

How to Make an App



**HMTL5 Mobile
Edition**

Perhaps you are a web developer looking to leverage your current skillset on mobile platforms. Or maybe you are an iOS or Android developer looking to enhance your skills with web technologies. Whatever your background, if you want your next mobile app to reach a huge market, then you should consider writing it in HTML5.

Even before HTML5 became an official W3C (World Wide Web Consortium) standard in December 2012, developers — even big ones like Facebook (see sidebar) — were jumping onboard. HTML5 offers a very compelling “write once – run anywhere” value proposition. There are also thousands more developers with HTML5 skills than there are developers with iOS or Android development skills. Furthermore, the number of HTML5-compatible phones on which your application could potentially run is very large — expected to top 1 billion in 2013, according to the research firm Strategy Analytics.

Let’s look at what it takes to write an HTML5 mobile app — what an app consists of, what are some of the more popular tools available to help you through the process — and explore why (or why not) you might choose to go the HTML5 route. But let’s start with a look at what HTML5 is and what it does.

HTML5 is a Spec

Like its name implies, HTML5 is the fifth official version of HTML, the specification of the markup language for structuring and presenting content on the World Wide Web. Unlike a “programming” language such as, for example, Objective-C (the language of iOS apps), a markup language is not compiled, which means it isn’t converted into a file of machine-executable binary code. An HTML5 app is a plain old web app that a web browser loads and renders as the web pages we are all familiar with.

Like any web page, an HTML5 app may need to be downloaded at runtime or, alternatively, it can be packaged as a self-contained app that runs in its own web view and that many people might find indistinguishable from a “native” app.

Unlike an HTML5 app, a native mobile app is one that is written specifically for a particular mobile operating system (iOS, Android, etc.) using the programming language, libraries and other tools supporting that OS. Those tools include APIs that apps can call to leverage device-specific functions directly from the OS, including camera, accelerometer, storage, and others.

Unlike previous HTML standards, HTML5 also supports APIs on devices as well as other features, previously missing in the standard, that make HTML5 a viable language for mobile app development. It is the first HTML version to support multimedia without plugins (like Flash, which is not supported on many

An HTML5 app is a plain old web app that a web browser loads and renders as the web pages we are all familiar with



[Click to tweet](#)



phones). HTML5 also integrates CSS3 (Cascading Style Sheets 3) and JavaScript, making it possible to implement “beefier” apps directly in the HTML5 standard whose user experience (UX) rivals that of native apps in terms of high visual appeal and high interactivity. In fact, it is fair to say that many of HTML5’s innovations are there so that HTML5 apps can be “just as good” as native versions — or even better because they allow developers to reach the largest audience with the least amount of effort. Let’s look more closely at the “web vs. native” debate.

HTML5's Advantages Over “Going Native”

First of all, the HTML5 vs. native choice does not have to be an either/or type decision — given that the best applications run on multiple form factors and that the web is not going away. That said, different environments do have their advantages. Whether you decide to go the HTML5 or native route (at least initially) largely depends on whether:

- A rich user experience is a high priority
- You want to monetize the application
- You're in a hurry to put the application on multiple platforms
- You're looking to collaborate with a large pool of developers
- You want to change the application and content frequently

While HTML has raised its game in terms of user experience with HTML5, native does still hold an edge (for mobile), as it does also in monetization — for reasons we will explore in the next section under native's advantages. For now, let's look at three factors mostly favoring HTML5 mobile development:

Support for Multiple Platforms

One of the major reasons HTML5 was created, as just noted, was so developers could write one implementation of an application and have it run more or less the same everywhere with a high degree of

The HTML5 vs. native choice does not have to be an either/or type decision



[Click to tweet](#)

Writing the app once in HTML5 can avoid those extra costs and get your app up and running on a myriad of platforms much more quickly.



[Click to tweet](#)

interactivity and visual appeal. The motivation behind that goal is easy to understand given that there are at least five major platforms: iOS, Android, BlackBerry, Windows Phone 7/8, and Symbian — with over 28 implementations of Android alone by multiple manufacturers, including Google, Motorola, Samsung, HTC, Kindle, Sony, and others. So let's say you do what many organizations do, which is to write the app first for iOS because you want to show off cutting edge design and because all iPhones run the same version of iOS. Then you want to achieve that same experience on Android (including its many versions). That second effort usually means building out a whole separate team, which can easily double or even triple the cost of the new version. But then you also have to build out a team for each of the platforms as well. Writing the app once in HTML5 can avoid those extra costs and get your app up and running on a myriad of platforms much more quickly.

Proponents of a native approach might then argue, "Yes, sure, but what about all the different browsers and different versions of browsers that are out there? Don't you have to account for those differences, too?" There are in fact over 200 different types of available browser interpretations of HTML5. On the other hand, you can argue that only a small portion of those 200 really matter and that tweaking a program to run on a different browser version is much easier than writing a new implementation from scratch. There are also tools like Modernizr and PhoneGap (both discussed later) that remove much of the burden of making HTML5 apps work cross-browser or even (in the case of PhoneGap) cross-platform. Another approach is simply to put feature detection logic in your code to see whether something is supported in a particular browser (so you can provide an

alternative execution thread if it's not). A good example is the @supports tag in CSS and css.supports in JavaScript to detect browser support for a given style directive.

