE cloudinary.com | **TWITTER** @Cloudinary | **BLOG** cloudinary.com/blog

Security Best Practices for Managing Mobile API Access Tokens

BY ISABELLE MAUNY
CTO, **42CRUNCH**

QUICK VIEW

01. Dive into why access tokens are a fundamental part of API security.

02. Learn why API providers must provide access to security tokens according to the API's associated risk.

03. See how API developers must handle tokens with care, as if they were passwords.

Modern applications, both web-based and native, rely on APIs on the backend to access protected resources. To authorize access to those APIs, a request must include some kind of access token or key. This article focuses on security best practices for access token management — for API providers and application developers alike.

LET'S TALK ABOUT TRUST FIRST!

When dealing with security, a single rule prevails: *trust no one*. If you're an API provider, you can't trust that the application invoking the APIs is the one you expect, that the token you received has not been stolen, or that the communication between client and the server has not been intercepted. On the client side, you can't trust that the application will not be decompiled (exposing embedded secrets), that the application storage will not be compromised through an XSS attack, or that your users are not being fooled into submitting forged requests.

This implies that you must put into place proper measures to securely obtain, store, and manage the security tokens required to invoke backend APIs.

Additionally, you may think your APIs are safe if you have never publicly advertised them. To you, they feel private because they are only used by your enterprise applications. However, if they can be used from a mobile application, they are on the public internet, and thus public. Any API exposed outside your enterprise network must be considered public.

OBTAINING TOKENS AND API KEYS

When it comes to using an API, you are usually offered two choices: pass a static piece of information together with the API call *or* obtain that piece of information dynamically prior to invoking the API. This piece of information is usually an access token or API key. BasicAuth is still used for some APIs for legacy reasons but is deprecated as a mainstream solution.

When designing the security aspects of your API, you must choose wisely how your API consumers access it. As per usual with security measures, the induced risk is the key factor to take into account. Securing an API that only allows for consulting weather data is very different from securing a banking payments API.

While using an API key is easier for the developer, it does not give the same level of security as an access token obtained with two-factor user authentication and the proper identification of the client application. Moreover, an API key does not carry any information about the user and can't be used at the backend level

to decide which operations the API consumer is allowed to invoke. Finally, API keys never expire unless revoked by the API provider.

OAuth was created to address these drawbacks:

- The application accessing the resource is known (using client application credentials).
- The API provider can define scopes to limit the access to certain operations (you can GET a catalog entry, but you can't PUT a new catalog entry, even with a valid token).
- Tokens have a limited lifetime.

LET'S START WITH SOME TERMINOLOGY

The OAuth terminology can sometimes be confusing. In the table below, we present a mapping from practical, development-focused terminology to OAuth terminology.

PRACTICAL NAME	OAUTH TERMINOLOGY	DESCRIPTION
Application	Client	The client is the application accessing a resource on behalf of a user.
Web server apps	Confidential client	An application running on the server side and capable of safely storing an application secret.
Single-page apps/browser-based apps/ mobile apps	Public client	An application entirely running on the client side or on a device that cannot safely store an application secret.
API	Resource server	The API is the means to access the resources belonging to the user (e.g. a bank account).
OAuth server	Authorization server	The OAuth server is in charge of processing the OAuth token management requests (authorize access, issue tokens, revoke tokens).
User	Resource owner	The person granting access to the resource the application is trying to access.
Access token	Bearer token	The access token authorizes the application to access the API.

OPAQUE VS. JWT

OAuth does not mandate the access token format, and as such, depending on the OAuth server implementation, the access token could be opaque (typically a long string carrying no information) or a JSON web token (JWT).

The key advantage with JWTs is the ability to contain claims, or information about the user, which the backend services can use to make business logic decisions.

LEARNING THE OAUTH DANCE

OAuth grant types define how a client can obtain a token. Our team commonly refers to this as the "OAuth dance." There are many ways to dance in the OAuth world, but there is only one you *must* learn: authorization code. Other grant types can be useful in some circumstances, but the authorization code grant type is the recommended way to obtain an access token for all types of applications: web apps, native apps, and mobile apps.

