Trigger.io

White paper

May 2012

Mobile applications and the mobile web represent a huge opportunity for developers. But there are challenges for web developers looking to access the $15bn app market. Trigger's development framework, Forge, enables web developers for mobile by combining HTML5 technology with native power and distribution.

Section 1

Introduction

Rapid growth of smartphones, rapid growth of mobile apps

Smartphones are now one of the most indispensable technological devices an individual can own. comScore's November 2011 Mobile Report highlights the smartphone surge by stating that 91.4 million people in the US — nearly a third of the population — own a smartphone.

Apple iOS devices currently represent 28.7% of this market, and sales are projected to double over the next year. Meanwhile, Android devices, holding 46.9%, are expected to continue to dominate in sales and potentially increase to over 50% of the overall market.

With the rise of smartphone usage, mobile apps are now the dominant user trend with app usage now exceeding web usage by over 20%. App sales stats also reflect this: yearly global app downloads have tripled since 2010 alone. Users have come to expect a multitude of applications to address almost every need, and we are seeing the market rise to those expectations.

Apple's App Store boasts over half a million available apps, with an all-time download count of over 18 billion and current download rates of over 1 billion apps per month. These numbers are remarkable, but it is even more remarkable that App Store downloads represented only a 31% market share for app downloads in the second quarter of 2011. These numbers don't even begin to touch overall app usage. Google Play's app market, comprising a hefty 44% of the market share for app downloads in the same quarter, offers over 400,000 apps with downloads totaling over 10 billion overall and an increase of 4 billion downloads in the last 6 months alone.

Businesses must build for a mobile audience

App usage is increasing exponentially due to consumer demand and will only continue to do so as smartphones become the standard for users. With this increase will come changes in user expectations and a viable business opportunity for developing mobile applications. If businesses want to remain relevant in an increasingly mobile-driven culture, they must rise to meet these user expectations by substantially increasing their mobile presence. A website is no longer enough. Businesses will be expected to add a mobile version of their site and an app for the Android and iOS platforms.

Creating an application for a single platform (web, mobile web, Android or iOS) limits the reach of your app. Users expect a viral loop — a continuous, shared experience across platforms — and don't want to be locked into a single device. Businesses must not be multi-platform so much as fluidly platform-agnostic: any barriers to content because they arrive from a different platform (through a Twitter link on the web, or even an email) shouldn't present a barrier to your content.

This viral loop is also vital for growth. iPhone-only or Android-only applications can create a bad reputation with the community they did not serve, with a user's social circles rarely representing a single platform. One of the worst experiences for any user is to share an app with peers who can't access it, or for a potential user to seek out an app and not be able to experience it.

Building cross-platform apps can be complicated, and there are many considerations and potential complications for those looking to take advantage of the mobile opportunity (which we'll address in the next section). It's these problems that our framework has been designed to overcome.

Section 2

Considerations for developers

Both for existing web developers looking to break into the native app market, and for businesses looking to build their mobile presence, mobile development presents a number of complications. Which platform — which language, which app market — is worth investing with your time and resources? Native platforms, their programming environments and their marketplaces, each have their own advantages and disadvantages. This section will survey some of these.

Native app development: the mobile programming landscape

With a number of smartphone platforms sharing a busy industry space, developers are likely targeting the two leading app markets — iOS and Android — before considering the others. We saw in the introduction the health of these markets and the opportunities they bring.

Until recently, iOS was a clear favorite for mobile development, and learning Objective-C was a worthwhile investment for any developer looking to capitalize on a boom in mobile. Since, Android has seen spectacular growth and the picture is more complex. Should existing Objective-C developers familiarize themselves with Java, to compete with other development shops? Should those new to native app development be learning both? And looking to the future, shouldn't developers and app shops also think about Windows Phone 7 or the Windows 8 for PCs and tablets?

First amongst developers' consideration will be the ease of learning and coding with the language. The workflow from conception to deployment can vary wildly. Of course, choosing which platform is 'easier' to write in, or even which is more powerful, will depend on the kind of applications you build as well as your background as a developer.