Large Open Collaborative Community

Organizations have spent the last 15 years investing in web technologies and skills — much longer than they have in mobile app development — so it's probably safe to say that there are many more HTML and JavaScript developers than there are (for example) Objective-C developers. This is backed up by the ranking of projects in GitHub and the number of questions in StackOverflow. Also, as the web matures and the number of new traditional websites coming online declines, many of these web developers can be expected to compete for jobs on mobile app projects. This all means that companies will have many more HTML and JavaScript developers from which to pick than they will developers skilled in any specific mobile platform (and so companies can also expect to pay less for them). The good news for HTML5 developers skilled in writing mobile apps is that they should be at the front of the line for the most interesting and highest paying jobs. What's also good news is that there are many more free resources available, and a lot more knowledge sharing, in the web community than you'll find when developing for a native platform.

Changing the Application Frequently

One "native" advantage is the opportunity for an app to be highlighted in an "app store," thereby easing distribution and monetization. The downside of going through an app store, however, is that it puts third-parties like Apple, Google and Microsoft between you and your customer — so they have ultimate control of how fast apps and updates become

HTML5 developers skilled in writing mobile apps should be first in line for the highest paying jobs



[Click to tweet](#)

available to your customers. If you want users to get app updates frequently, giving those updates directly via a website or email link is the fastest way to go. Every time a user logs into the web app, they get the most recent version of the program. HTML5 is also preferred if you want to serve users who opt to never download an app.

“Going Native” Advantages Over HTML5

The two advantages often cited for native implementations are in user experience and monetization. Let's take a look at each:

Rich User Experience

In a recent Compuware APM survey, 85% of mobile device users said they prefer native mobile apps over mobile websites (i.e., HTML implementations). The top three reasons were:

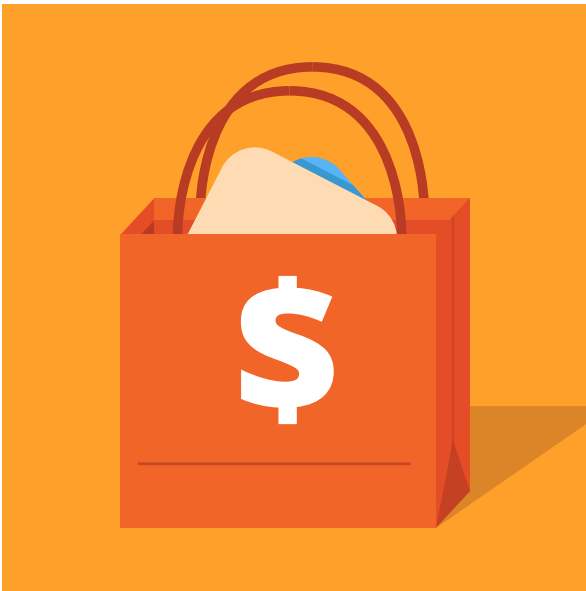
- Apps are considered more convenient
- Apps are faster
- Apps are “easier to browse” (i.e., navigate)

Clearly users want apps that help them do what they want to do fast — hence, the survey results. Native apps respond more quickly to touch events and lag less when users scroll. On the other hand, as network bandwidth continues to increase and as HTML tools become better at producing more efficient code, the speed gap between native and HTML implementations for a large percentage of app types will close. Another performance gap is the ability of HTML5 apps to access native device functions like geolocation, camera and accelerometer. HTML5 does include APIs to the most popular device features, including GPS location and accelerometer, and is adding more. That's “most” but certainly not all. Native APIs outnumber HTML5 APIs by a huge amount — a number

The two advantages often cited for native implementations are in user experience and monetization



[Click to tweet](#)



that just keeps growing. Apple, for example, introduced 1,500 new APIs with iOS 7, including AirDrop and AirPlay. So, if your app relies heavily on APIs to access external functions or services, you may want to think about a native implementation.

One exception — where HTML5 implementations do have a performance advantage — is shopping and weather apps. Both rely heavily on user analytics, which web-based apps can access and provide faster.

Monetization

Of course, the reason many people develop apps is to sell them. And selling apps has become a very big business. Apple's App Store holds over 400 million customer accounts, including credit card information, making it the largest transactional site on the Internet in terms of number of participants. Those customers have downloaded over 50 billion apps and Apple has paid out over \$5 billion to developers. Google's own app store, Google Play, now holds over 800,000 apps. The single HTML5 equivalent is Google's Chrome Web Store, which distributes web apps for the Google Chrome browser. The only other alternative is to distribute your app directly to the public via a website and accept payment the same way.

The Facebook Story

If you're still undecided about taking the HTML5 or the native route, you are not alone. Even the biggest names in the industry have struggled over this decision — and by their own admission have sometimes gotten it wrong.

Facebook is a classic example. Speaking at the 2011 Facebook Developers Conference, here is how Facebook's engineering manager, Dave Fetterman, explained the company's decision to implement the company's mobile app in HTML5:

"Being able to write it once today and ship it tomorrow? That is something that Facebook is really good at and that we love doing, and that is at the center of being able to move fast. Move fast has an implicit third clause — move fast, break things, and fix things fast. That is very difficult to do if you have already shipped your binary to Apple or Android and they have to download another version of it."

But only a year later Facebook announced it had since totally rebuilt the iOS version of its mobile app as a native implementation. As Facebook developer J.P. Dann wrote in a blog post:

"So, we rewrote Facebook for iOS from the ground up (I really did open up Xcode and click 'New Project') with a focus on quality and leveraging the advances that have been made in iOS development."

So why the reversal? The native iOS version loads faster, loads news feeds faster, scrolls faster and is generally more responsive. As Dann explains in his [**blog**](#):

"One way we have achieved this is by re-balancing where we perform certain tasks. For example, in iOS, the main thread drives the UI and handles touch events, so the more work we do on the main thread, the slower the app feels. Instead, we take care to perform computationally expensive tasks in the background. This means all our networking activity, JSON parsing, NSObject creation, and saving to disk never touches the main thread."