For public clients and mobile apps in particular, an additional security measure is recommended to prevent the theft of the authorization code. This security layer is described in the Proof Key for Code Exchange standard (PKCE, pronounced "pixy"). You can learn more about PKCE and how to use it here. If you are an API provider, make sure your OAuth server supports this option.

You should use special care with the Resource Owner password grant: being the simplest to implement, it's quite attractive. However, since its core requisite is client-server trust, you probably should never use it.

TOKEN MANAGEMENT RECOMMENDATIONS

BEWARE OF OAUTH APP CREDENTIALS LEAKS

Storing your application code in GitHub? Are your OAuth app credentials stored there, as well, and, in particular, the client secret? This is the *number-one source* of credentials leaks today. If those credentials are stolen, anybody can pretend to be you. If you believe credentials could have been compromised, regenerate them immediately.

Additionally, never put your client secret in distributed code, such as apps downloaded through an app store or client-side JavaScript.

DON'T HARDCODE TOKENS IN APPLICATIONS

It can be tempting to simplify code to obtain a token for a long period of time and store it in your application. Don't. Do. That.

TREAT TOKENS AS YOU WOULD TREAT PASSWORDS

Tokens are the door key! Token and API keys allow anybody who has them to access a resource. As such, they are as critical as passwords. Treat them the same way!

OAUTH IS NOT AN AUTHENTICATION PROTOCOL

OAuth is about delegating access to a resource. It is not an authentication protocol (despite the name). Think of tokens as hotel cards. You need to authenticate yourself to obtain the hotel key, but once you have it, in no way does it prove who you are. API providers must not rely on token possession as a proof of identity, as proven by a recent user information leakage.

You really should look at OpenID Connect (OIDC), a complementary specification, rather than trying to implement authentication on top of OAuth yourself. OIDC allows a user to share some aspects of their profile with an application with no need to share their credentials.

BEWARE OF WHAT YOU STORE IN JWTS AND WHO HAS ACCESS TO THEM

JWTs can store plenty of information in the form of claims and can easily be parsed if captured (unless they are encrypted). If you are using JWTs to carry information that is only useful to the backend services, you can take a different approach:

- Use an opaque string or basic JWT between the client and the backend.
- At the backend, validate the request and inject a new JWT with a payload containing the claims that are consumed downstream. This feature is provided by many API security gateways out of the box.

If you want to use the same token across the entire flow and it can potentially carry sensitive information, encrypt the token payload. This said, never use a JWT to carry user's credentials, such as passwords!

VALIDATE JWTS THOROUGHLY

When you receive a JWT on the server side, you must validate its contents thoroughly. In particular, you should reject any JWT that does not conform to the signature algorithm you expected or that uses weak algorithms or weak asymmetric/symmetric keys for signing. Additionally, you must validate all claims, expiration date, issuers, and audience.

Certain libraries and tools do this for you; others need to be configured properly first; some only do partial checks. Take the appropriate measures depending on the libraries you use.

DON'T STORE TOKENS IN LOCAL STORAGE; USE SECURE COOKIES

Browser local storage and session storage can be read from JavaScript, and as such are not secure to store sensitive information such as tokens. Instead, use secure cookies, the `httpOnly` flag, and CSRF measures to prevent tokens from being stolen.

ALWAYS TRANSPORT TOKENS VIA HTTPS AND IN THE REQUEST BODY

In doing so, you are limiting the risk of the token to be captured in flight or to be written into proxy logs or server logs. You should also ensure you only use TLS 1.2/1.3 and the most secure cipher suites across all actors involved in issuing and validating tokens.

USE A DEDICATED BROWSER VIEW TO REQUEST CONSENT INFORMATION

Many applications use an embedded user agent, but this should be avoided because it does not allow the user to properly validate to which site they are talking to. Moreover, the app would have full visibility of the credentials the user enters. Some API providers take strong measures against this practice, as encouraged by the OAuth native apps best practices standard.