— iOS and Objective-C

Apple's iOS mobile operating system, running on iPhones, iPads and iPod Touch devices, is the market leader. With a more mature app market, with far greater revenue opportunities, many choose learning the Apple-only programming language to capitalize on the unique opportunity the App Store brings.

Developing for Apple hardware means writing in Objective-C, an object-oriented extension of C, with the Cocoa Touch framework. Cocoa Touch is the API for building apps for iOS devices — based on the desktop OS X Cocoa API toolset, it is simply a set of libraries written with the Objective-C language. Creating, compiling, running and debugging is done through the XCode development environment and its simulator, which must be run on a Mac.

In general, developers who favor iOS programming cite great documentation, tools and advice on best practices that help smooth-out development and guide through new developers, as the benefits of the platform. Easy GUI building with Interface Builder, and mature, well-worn APIs can make regular development tasks quick and straightforward. Another great benefit — when compared with other platforms — is intra-platform compatibility. As a single hardware manufacturer and software gatekeeper, Apple achieve great consistency across devices — not just in hardware, but in software updates too. iOS developers can build with confidence that they build for one OS, and the platform will be consistent.

Amongst its disadvantages, Objective-C is arguably harder to learn (than Java), depending on programming background, and developing for iOS requires Mac hardware. These are stumbling blocks for some.

— Android and Java

Android apps are written in Java, another objected-oriented extension of C. In syntax, Java looks very different from Objective-C though its behavior is similar. In programming basics, there isn't too much to differentiate learning and developing in Java over Objective-C more than personal taste. In terms of development environment on offer, Android might be either more open, friendly and rough-around-the-edges (and more customizable for it) — you aren't locked into XCode — or you might prefer the guidance of Apple's smoothed-out developer kit in that its APIs can handle many tasks and functions for you.

As the best-selling smartphone platform, with 700,000 devices activated daily, Android is an enormous market to avail of. Its marketplace doesn't offer as much revenue as Apple's App Store — but that is expected to change. But as the number of Android devices grow, the number of differentiated development points grows too. With a wide range of devices with their own OS quirks (at the hands of operators, rather than Google), screen-size and resolution differences, Android brings more compatibility challenges within the single platform than any other OS. Developers must design for up to 100 different devices that run its OS — either adapting for each device, or accommodating to the lowest common denominator.

You can learn the best practices for building with multiple screen-size support and the like from the beginning, and most devices will be similar enough in form and firmware to render your apps more or less identically — but even large development teams lack the resources to test and debug for every Android device. And this is a picture that, for now, is only getting worse.

▪ Developing for web and mobile web

As mobile grows, the web remains crucial for businesses to maintain a competitive presence — to be in prospective customers' web search results. Businesses working with a single-platform app alongside a website are still handling multiple streams of development work.

Depending on the business, where resources are tight the web can be neglected in favor of mobile only — resulting in a static site that links to app stores in the hope of driving customers on the promise of the app's pitch — but with no engagement with the app itself. This only results in lost conversions.

Other app businesses, like Instagram or Foursquare, recognize the value in also serving their users' content to the web, and see it drive users to the mobile apps by providing a taste of it on the web or mobile web — converting engaged web users to mobile installs.

Yet designing for the web this way is another development project, another team and another codebase — a lot more work just to deliver the same back-end.

Multi-platform development

For businesses to build a strong mobile presence, there is great resistance to choosing one platform at the exclusion of the other. To be on multiple devices, and visible in multiple markets — not to mention the web — a multi-platform strategy can be essential.

Yet in approaching a cross-platform strategy, companies face building the same apps with the same features in radically different programming languages and development frameworks — each with their own native quirks and development requirements.

Maintaining two or more native app development streams, on top of web and mobile web development, typically involves hiring in the extra development talent to work on parallel apps. Not only does this come at greater expense, and at least doubles the time it takes to develop and deploy features, it stunts innovation and makes consistency across platforms almost impossible to maintain.