The move appears to have worked. As [**Mashable**](#) reported, following the new version's release, ratings jumped from 1.4 to 5 stars in just a few weeks.

Choose your Tools

Another factor that weighs in on the “HTML5 or native” decision is the selection of tools you may choose to develop your app. HTML5 programs consist of three main components: the HTML text markup language for page layout, CSS3 for style, and JavaScript for in-browser execution of program code. And although it is possible to sit down at a text editor and just write your code, there are tools that streamline the process considerably. These include:

- Basic frameworks (e.g., Backbone, Ember.js, Angular) that supply “prefab” structures on which to build a finished app instead of starting from scratch
- Enhance frameworks (e.g., JQuery Mobile, Enyo, Sencha Touch) that provide extra help such as pre-built widgets (e.g., buttons) and CSS auto code generation
- Hybrid app frameworks (e.g., PhoneGap) that turn HTML5 apps into native apps for iOS, Android, Windows Phone, and other mobile platforms

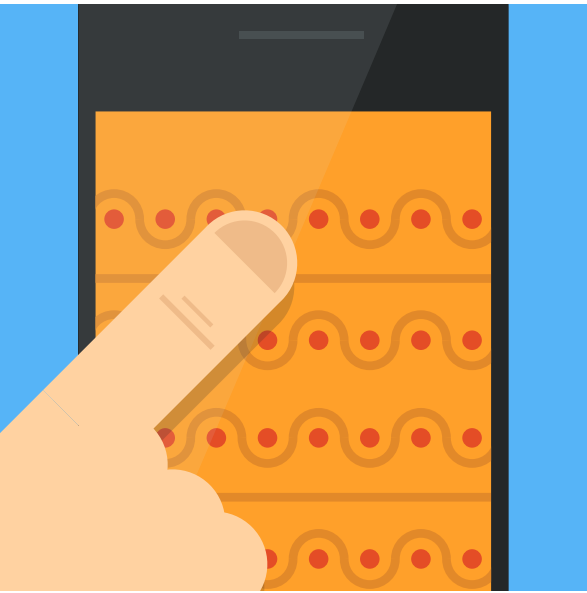
Some developers may like working with these tools more than they like working with native tools. Perhaps they find them easier to use, or the tools let them more easily reach more users with better quality code.

A look at how some of these tools benefit developers helps show why HTML5 is often a great choice for implementing mobile apps.

Some developers may like working with [HTML5] tools more than they like working with native tools



[Click to tweet](#)



JQuery Mobile (jQM)

This is a “touch optimized” framework, meaning that it offers special widgets that streamline the writing of features (like buttons) important on a mobile phone. It’s also “platform agnostic,” meaning that it is compatible with all major mobile platforms as well as all major desktop browsers, including iOS, Android, Blackberry, Symbian, Windows Phone 7, and more. It is built on top of jQuery (a JavaScript library meant to simplify client-side scripting) so it has a minimal learning curve for people already familiar with jQuery syntax.

Other key features include:

- Theming framework that allows creation of custom themes
- Limited dependencies and lightweight to optimize speed
- The same underlying codebase will automatically scale to any screen
- HTML5-driven configuration for laying out pages with minimal scripting

To use JQuery, you link to the JQuery Mobile libraries and stylesheet, either on the JQuery Mobile CDN (content distribution network) or after downloading and hosting locally. Like all HTML5 programs, a JQuery Mobile app has a `<!DOCTYPE HTML>` document type, a header, and a body. The body consists of various “div’s” or html elements that are described by an attribute called “data-role” that tells JQM what the div contains and how it should look. Note that HTML5 made custom “data-” attributes possible and JQuery Mobile makes heavy use of them, as in this example:

```

<body>
  <div data-role="page" id="first" data-theme="a">
    <div data-role="header">
      <h1>Page Title1</h1>
    </div><!-- /header -->

    <div data-role="content">
      <p>Page content goes here.</p>

```

Other roles could be header, content, footer, etc. A div with a “page” data-role represents either a single page or multiple internal linked pages within a page (so internal links are more responsive when tapped).

Note the use of the “data-theme” attribute, which specifies which design theme to use for elements within the div, and can be set to a, b, c, d, or e. The “data-theme” is in fact one of the key ways JQuery Mobile streamlines mobile app development. Rather than write CSS script yourself, you can use the JQuery Mobile Theme Roller. It is essentially an online form that lets you specify various styling choices, including colors, for buttons, pages and other elements, and then group those styling choices together as a theme, which you label a, b, c, d, or e. Theme Roller then automatically generates the CSS code for that theme, which you can then download. To apply a theme’s CSS in your program, simply specify its letter, as in the example above.

Other examples of how JQM expedites HTML5 development are data attributes “data-position” and “data-transition.” Use data-position to specify whether an element should be fixed, in which case it will either always render at the top (if a header) or at the bottom (if a footer). The data-transition attribute lets you specify how the app transitions from one page

to another when a new page is loaded: side, slideup, slidedown, flip or fade.

JQuery Mobile isn't ideal for every app, however. Some critics may complain, for example, that JQuery Mobile apps tend to look alike and lack visual appeal. Even though developers don't have to use JQM's theme building features, most JQM developers do — which means the apps might not look that creative.

Dojo Offers Out-of-the-Box Themes

Like JQuery Mobile, the Dojo Toolkit is also a framework meant to speed HTML5-based development, especially for multi-browser support. It consists of several parts:

- **dojo** contains the core and most non-visual modules
- **dijit** is a library of user-interface modules for widgets and layout
- **dojox** holds assorted modules not yet considered stable enough to include in *dojo* or *dijit*
- **util** includes build tools such as optimization, documentation, style-checking, and testing

Like JQuery Mobile, the Dojo Toolkit also offers themes. But rather than a service that lets you style your own, Dojo Toolkit offers (currently) four “out of the box” designs in dijit: Claro, Tundra, Nihilo and Soria. Other “out of the box” widgets include menus, tabs, tooltips, sortable tables, maps, gauges and more.