CONCLUSION

Access tokens are fundamental to the implementation of modern applications — but must be handled with care. As a backend developer, you must ensure you provide the proper grant type to obtain access tokens, support PKCE for mobile apps, and thoroughly validate JWTs. As a frontend developer, you must control the storage of JWTs and secure app credentials.

REFERENCES

- PKCE
- JWT Validation Best Practices
- Native Apps Best Practices OAuth

ISABELLE MAUNY has been solving integration problems for as long as she can remember: at IBM for more than 15 years, then at Vordel (now Axway), then at WSO2, and now at 42Crunch — a company she co-founded late 2016. She worked in pre-sales and implementation services at customers across the globe and has been focusing more on product management and delivery for the past ten years. Along her career, she has worked with a myriad of products, such as application servers, portals, BPMs, XML security gateways, ESBs, API management, and now security firewalls. If you want to know a bit more, check out her website.

What Developers Need to Know: 3 Tips for Effective Mobile Testing

BY CLINT SPRAUVE
MOBILE APPLICATION TESTING EVANGELIST AT **SAUCE LABS**

QUICK VIEW

01. More than half of all web traffic now comes from mobile devices, so mobile testing is critical.

02. Testing is no longer solely the purview of QA, so developers must be equipped to develop effective mobile test strategies.

03. Start by choosing a mobile framework that is best suited for your development and testing pipeline as well as application type.

04. For the best coverage at the best price, test on both emulators/simulators and real devices.

These days, developers are handling an increasing share of the software testing workload. This is a good thing, as it means testing is taking place earlier and quality is being owned by more of the organization. And with people accessing the internet on mobile devices far more than traditional desktops, it's essential that developers are equipped to create and execute an effective mobile testing strategy. Gartner has estimated that this year, more than 50 percent of users will go to a tablet or smartphone first for all online activities, so software development today must have mobile in mind from the start. However, developers aren't always prepared to take a strategic approach to testing: they're accustomed to writing code, not testing it. This article offers a crash course in creating an effective mobile test strategy for developers who need to get up to speed and make decisions quickly.

Here, we'll explore three specific, practical areas needed for developers to get started on a mobile test strategy. First, we'll cover how to choose an automated testing framework. Second, we will review the pros and cons of testing using emulators and simulators versus real devices. And finally, we offer suggestions on how to decide which tests to automate — and which to continue testing manually.

TIP #1: HOW TO CHOOSE AN AUTOMATED TESTING FRAMEWORK

There are several frameworks to choose from when it comes to automated testing. Choosing one can be intimidating, but it's an important first step.

Selenium is the industry standard and is a widely-used framework. It's a great choice if you need to automate tests across a variety of browser and operating system combinations. But for mobile testing in particular, you'll likely want to use Appium, Selenium's mobile cousin. You should also consider native frameworks, such as XCUITest for iOS and Espresso for Android.

It's important to remember that you don't just have to choose one. You should have multiple frameworks to choose from, because some lend themselves better to specific use cases.

Speaking of use cases, before you choose a mobile test automation framework, you must understand what you need to test. Here is a checklist that can act as a starting point for determining your mobile testing requirements:

- Which mobile platforms do you need to test on? Are you looking at iOS, Android, or both? Also, which versions are the most important?
- Is your app native, hybrid, or mobile web? If it's mobile web, which browsers (and versions) are the most important for testing?
- Which elements of your app are you most concerned with testing? GUI? Memory and resource use? Load and stress? Basic functionality? All of these?
- Are there any specialized/unique features that you need to test? Do you have a clear strategy for testing them?
- Do you need code-level testing? Black-box, white-box, or gray-box?
- Approximately how many different tests (within an order of magnitude) do you anticipate running? How many of those can be run in parallel?

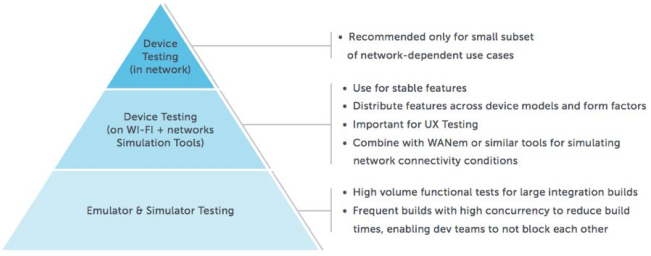
- Which test scripting languages do your test engineers have the most experience with? Do they have a preference (e.g. are they happy with Python and JavaScript, but don't want to touch Ruby or C# — or vice versa)?