As a web developer, or web development team, looking for a slice of the app market, learning one or several languages presents a significant barrier to entry. Training to a professional level in a language, with the long-term benefits of those skills unknown, is not a straightforward or easy solution, especially when you can take advantage of what you already know.

Section 3

The HTML5 opportunity

Having surveyed the complications developers face in approaching a multiple-platform development strategy, in this section we'll address the opportunity wrapped HTML5 apps offer: both as a way for web developers to leverage existing skills to compete as native app developers, and for companies needing to solve cross-platform deployment headaches by simplifying their development stream.

‘Write once, run everywhere’

In approaching a multi-platform strategy for building a mobile presence, companies face building the same apps with the same features, for four or five platforms at once — each in different programming languages, with their own native quirks and development requirements. Maintaining that kind of development stream not only slows progress to a crawl and stunts innovation, but makes consistency across platforms almost impossible to maintain. Companies face hiring new developers with different skill-sets to work independently on the same features — an expensive, slow mess, that creates unnecessary friction to building a strong mobile presence.

HTML5, cross-platform apps are therefore a great opportunity. Rapid innovation in the technology in recent years has led us to a point where rich, complex applications can be built with the open languages of the web — HTML5, CSS3 and JavaScript — and can match native apps in functionality. Whilst mobile platforms differ in their programming languages and development environments, browsers have far more in common across devices, with all major mobile browsers built around the WebKit layout engine. This means you can reliably develop for the single mobile browser with one codebase, deploying the same app across multiple devices.

With a hybrid wrapper, you can then have your app packaged as a native app: allowing you to access native functionality, and marketplace presence. This way, you match the benefits of native apps with the benefits of cross-platform.

Understanding hybrid apps

— HTML5 code with native wrapper

Whilst much can be achieved in an HTML5 web-app, bookmark-able by the user and accessed through the device's mobile browser (or PC browser extension), they lack native functionality on mobile. If you want to access the phone's data and architecture — to take photos from the device's camera roll to manipulate in the app, to deliver notifications, and so on — you need a native wrapper. Using a native wrapper allows you to also take advantage of platform marketplaces, and the revenue opportunities therein.

Trigger's own native wrapping framework is designed to be as straightforward as possible, getting out of the way whilst you write your app. Write your app in HTML, CSS and JavaScript — building towards a target test platform (Chrome or Android), before using Trigger's command-line Forge tools to compile your app for other platforms (Android, iPhone, or as a browser extension). The result is packaged native builds that are ready for you to distribute to Google Play, Chrome Web Store, the iOS App Store, and so on.

— Cross-platform

The Forge API can access native functionality on Android, iOS and desktop browsers. The wrapper acts as a bridge between your code and the native phone architecture: you write JavaScript commands in your code that will invoke native features. Write a notification alert, and it will be invoked in Android's notifications tray and in iOS as an alert or banner in Notifications Center.

— Designing for all devices

From the codebase you write once, you're developing for all smartphones — a truly enormous audience. An estimated 1 billion HTML5-compatible phones will be sold in 2013, up from 336 million in 2011. You reach them all with minimum extra effort.

By designing for the browser you benefit from the browser's greater responsiveness and fluidity when it comes to dealing with devices with different screen sizes. If you're using decent resets and templates, they'll help too — and you can use JavaScript to direct certain functionality to certain screen sizes if required.

▪ Leveraging community tools and libraries

Working with HTML5, CSS and JavaScript allows you to take advantage of a healthy community of third parties, developing dozens of tools and libraries that can help quicken and improve your development. The community about web and mobile web development goes way beyond that of native programming — and the volume and variety of timesaving tools, templates, functionality libraries (not to mention community forums, tips and tutorials) will allow you to build rich, complex applications and neat features with far greater ease.

Here are a few worth mentioning. In the next section you'll find some examples of how these can be used in practice.