Bootstrap: “Packed with Features”

Bootstrap is a popular web UI framework started by Twitter with a wide following and large contributor base. Bootstrap 3 will have a focus on mobile and add to the existing functionality. At the time of writing this ebook Bootstrap 3 wasn't officially released yet.

As noted on its developers' [website](#), “Bootstrap was made to not only look and behave great in the latest desktop browsers (as well as IE7!), but in tablet and smartphone browsers via responsive CSS as well.” Features include a “12-column responsive grid, dozens of components, JavaScript plugins, typography, form controls, and even a web-based Customizer to make Bootstrap your own.”

PhoneGap

JQuery Mobile and the Dojo Toolkit are useful frameworks for quickly building apps that offer a mobile-like user experience on many different browsers and on many different platforms. But what if you're an HTML5 developer who also wants your app to run as a native app? In other words, it's packaged as a native app, can be distributed through app stores, and can access native device APIs? You also don't want to give up the same write once – run anywhere benefit of HTML5. You want to write the HTML5, JavaScript and CSS code once and have it natively on all the major platforms.

The answer may be PhoneGap, a mobile development framework originally developed by Nitobi in 2009 and then purchased by Adobe Systems in 2011. Unlike JQuery Mobile, it comes with a web-based service (PhoneGap Build) that lets you essentially put the

What if you're an HTML5 developer who also wants your app to run as a native app?



[Click to tweet](#)

HTML-based app inside a wrapper that makes it look like a native app to the phone. The result is a hybrid app — hybrid because the app typically runs in a web-view container and is wrapped by a native UI layer, giving access to native navigation and hooks to standard hardware events.

To perform the conversion using PhoneGap Build, simply enter the web address of a repository (like GitHub) holding your app's HTML, JavaScript and CSS3 files. You can also upload the files as a zip archive. The PhoneGap service includes a dashboard that lets you select which platforms you want the app to run on and also insert any credentials needed to submit the app to a platform's particular app store. (Apple has approved apps built using the framework.) The service then automatically generates the hybrid app.

A product that shares many of the same goals as PhoneGap — and is actually built on top of PhoneGap — is IBM's Worklight. In addition to repackaging apps to run natively, like PhoneGap, Worklight also includes (among many other features) a Worklight Studio, a full IDE (integrated development environment) such as you might use Android Studio/Eclipse (for Android development) or Xcode (for iOS). Other features include support for mixing HTML5 and native coding, encryption of locally stored data and offline authentication — plus third-party integration with frameworks such as PhoneGap and JQuery Mobile.

Progressive Enhancement

One of the arguments against using HTML5 for mobile apps is that different browser/platform combinations support different HTML5 features. Solutions like JQuery Mobile, Dojo and PhoneGap attack this problem by producing code that is widely compatible across environments — leading to what some might call a “lowest common denominator” result. But what if you want your apps to take advantage of the latest and greatest HTML5 enhancements everywhere you can — or can someday when more browser support arrives? This future oriented view is called “progressive enhancement.” A developer who subscribes to this view may not care that few (if any) browsers currently support a feature — the developer will put the feature in anyway, along with feature detection logic that turns it on when the feature is supported someday.

Developers can write that logic themselves — if/then conditional statements that test the browser to see if the feature is present — and then branch to the appropriate code that either uses the feature or does a workaround. Another approach is to use Modernizr, a small JavaScript library of browser tests. Upon page load, Modernizr sends an instruction to the browser to test each of dozens of features. It then adds classes to the HTML based on what features are or are not supported. For example if the browser does not support HTML5’s text shadow feature, Modernizr writes a `.no-textshadow` element. If the answer is “true” when the program tests for that element, then a workaround might be to display the text in a different color — one that doesn’t need a shadow to stand out (like white text on a white background would).

Building in support now for features that will be supported later means you might not have to update the program later, and that there is no delay between the time a browser supports a feature and when users get to enjoy it.

Modernizr may be the best known example of a class of products designed to fill gaps in browser capability at run time, but it is not the only one. Another example is a class of products called polyfills (or polyfillers), which takes its name from a product used in the UK to fill cracks in walls, like Spackle is used in the U.S.

A polyfill is a downloadable piece of code that provides capabilities not built into a web browser. There are literally dozens of polyfills available on GitHub. An example is Remy Sharp's **[storage polyfill](#)** that enables IE7 AND IE6 support of localStorage and sessionStorage. For more information on polyfills, check out the Polymer Project at **www.polymer-project.org**.

Why Won't My Web App Perform Like a Native App?

One of the ways a user can usually tell a web app is a web app is if scrolling isn't smooth or transitions and animations stutter across the screen. This is called a "janky UI" and it happens when the browser's rendering engine can't keep up with all the things the app is trying to do. The problem is that the browser only has roughly 16ms to run all the JavaScript, perform layout, paint and do whatever else it has to do to display the frame before it has to start all over again with the next frame. If the browser takes longer than that interval it's called "busting the frame budget" and the display "janks."

There are measures developers can take to prevent jank. One is to use the requestAnimationFrame (rAF) API instead of what many developers do, which is to use either setInterval or setTimeout every 16ms. The problem with these two calls is that the developer is assuming that the system knows what a millisecond is (i.e., timing resolution is perfect) and that all screens refresh at exactly the same rate — neither of which assumptions are correct. The clocks inside computers and smartphones are only accurate to a few milliseconds, so the software can never know for sure how long 16ms really is. Furthermore, depending on factors like brand of mobile device and whether background tabs are hogging resources, screen refresh rates will vary. So even if the clock inside the smartphone were

perfect, the browser would still drop frames because it's not ready to display them.