These questions offer a great starting point for developers needing to get over the hurdle of determining which frameworks to test on the path to mobile test automation. However, don't feel you have to choose just one. It's a good idea to adopt a test infrastructure that allows you to change it up between different frameworks as needed. Cloud-based testing supports this flexibility. If you set up your test environment on a single framework on your own infrastructure, you will need to make a lot of changes in order to switch frameworks if you find yourself needing to use another. But when you use a public testing cloud that supports multiple frameworks, you're all set to switch back and forth as needed.

TIP #2: TEST ON BOTH EMULATORS/SIMULATORS AND REAL DEVICES

The use of emulators and simulators to test mobile applications has historically been met with resistance. This is based on the perception that if you're not testing on a real device, you're not really testing at all. I beg to differ! Although real devices do give more accurate test results, using them alone is by no means ideal for continuous testing and continuous delivery. Testing with real devices can be costly, too. Due to budget issues, some organizations don't bother at all with real devices as they're too expensive, and instead opt for emulators and simulators.

The reality is that effective and efficient mobile app development requires both emulators/simulators and real devices. An emulator, as the term suggests, emulates the device software and hardware on a desktop PC, or as part of a cloud testing platform. The Android (SDK) emulator is one example. A simulator, on the other hand, delivers a replica of a phone's user interface, and does not represent its hardware. The iOS simulator for Apple devices is one such example. Emulators enable parallel testing in a way that can't be achieved having to manually prepare each emulator for the tests. Further, automation is easier with emulators as the tests can be executed without manual intervention and can be controlled remotely.



A combination of emulators, simulators, and real devices provides the best coverage for all use cases.

The bottom line? Using both emulators/simulators *and* real devices offers the best coverage and performance at a reasonable cost.

TIP #3: DETERMINE WHICH TESTS TO AUTOMATE

Believe it or not, your goal shouldn't be to automate every test. Automated and manual testing are both required for an effective mobile testing strategy. Automated mobile testing can be applied to cumbersome tasks and it's very useful for regression testing at scale, as well as for load tests. To effectively identify which tests are the best candidates for automation and prioritize them accordingly, start with the following questions:

- Which tests will be performed most often? These are often the best ones to automate.
- Which tests involve purely technical elements (such as whether an application starts as expected in a given browser), as opposed to tests that are subjective in nature (such as how users interact with a UI)? Purely technical tests are easy to automate; those that have subjective elements may need to be performed manually in many cases.
- Which tests take the longest to perform manually? Automating the most time-consuming tests will deliver the greatest benefits in efficiency.

Answering these questions will help get you started in the right direction. At SauceCon 2018, Angie Jones presented a great framework on how to choose which tests to automate. Her session offers a thorough, structured plan for determining which tests merit automation.

CONCLUSION

With developers taking on more of the software testing burden, it's important for them to know how to get up to speed quickly with testing. And when developers can add software testing to their list of skills — particularly mobile testing — it makes them more well-rounded and valuable to their organizations. With the tips from this article, developers will be able to plan and execute an effective mobile test strategy.

CLINT SPRAUVE is the mobile application testing evangelist at Sauce Labs. Previously he was the senior product marketing manager for Hewlett-Packard Enterprise functional testing. He was also the director of product strategy for the Borland Solutions Division of Micro Focus, where he created Borland's mobile strategy for functional test automation.

Executive Insights on Web and Mobile Development

QUICK VIEW

01. The keys to developing effective web and mobile applications are: 1) knowing the business problem to solve; 2) UX/UI; and 3) testing.

02. The most significant changes to the development of web and mobile applications has been DevOps and progressive web apps (PWAs).