- HTML and CSS resets like HTML5 Boilerplate are a useful starting point. Resets come in a variety of grades of severity that you can tailor to your needs. As each mobile browser implements WebKit slightly differently, with some variation in defaults and behavior, a cross-mobile reset helps normalize across browsers and optimize performance.
- Zepto.js is a lightweight JavaScript library — like jQuery, but shrunk to under 5kB — optimized for mobile browsers. If you commonly use jQuery for simplifying your workflow and performing common JavaScript tasks, Zepto.js will cover most of your needs, with jQuery-compatible syntax, but leave a much smaller performance footprint.
- Backbone.js will help structure your app as a single page, and manage operations and transitions between different parts of your site. Backbone.js will handle the separation of the sections of your code and add the functionality for common tasks that organize and handle your pages and data.
- jQuery Mobile offers a user interface system for mobile that makes creating native-feeling apps extremely straightforward. Its code is lightweight and flexible, and provides a range of tools for building touch interfaces (with degradation and progressive enhancement). It has default layouts and UI widgets you'll use repeatedly.

Section 4

HTML5: fact and fiction

There are limitations to working with HTML5 and native wrappers which are important to be kept in mind and to inform your development approach. HTML5 is hugely powerful, and hugely flexible, and most types of app are well-suited to this approach — but others, particularly graphically intense applications and games, remain better suited to native development for now.

Of course, some assumptions pertaining to HTML5 apps and their capabilities — that they can't run offline, access device data or native dialogues — are plain fiction. Many limitations (particularly speed and performance) have been overcome in recent years thanks to advances in the technology, in HTML5 standards, third-party libraries and constant improvements to browser engines and device hardware. Full, rich applications are well within reach using just web technologies, and the advantages are enormous.

Speed

In the main, web apps will run slower than their native equivalents, simply as a result of requiring more server calls. It's important to weigh up this fact with the benefits of working cross-platform, and the suitability of the framework for certain types of app.

We continue to benefit from regular, significant improvements in browser technology, making web apps run ever faster. The big browsers are constantly improving their browser engines, and new hardware means devices are getting faster, too. The effect is significant: Android's 4.0 browser showed page rendering speed gains over 2.3 of nearly 220%; Galaxy Nexus hardware approximately doubled the speed that the browser ran over the Nexus S (devices just one year apart). Your apps will inherit these benefits.

With Trigger, our core strategy has been to make our framework exceptionally lightweight. The JavaScript we use to wrap your app's code is less than 14kB, and our bridge to native functionality is as direct as possible.

Amongst other current developments, HTML5 apps will continue to see improvements to speed through improvements to HTML5 offline caches. And in general, extensive caching and pre-loading content is good practice — by querying local data and avoiding calls to the server, performance will improve. Beyond that, certain tools and tricks can help you keep your app running fast:

— Backbone.js

Backbone.js saves you from unnecessary page loads — and waiting times — by loading your app as a single page and manipulating the data to be served as if traditional distinct pages, showing and hiding elements to manage transitions. Rather than setting up URL routes on your web server, set up hash fragment routes on Backbone.js:

```
MyApp.Router = Backbone.Router.extend({
 routes: {
   "": "index",
   "friends/": "friend", // e.g. friends/
   "friends/:friendId": "friend" // e.g. friends/42
 },
```

... and have data from your server propagate efficiently through your app:

```
var friends = new Backbone.Collection;

// Show friends as we get data from the server
friends.bind("add", function(friend) {
 new FriendView({model: friend});
});

// Request friend data from the server
forge.request.ajax({
 url: "https://example.com/friend_list/",
 success: function(data) {
   friends.add(data);
 }

});
```

— Zepto.js

HTML5 apps can be slowed by needlessly large libraries like jQuery, which carry many features superfluous to mobile deployment — including all kinds of cross-browser compatibility bloat. We recommend a smaller framework built just for mobile, like Zepto.js (see also XUI and jQuery Mobile) as a worthwhile replacement.