The advantage of using rAF is that it only gets the frame when the browser is ready, regardless of what the refresh rate is. It also pauses animations in background tabs, conserving resources and battery life.

But rAF alone does not completely solve the problem. You still have to make sure that whatever work occurs during the rAF call takes less (hopefully, significantly less) than 16ms. For example, don't do a lot of process in input handlers. A lot of jankiness happens because an app tries to perform too much JavaScript or attempts to rearrange a whole page while it is also trying to display a frame. A strategy that lets you avoid JavaScript completely in your callback is to use CSS animations instead. Using CSS animation both simplifies your app and lets animations run smoothly even while JavaScript is running — essentially guaranteeing jank-free display performance.

Which means no one may ever know or care your web app is really a web app.



In this example, we will show how to use some of the concepts covered so far to build, debug and deploy a mobile app that uses modern JavaScript tools and frameworks. We're assuming you already know some basic HTML and CSS. So rather than provide a blow-by-blow description of the application's logic, we will walk through the process of creating a mobile app from source code that already exists in GitHub, which you can download as a zip file from here:

<https://github.com/m0rganic/applicant-tracker>. Note that this example was originally conceived from the start as a mobile app rather than as a web app that would then be ported to mobile.

The sample app, called "Applicant Tracker," allows the user to view a list of job applicants along with some details about each applicant, such as their GitHub activity. There are three screens: a login screen (Figure 1), a main screen (Figure 2) that