03. The most frequently mentioned new technical solutions are React, PWAs, and Xamarin.

BY TOM SMITH
RESEARCH ANALYST AT **DEVADA**

To gather insights on the current and future state of web and mobile development we talked to 19 IT executives. Here's who we spoke to:

Respondents:

- Malcolm Ross, Vice President, Product, Appian
- Gil Sever, Co-founder and CEO, Applitools
- Adam Fingerman, Founder and Chief Experience Officer, ArcTouch
- Jon Janego, Senior Product Manager, Static Analysis, CA Veracode
- Doron Sherman, V.P. Evangelism, Cloudinary
- Himanshu Dwivedi, Founder and CEO, Data Theorem
- Doug Dooley, COO, Data Theorem
- Rimantas Benetis, Technology Director, Devbridge Group
- OJ Ngo, CTO and Co-founder, DH2i
- Nate Frechette, CTO and Co-founder, Dropsource
- Will Bernholz, V.P. Marketing, Dropsource
- Anders Wallgren, CTO, Electric Cloud
- Lucas R. Vogel, Principal Consultant, Endpoint Systems
- Sriram Krishnan, Head of Product, Headspin
- Joshua Strebel, CEO and Co-founder, Pagely
- Brad Hart, Vice President of Product Management, Perfecto
- Robert Warmack, Director, Sencha
- Jeffrey Martin, Director of Product Operations, SmartBear
- Eric Sheridan, Chief Scientist, WhiteHat Security

1. The keys to developing effective web and mobile applications are: 1) knowledge of the business requirements; 2) UX/UI; and 3) testing. There must be an alignment of business requirements between the product owner, the business, developers, and testers. Determine what kind of app is needed to solve the business problem before you begin to build the app. Understand the priorities and goals of what you are trying to accomplish.

Web and mobile applications must be designed and developed with a focus on user behaviors from day one. Since web and mobile apps involve a myriad of design choices, the keys to successful development depends on the developers' ability to eliminate complexity. Ease of use is critical. If an app is not reliable and easy to use, it will not be used. Prioritize mobile UX/UI since this has the greater impact on the bottom line, and make sure UI is consistent across all platforms. Today, users expect visual perfection across browsers. Make sure the app will scale, be stable, and be supported over the long-term.

Implement continuous testing, integration, and development that's solid and stable. Automate the process – especially testing. Automation can shorten the testing cycle by 50 to 95%.

2. The most significant changes in the development of web and mobile applications has been DevOps and progressive web apps (PWAs). There has been an increased focus on DevOps across a number of development teams incorporating automation into nearly every stage of application development. Agile and DevOps methodologies encourage faster iteration, deployment, and response.

PWAs are beginning to deliver on the promise of collapsing web and mo-

bile user experiences and the development effort. When combined with development of WebAssembly, the difference between a website and a mobile application is quickly fading.

In addition to PWA, other significant trends in the web and mobile app development include: 1) responsive web design (ensuring proper layout and resolution of content, especially rich media like images and videos, is delivered to users' devices, which vary considerably in form factor and device-pixel ratios); 2) serverless (virtualizing backend logic at a functional level, eliminating the need for server installation/configuration and ensuring unlimited scalability and geographical independence); and 3) streaming (highly interactive content that includes animation, video, and other real-time content elements that change in response to front-end and/or back-end events that increasingly entail use of third-party service providers).

3. The most frequently mentioned new technical solutions are React, PWAs, and Xamarin. React Native in particular is very popular for web applications. PWA is a hot topic and an alternative for content organizations that want to build and launch on mobile. Xamarin is an open-source framework for C# applications that works across platforms.

4. The most common hurdles affecting the development and deployment of web and mobile applications are: 1) the diversity of solutions; 2) sufficient knowledge/skills; 3) security; and 4) delivering business value. There are a multitude of platforms and screen sizes between tablets and phones. There is a high rate of change in the core technologies, making it difficult for organizations to find experts capable of using all of the features available. Because the market is so fragmented, it's difficult to know where to start.

There's continued demand for apps from business and a limited supply of skilled IT professionals to deliver them. DevOps professionals are used to automation; however, security professionals are not. We need to make security the "VP of yes" rather than the "director of no" in order to show a contribution to business value in a timely manner.