Example:

```
$('div > .flip').tap(function() {
 $(this).parent().animate({
 rotateZ: '180deg',
 opacity: 0.8
}, {
 duration: 500,
 easing: 'ease-out'
});

$('.hide').bind('click', function() {
 $(this).closest().hide();
});
```

▪ Native look and native feel

One challenge of HTML5 apps is in delivering the equivalent look and feel of native apps — an issue that faces those developing for iOS devices in particular. Without due attention to looks (a white background is a giveaway) and responsive features — such as scrolling effects — an HTML5 app will lack the polish of a natively-built app. But a well-designed HTML5 app should appear no different to the end user: they should not notice, nor care for, the difference.

What you can do:

— iScroll

Employing JavaScript libraries in your code can cater to these issues with very little extra work. iScroll allows you to build in native scrolling within divs, so you can build absolutely-positioned headers and footers (that provide the native 'look') with scrolling content in between (providing the native 'feel'). iScroll is less efficient than native scrolling, however, and apps will run more smoothly without fixed headers and footers — but where it's needed, iScroll should do the job.

Example:

```
<div id="wrapper">
     <ul>
          <li>One</li>
          <li>Two</li>
          <li>Three</li>
     </ul>
</div>


var myScroll = new iScroll('wrapper',{
     snap: 'li',
     momentum: true,
     hScrollbar: false,
     vScrollbar: false }
);
```

— jQuery Mobile

jQuery Mobile offers a range of JavaScript and CSS tools for building touch interfaces, with default layouts and UI widgets that make it trivially easy to add native-feeling navigation and the like.

For example, we can build a full native-looking navigation menu using jQuery's CSS, and use 'data-transition' attribute tags to call in transitions.

Simply include jQuery's JavaScript and CSS files in your HTML header:

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0a1/
jquery.mobile-1.0a1.min.css" />

<script type="text/javascript" src="http://code.jquery.com/jquery-1.4.3.min.js"></
script>

<script type="text/javascript" src="http://code.jquery.com/mobile/1.0a1/
jquery.mobile-1.0a1.min.js"></script>
```

Then use data attribute tags to specify features — including roles, themes and transitions:

```
<div data-role="header">
     <h1>Page Title</h1>
</div>

<a href="index.html" data-transition="flip">Hardware-accelerated 3d transforms!</
a>
```

```
<ul data-role="listview" data-theme="g">
     <li><a href="theming.html">easy theming</a></li>
     <li><a href="transitions.html">page transitions</a></li>
     <li><a href="fast.html">fast single-page apps!</a></li>
</ul>