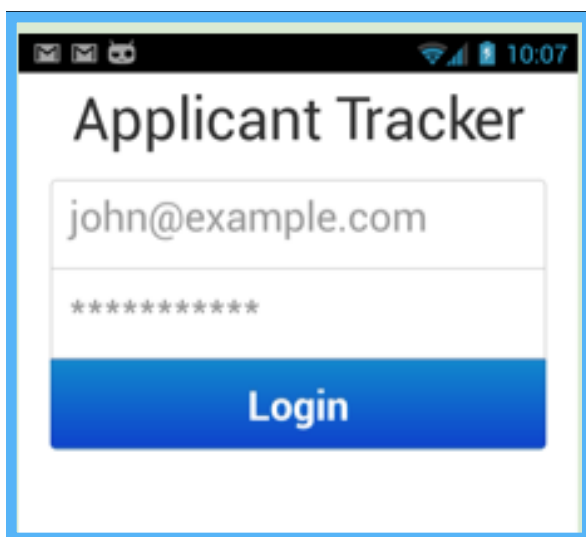


Figure 1: Applicant Tracker's login screen

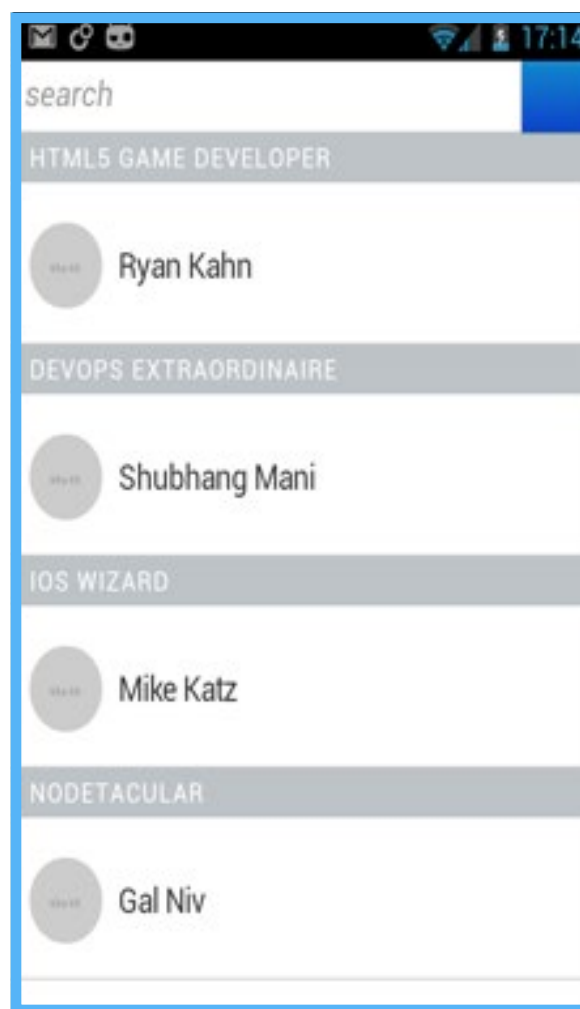


Figure 2: Main screen showing applicant list

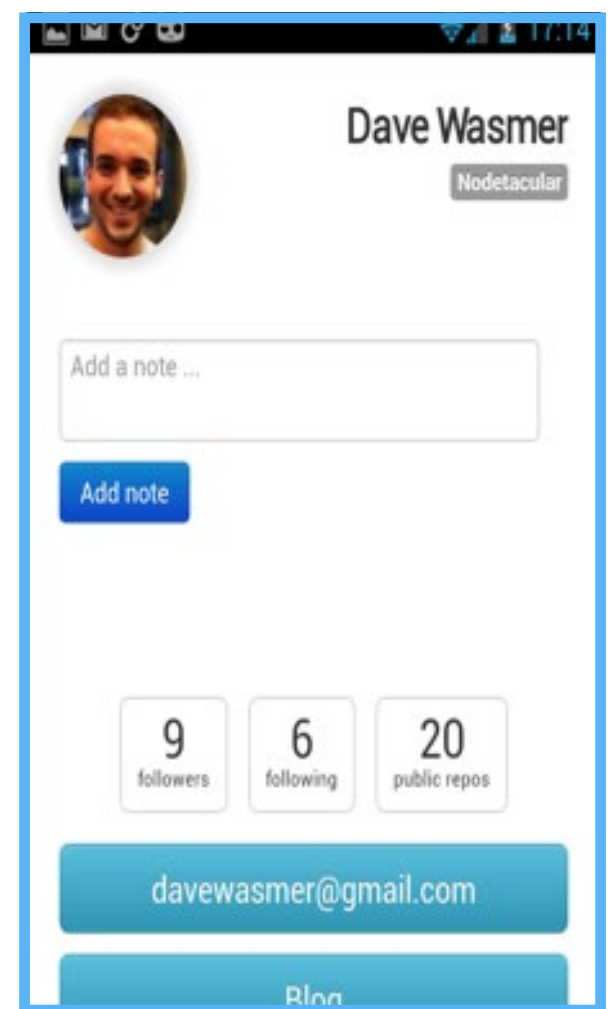


Figure 3: Applicant detail screen displays the GitHub profile, including avatar and follower statistics

shows the applicant list, and an applicant detail screen (Figure 3). The applicant list is stored on Kinvey (as an example of how to integrate a backend) with GitHub details pulled from GitHub (as an example of how to incorporate a third-party API in a data fetch).

Tools Overview

Before getting started let's review some of the tools you will be working with:

- **Chrome** is a modern web browser that includes Chrome DevTools to debug on the desktop.
- **Node.js** is a server-side JavaScript framework used to run the local webserver (which will serve our app once it's built) and our app's build process (which is **grunt**).
- This sample web app uses client side frameworks and libraries including **Backbone.js** and **LESS**, a stylesheet language whose syntax grunt converts to CSS during the build.
- The sample app depends on various third party services: **Kinvey** for the backend (data storage, user management, etc.); **PhoneGap** to generate native apps (iOS, Android, etc.); and **GitHub** to host source code and retrieve applicant profile data. You will need accounts and API keys for all.
- You can run the sample web app in **any modern browser**.

Process Overview

We will walk through the process of creating the app in three steps:

1. Get the source code, run it in Chrome and practice debugging
2. Get a Kinvey account, add a new user and configure the app
3. Sign up for a PhoneGap account and generate native apps for various platforms

Step 1: Get the App, Run It in Chrome and Practice Debugging

This step has four key objectives:

1. Get the latest stable versions of Chrome, Node.js and Grunt
2. Download the source code for the sample app
3. Run the intermediate version of the app on desktop Chrome
4. Look at debugging information for the running app

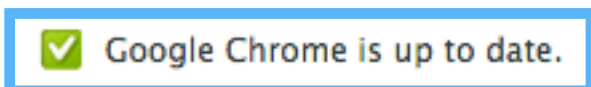
Download or update Chrome

If you don't have a copy of Chrome:

1. Go to <https://www.google.com/intl/en/chrome/browser/>
2. Download the version for your OS
3. Install it. You're done!

If you do already have a copy of the Chrome Browser, make sure you're on the latest stable build:

1. Launch Chrome.
2. Open About Google Chrome
3. If the following icon does not appear within in the page, click Check for Update



4. When the update is complete, restart.

Download or update Node.js

You can either install node directly or use a version manager (for Mac OS X.)

If you're running Windows and don't have a copy of Node.js:

1. Go to <http://nodejs.org/download/>
2. Download the version for your OS
3. Install it. You're done!

If you're running Mac OS X, a Node.js version manager allows you to install and flip between versions freely. There are many advantages to using a version manager, such as running many different projects on different versions of Node.

- nvm - <https://github.com/creationix/nvm>
- n - <https://github.com/visionmedia/n>
- nave - <https://github.com/isaacs/nave>

Install grunt using npm

Grunt is an automation dev tool particularly useful for web apps. Npm comes bundled with Node.js and allows you to install different supporting libraries and

tools that will assist in your development.

Here we use a few things that have useful grunt plugins like **less** and the Node.js simple bootstrapped web server.

If you have installed grunt globally in the past you may need to first uninstall it.

```
npm uninstall -g grunt
```

To get started you will want to install grunt command line globally. You may need to install it using sudo (on *nix, Mac OS X).

```
npm install -g grunt-cli
```

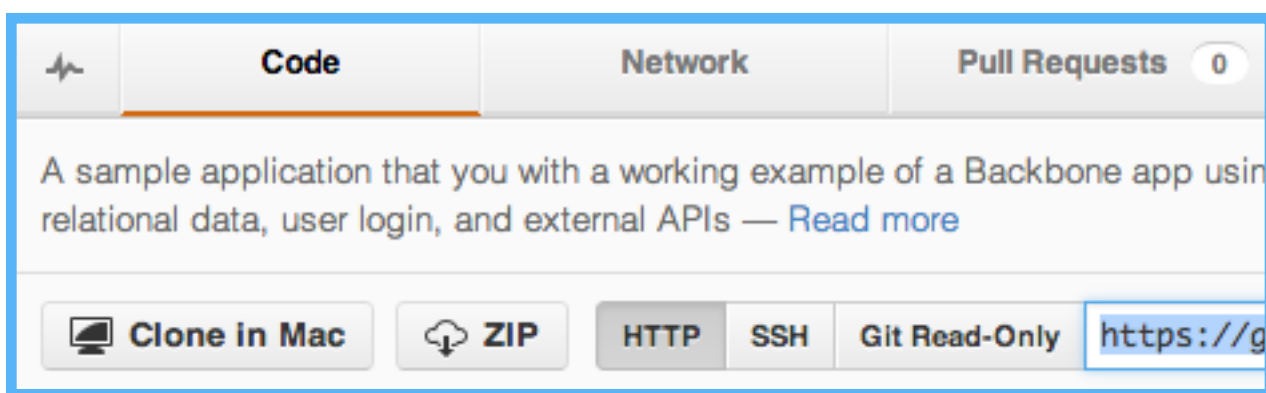
Download the app's source code

You can either clone the git repo or download the zip file.

```
git clone https://github.com/m0rganic/applicant-tracker.git
```