As the functionality of web and mobile apps gets more sophisticated, there is increased use of external libraries and software components, as well as integration of third-party services. The resulting hurdles developers are now encountering routinely include: 1) security risks, which necessitate safety audits of imported code; 2) performance implications for UX, due to the increased size of external code embedded within web pages; 3) brittle app builds, due to the number of code dependencies, library versions, and inter-team collaboration; and 4) time spent by developers staying up-to-date on the ever-evolving software tooling, frameworks, and infrastructure they're using to build apps.

5. The future of web and mobile application development is the continued consolidation of technologies, PWAs, and optimization. We're seeing a consolidation of DevOps tools. Technology will merge with more of the web being shown on the browser through the mobile device. PWA is the future where mobile's unique capabilities are moving toward browser OS versions. Mobile performance, quality, and reliability will continue to be optimized.

Web and mobile application development is trending toward a unified development and deployment experience with applications that are increasingly composed instead of coded. In the future, the industry will see a strong trend toward low- or no-code development platforms for many uses of mobile or web applications — especially business apps.

6. Developers need to be open to continuous learning in order to develop effective applications for an ever-changing landscape. As Bill Gates says, "innovate or die." Explore new technologies, continue to learn, be open to what's new and to the change coming with mobile. Have a great understanding of the fundamentals (HTML and CSS) while actively seeking new experiences with new languages (Kotlin, Golang, Scala). UX is more important than ever. Microservices have become as important to learn as object-oriented programming once was.

Developing effective web and mobile apps lacks hard and fast rules or guides. The skills developers need to possess are still rapidly evolving. However, some basic principles are immutable such as choosing tools, frameworks, and languages, as well as infrastructure, that are mature beyond the early excitement phase, and are already enjoying broad and active community support. If a developer has a question or runs into a problem, there is a greater chance they'll find an answer or solution when there's a sizeable community available to assist.

7. It's a mobile-first, API-first world. Web developers tend to be from marketing websites, while mobile developers tend to be from IT. They think differently and have different skillsets. Mobile development is taking a more dominant approach to web app design and development. The look and feel are still fundamentally different in web and mobile development, as web is still mostly tied to HTML and CSS. Additionally, the lack of effective cross-training of developer skills leads to separate delivery teams.

Web and mobile development are still largely considered different disciplines with organization priorities dictating which type of development takes precedence on a project-by-project basis. Mobile development is typically done using native languages, frameworks, and tools that are particular to the target devices/OS (most commonly iOS and Android). Development team boundaries naturally segment based on their skill sets and design preferences (e.g. mobile-first app design). Some organizations use cross-platform tools and frameworks, while others opt for unifying platforms (e.g. low-code platforms) to eliminate these boundaries and bridge the differences in design, development, and deployment between these otherwise separate disciplines.

TOM SMITH is a Research Analyst at Devada 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
Android Studio	Android Studio	Mobile app building platform	Free tool	developer.android.com/studio
Angular	Angular	Web app development framework	Open source	angular.io
Apache Software Foundation	Apache Cordova	Hybrid web development platform	Open source	cordova.apache.org
Apple	Xcode 10	App development on Apple platforms	Open source	developer.apple.com/xcode
Aryaka	SmartCDN	Web app acceleration	Available by request	aryaka.com/services/web-app-acceleration
Backtrace	Backtrace	Debugging platform	Available by request	backtrace.io
Blue Spire	Aurelia	JavaScript framework	Open source	aurelia.io

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Browserling	Browserify	Build script tool & package manager	Open source	browserify.org
BrowserStack	BrowserStack	Web testing	Available by request	browserstack.com
Bugsnag	Bugsnag	Application stability monitoring	14 days	bugsnag.com
Cake Foundation	CakePHP	PHP web framework	Open source	cakephp.org
ClojureScript	ClojureScript	Compiler	Open source	clojurescript.org
CloudFlare	CloudFlare	JavaScript CDN	Free tier available	cloudflare.com/cdn
Cloudinary	Cloudinary	Media Full Stack	Free tier available	cloudinary.com
CommonJS	CommonJS	Package manager	Open source	commonjs.org
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	Available by request	engineyard.com
Facebook	React.js	JavaScript library	Open source	facebook.github.io/react
Facebook	Flow	Static type checker	Open source	flowtype.org
GNU	Make	Build script tool	Open source	gnu.org/software/make