<div data-role="footer">
     <h4>Footer content</h4>
</div>
```

— Trigger.io Forge

Our framework, Forge, plays its part, too. The native wrapper will call native functionality dialogues: with our API, you can use native interfaces to be invoked from your own JavaScript, functioning in just the same way as natively built app. You can make use of native notifications, cross-domain requests, file access and address book access.

More than that, you can use Trigger.io for implementing true native UI components, like a fixed native header bar — so you don't even have to fake it.

Best types of app for HTML5

Some types of app are particularly well suited for HTML5. If you're serving heavy text, and interacting with data, then managing text and communicating with web APIs is going to be that much easier with web languages. Popular apps that have shown the most promise with HTML5 have tended to be tie-ins with existing web services — such as Amazon's, Etsy's, Facebook's — that require frequent interaction with web data. Twitter clients, apps serving newspaper or other content APIs — all are strongly served by working with web architecture.

Graphically intense applications and games remain better served by native builds. You can certainly develop minimal, low-graphics games and puzzles — turn-based multiplayers, board and card games, SuDoku puzzles and the like — but mobile browsers can't yet handle extensive use of animations and 3D or heavy-2D graphics without significant performance issues.

Those familiar with HTML5 proof-of-concept experiments will know that graphically intense games are well within its power, too — but for now, to really push the power of mobile devices, native apps will serve game developers better.

Section 5

Case studies

Facebook

Facebook's mobile team faced extreme cross-platform headaches: with a responsibility to develop for multiple platforms, devices and screen-sizes, they also needed to deploy a large, complex and constantly evolving feature-set that could be consistent across all platforms. They were perhaps the first to truly tackle these problems head on, and their solution is a model of good practice in cross-platform HTML5 app development.

Facebook currently support all major devices from their single mobile website, which they run within an integrated browser contain that they deploy in native packages for all platforms. The site code stems from the original m.facebook.com site (which itself preceded app marketplaces) but serves as the basis for the degraded site for low-end devices.

Wrapping their mobile site within a native container, they offer platform-specific components (or high-end features vs low-end features) within small feature loops on one site, rather than building separate versions of the site. Compatibility is served on a component-by-component basis, with high-end functionality delivered to capable phones — more touch features, AJAX and HTML5 functionality. Using JavaScript commands, their container acts as a bridge to features of the site that run natively: such as opening the camera to take photos, which can then be directly uploaded to main site from within the app.

Initial performance issues have been solved by extensive pre-caching. On the initial load, the app fetches and displays the basic structure of the news feed immediately, before seeking out supplementary content from within the feed that the user might call upon next. The result is that the app is fast and highly responsive, appearing to have fully loaded all content — whilst being ready to deliver it as soon as the user clicks.

With a single web-backed app in all devices, Facebook can ship versions of their app daily, testing new features (or removing buggy features), without repacking or waiting for App Store approval. They can also rely on knowing all their users will be running the single latest, live version of the app.

NeedAnAccountant.org

Trigger.io supported enterprise development shop Appnovation in building their first app with Forge. The app, for client NeedAnAccountant.org, lets users in Canada find their nearest accountants, and uses our framework to wrap a pure HTML5, CSS and JavaScript codebase as native apps for both iPhone and Android, handle cross-domain requests within the code and to enable native geolocation features by using the Forge API. NeedAnAccountant.org's app serves as a great straightforward example of Forge in action.

The map, at the heart of the app, is integrated with conventional HTML and JavaScript using the standard Google Maps API, with calls to Forge handling geolocation:

```
google.maps.Map.prototype.setCurrentLocation = function(fn) {
     forge.geolocation.getCurrentPosition(function(position) {
          map.currentLocation = {
               “lat”: position.coords.latitude,
               “lng”: position.coords.longitude
          }
          fn();
```

```
    }, function() {
        // Dummy data if geolocation failed
        map.currentLocation = {
            “lat”:undefined,
            “lng”:undefined
        }
    });
    fn();
}
```

Although it can be done with HTML5, geolocation is one feature that’s handled better through a bridge to native: this way the app won’t push ugly, distracting pop-up confirmations to the user on every run.

Forge also handles cross-domain requests within the app, using `forge.request.ajax`, providing a lot more freedom and flexibility than a straightforward HTML5 mobile-app:

```
forge.request.ajax({
    url: requestUrl,
    dataType: ‘json’,
    timeout: 5000,
    success: function(data, status) {
        forge.logging.info(‘[search-nearme] found ‘+status.length+’ results near
user’);
        $(”#loader”).hide();
        displaySearchResultsMap(data);
    },
    error: function(status, errorThrown){
        forge.logging.error(‘[search-nearme] ‘+status);
        $(“#loader”).hide();
        $(“#error”).show();
    }
});
```

Best of all, the app was accepted into the iOS App Store on its first submission. The Forge native wrapper has been built to be strict, neat and tidy, following Apple’s documentation as closely as possible – many apps are rejected due to use of undocumented APIs or failing to include interface elements such as launch images – so customers have less to worry about when it comes to the limbo of App Store approval. Appnovation were able to prepare their app, deploy it quickly and on deadline, without a hitch — keeping the client happy.

Section 6

Getting started

We've surveyed the industry opportunity, the ups and downs of native mobile development and the complications web developers face in approaching it. We've made the case for HTML5 apps and found out what they're capable of. Now we turn to our development framework, Forge, and to how you can start taking full advantage of the mobile opportunity.

Forge has been built to be the best toolset for cross-platform application development, both for mobile and the web. With it you can write your app once in HTML5 and JavaScript and build for native platforms as well as the web. Using Forge's JavaScript API, you can write functions that expose your app to native functionality across platforms.

We believe Forge is the simplest and best: both for web developers looking to build native apps, and for experienced developers looking for a straightforward solution to cross-platform development. Forge makes no demands on your hardware or development environment: write your code however you like, with whatever applications you already use, and use our command-line tool to build and run your app.

Convinced? This final section offers an introduction and overview to getting started with Forge, and building HTML5 wrapped applications. Our online documentation covers all of this in much more depth, with step-by-step tutorials on all aspects of our framework from writing your code and using our API to compiling and deploying your apps. This overview should give a feel for the Forge workflow — for further details and to dive right in, head to the docs.

You will need

A prerequisite is a good working knowledge of HTML, CSS and JavaScript, along with the basics of HTML5. You should be reasonably comfortable at the command line, too — though no specific knowledge is required, with the commands straightforward and detailed in our documentation.

Otherwise, the Forge workflow is platform-agnostic (use your favorite text editor on your favorite operating system), and requires no programming experience.

forge create

The first step is to sign up for free and download the Forge build tools. Making sure you've got Python installed, you can then go right ahead and run Trigger's build environment: `go.bat` on Windows or `go.sh` from the Mac terminal.

Next, write your app. Forge creates a directory with a file structure that holds your app's HTML, CSS and JavaScript files. With that in place, you simply add your code, along with the reference files and images that will constitute your app.

Major structural features, including native functions that you wish to take advantage of, are enabled and disabled in the config.json file.

Working with an existing app

Beginners might find it easiest to get familiar with the framework by starting an app from scratch, setting up the file structure and writing your code within that setup – but you can certainly bring existing extensions and apps to Forge.

If you are working with an app already on your machine, after activating the Forge environment you can simply change directory to the existing app. Within that directory, you should make sure you have an src folder which contains the code for your app — then continue as normal.

forge build

Forge splits the build process between your local machine and a remote build server to return compiled apps quickly. Compiling is as simple as running '`forge build`' from the command prompt or Terminal, and your apps will be generated according to the platforms supported by your account tier, returned to your development directory.

forge run

With your app built, you can run your app on devices or desktop simulators by using 'forge run'. With local simulators installed, you can type the command '`forge run ios`' to launch your iOS build on the simulator. Just the same, type '`forge run android`' to run the app on Android: either on an Android device connected by USB, or on the Android emulator. If you're deploying your app to the web using our Build to Web feature, with node.js installed you can type '`forge run web`' to open up a browser tab displaying your app.

Use Trigger.io's Catalyst debugging tool to help debug your app. Squash your bugs, edit your code, rebuild and run again. Once you're happy with your app, you can package and deploy your builds to app stores.

Deploying to app stores

Once finished, all that's left is to prepare and package your apps for app-store submission. Again, this is all handled with simple commands. The process is similar across the different platform marketplaces, for example:

— Android

First, create a 'keystore' to sign your app, available from Android's developer guides, and reference this password in your app. Then, run `forge package android` to produce an APK file, outputted to newly created 'release' directory on your computer, that is ready for submitting to Google Play.

— iOS

After creating a provisioning profile (either to produce an app that you can run on your own test devices, or for an App Store-submittable packaged app – the two choices are different), you can run `forge package ios` to produce an IPA file. You can drag this file to your iTunes library for your own use, or use the release-ready file for submitting to the iOS App Store.

— Web

After you have built your app, your generated web app can be deployed to any node.js platform. We've added the option to deploy directly to Heroku. This is as simple as setting up a Heroku account (following their tutorial steps), then running '`forge package web`'. Forge will confirm whether you want to create a new Heroku app to use an existing one, before deploying your app live to the web – providing you with a web address which you can link to your domain following Heroku's instructions.

We've developed several tutorials to help you get started and guide you through Forge in more depth. Once you've signed up, we recommend the 'creating a weather app' tutorial in our documentation.