To get the source as a ZIP file:

1. Go to <https://github.com/m0rganic/applicant-tracker>
2. Click the **ZIP** button



or, if you're using the new GitHub interface:



Run the application on the desktop

Now that you've downloaded the source code and all of the required tools, try running the application on your desktop:

1. Install all of the project's dependencies using **npm**. This command will read from the **package.json** file in the root project directory and download each library dependency we use in our build.

```
$ npm install
```

2. Launch a command line and run the grunt process on the project directory.

```
$ grunt
Running "clean:dist" (clean) task
Cleaning "dist"...OK
```

```
Running "jshint:gruntfile" (jshint) task
>> 1 file lint free.
```

```
Running "jshint:js" (jshint) task
>> 18 files lint free.
```

```
Running "copy:resources" (copy) task
Copied 1 files
```

```
Running "copy:js" (copy) task
```

Copied 24 files

Running “copy:img” (copy) task

Copied 2 files

Running “include_bootstrap:all” (include_bootstrap) task

Running “less:include_bootstrap” (less) task

File dist/styles.css created.

Running “connect:server” (connect) task

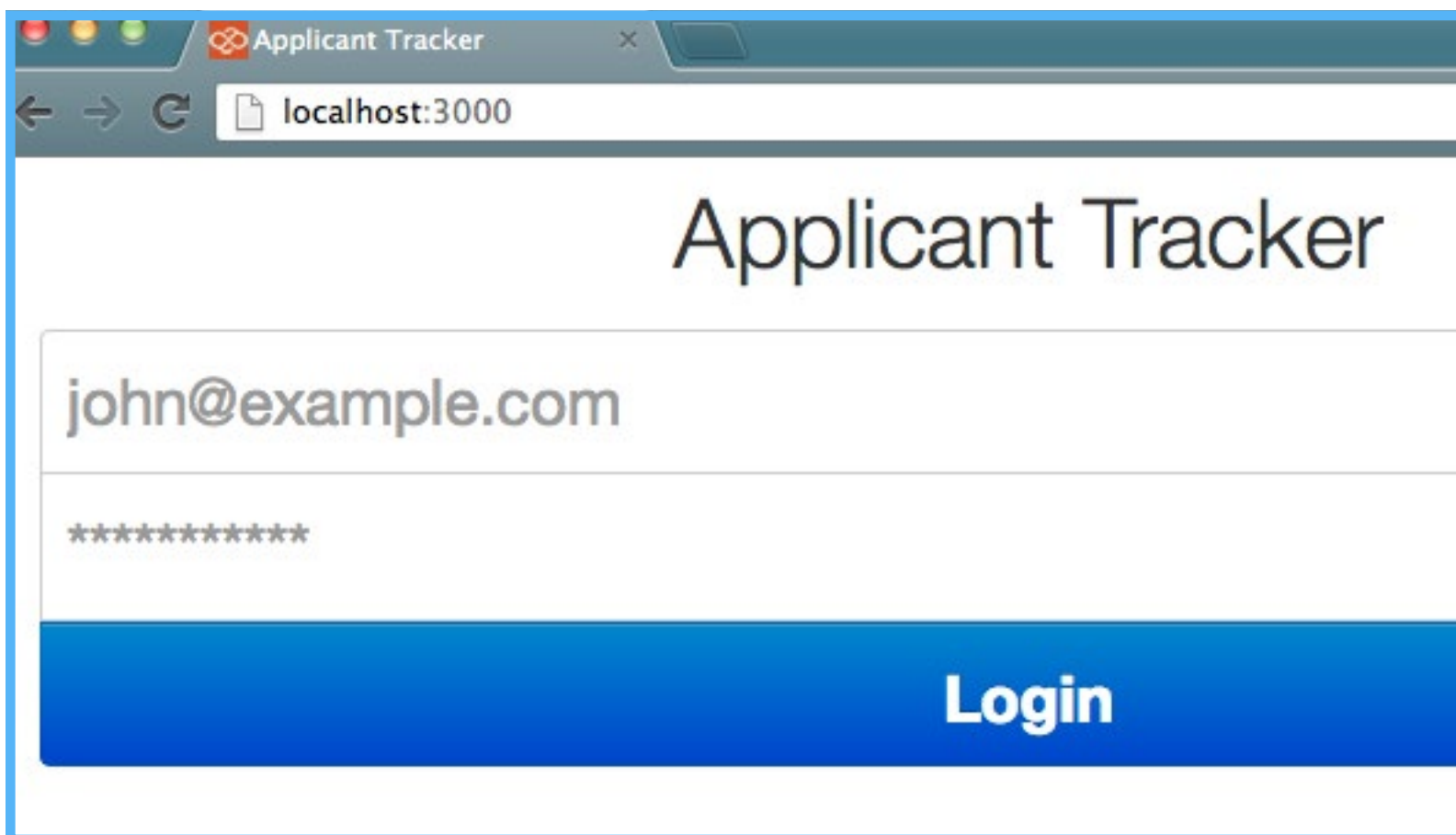
Started connect web server on localhost:3000.