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Google	Google Cloud Platform	PaaS	Available by request	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
GrapeCity	SpreadJS	JavaScript spreadsheet components	30 days	grapecity.com/en/spreadjs
GrapeCity	Wijmo	J avaScript UI Controls	30 days	grapecity.com/en/wijmo
Grunt	Grunt	Build script tool, package manager	Open source	gruntjs.com
Gulp	Gulp	Build script tool	Open source	gulpjs.com
hapi	hapi	JavaScript framework	Open source	hapijs.com
Hewlett Packard Enterprise	HPE Hybrid Cloud Solutions	PaaS	Available by request	hpe.com/us/en/solutions/cloud.html
IBM	Bluemix	PaaS	Free tier available	ibm.com/bluemix
IBM	LoopBack	Node.js API framework	Open source	loopback.io
IBM	IBM MobileFirst for iOS	MADP	Available by request	ibm.com/mobile/mobilefirst-for-ios
Infragistics	indigo.design	Visual design, UX prototyping, code generation, app dev	Free tier available	infragistics.com/products/indigo-design
Intel	Crosswalk	Web runtime	Open source	crosswalk-project.org
Jelastic	Jelastic	PaaS	Available by request	jelastic.com
JetBrains	PHPStorm	PHP IDE	30 days	jetbrains.com/phpstorm
JetBrains	PyCharm	Python IDE	Free tier available	jetbrains.com/pycharm
JetBrains	Resharper	Debugging platform	30 days	jetbrains.com/resharper

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
JetBrains	Rider	.NET IDE	30 days	jetbrains.com/rider
JetBrains	RubyMine	Ruby and Rails IDE	30 days	jetbrains.com/ruby
JetBrains	WebStorm	JavaScript IDE	30 days	jetbrains.com/webstorm
jQuery Foundation	jQuery	JavaScript library	Open source	jquery.com
jQuery Foundation	jQuery CDN	JavaScript CDN	Open source	code.jquery.com
jQuery Foundation	jQuery Mobile	Web framework, content-centric	Open source	jquerymobile.com
jQuery Foundation	QUnit	Web testing	Open source	qunitjs.com
Laravel	Laravel	PHP web framework	Open source	laravel.com
LaunchDarkly	LaunchDarkly	Feature management	30 days	launchdarkly.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	Free tier available	mendix.com
Meteor Development Group	Meteor	JavaScript framework	Open source	meteor.com
Micro Focus	Unified Functional Testing	Web testing	60 days	software.microfocus.com/en-us/software/uft
Microsoft	Ajax CDN	JavaScript CDN	Open source	docs.microsoft.com/en-us/aspnet/ajax/cdn/overview
Microsoft	ASP.NET	Web framework	Free solution	asp.net
Microsoft	Azure	PaaS	Free tier available for 12 months	azure.microsoft.com
Microsoft	TypeScript	JavaScript superset	Open source	typescriptlang.org

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Microsoft	Xamarin	C# for Android, iOS	Open source	github.com/xamarin
Microsoft	Visual Studio Code	IDE	Open source	code.visualstudio.com
MochaJS	Mocha	Web testing	Open source	mochajs.org
Node.js Foundation	Express	Web framework	Open source	expressjs.com
Node.js Foundation	Koa	Web framework	Open source	koajs.com
Node.js Foundation	Node.js	JavaScript environment	Open source	nodejs.org/en
Nodesource	N|Solid	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
Oracle	Oracle Cloud	PaaS	$300 of free credits for 3,500 hours of usage	cloud.oracle.com
OutSystems	OutSystems	PaaS	Free tier available	outsystems.com
Perfecto	Perfecto Mobile	Mobile test automation, performance testing w/real devices	240 minutes for 21 days	perfectomobile.com
Pivotal Labs	Jasmine	Web testing	Open source	jasmine.github.io
Pivotal Software	Spring	Java web framework	Open source	spring.io
Polymer Authors	Polymer	Web components library	Open source	polymer-project.org
Progress Software	NativeScript	Cross-platform mobile framework (HTML5->native)	Open source	nativescript.org
Progress Software	OpenEdge	PaaS	60 days	progress.com/openedge

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Qlik	Qlik Playground	Web app testing environment	Open source	playground.qlik.com
QuickBase, Inc.	QuickBase	PaaS	30 days	quickbase.com
Rainforest QA	Rainforest	Mobile test platform (functional & regression)	Demo available by request	rainforestqa.com
Raygun	Raygun	Real user monitoring	14 days	raygun.com/products/real-user-monitoring
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
Rollbar	Rollbar	Error tracking & reporting	14 days	rollbar.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	30 days	hcp.sap.com
Sass	Sass	CSS preprocessor	Open source	sass-lang.com
Sauce Labs	Sauce	Web testing	14 days	saucelabs.com
Sencha	Ext JS	JavaScript framework	30 days	sencha.com/products/extjs
SensioLabs	Symfony	PHP web framework	Open source	symfony.com
StackPath	SecureCDN	JavaScript CDN	30 days	stackpath.com

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Stylus	Stylus	CSS preprocessor	Open source	stylus-lang.com
Swisscom	Swisscom Application Cloud	PaaS	N/A	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
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 version available	vaadin.com
Verizon Digital Media Services	Edgecast CDN	CDN	30 days	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
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
Wordpress	Automattic	CMS	Free tier available	automattic.com
wzrd.in	wzrd.in	Browserify-as-a-Service	Open source	wzrd.in

GLOSSARY

ACCESS TOKEN: Authorizes the application to access the API.

ANGULAR: A platform for building HTML & JavaScript web applications, led by Google & based on TypeScript; a rewrite of the AngularJS JavaScript framework.

APPIUM: An open-source test automation framework for use with native, hybrid, & mobile web apps; drives iOS, Android, & Windows apps using the WebDriver protocol.

COROUTINE: Coroutines simplify asynchronous programming by putting the complications into libraries. The logic of the program can be expressed sequentially in a coroutine & 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 & XML documents as program objects to help control who can modify the document.

EMULATOR: Emulates device software & hardware on a desktop PC or as part of a cloud testing platform (e.g. the Android SDK emulator).

ESPRESSO: A native testing framework for Android.

HYBRID APPLICATION: A mobile app written in HTML, CSS, & JavaScript that uses a web-to-native abstraction layer, allowing the app to acess mobile devices APIs that pure web abbs cannot.

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 web dev for both client-side and server-side scripting. It is easier & faster to code than compiled languages but takes longer to run.

JSON: JavaScript Object Notation; a language-independent textual data interchange format 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 that are deployed separately & communicate with each other.

MQTT: A lightweight messaging protocol that provides network clients with a way to distribute information.

MVC: Model-view-controller; a way of relating the UI to underlying data models that lets developers reuse object code & reduce time spent developing apps with UIs.

NATIVE APP: A mobile app written in a programming language that is directly compatible with the target platform.

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 & server side. It is event-based, nonblocking, & asynchronous.

OAUTH: A standard for website security that allows users to grant websites or apps access to information on other websites without handing over the passwords.

REACT: A library in JavaScript used to build user interfaces.

SIMULATOR: Delivers a replica of a phone user's interface & does not represent its hardware (e.g. the iOS simulator for Apple devices).

SINGLE-PAGE APPLICATION: A web app that responds to user actions & 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 app.

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

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

WEB APPLICATION: A mobile app developed using web standards & accessed through a browser.

WEB SERVER APP: An app running on the server side and capable of safely storing an app secret.

XCIOTEST: A native testing framework for iOS.

INTRODUCING THE

Open Source Zone

Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

Visit the Zone

BROUGHT TO YOU IN PARTNERSHIP WITH Flexera