Running “watch” task

Waiting...


If you have trouble see [Getting Started with Grunt](#).

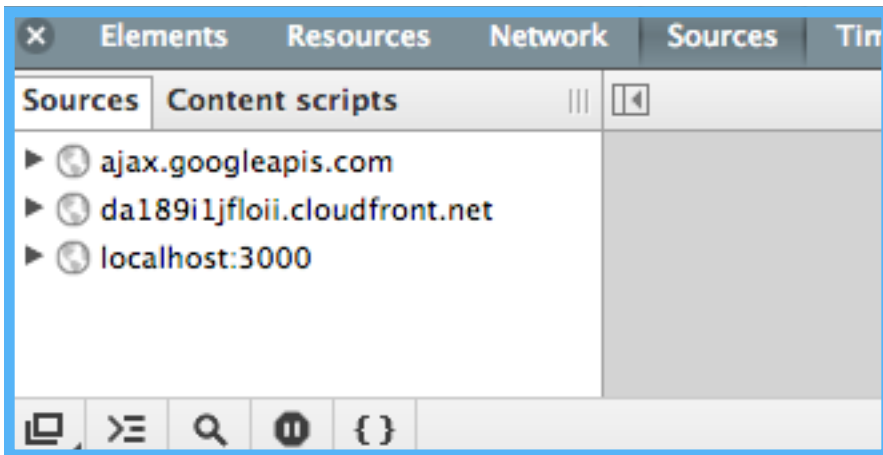
Run desktop Chrome and point it to <http://localhost:3000>. The app should initially look like this:



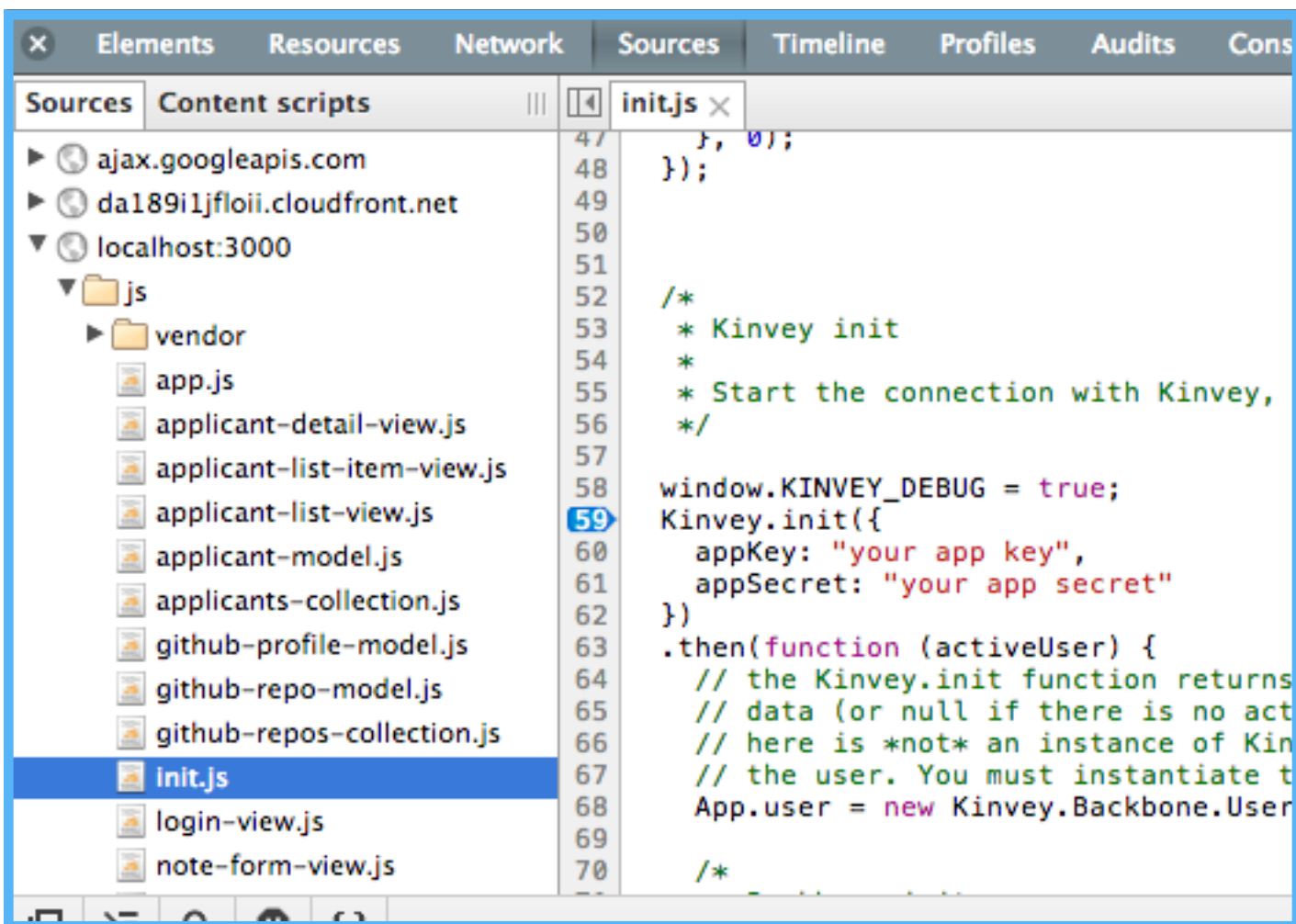
Debug the running app on desktop

The easiest way to debug a web app is using Chrome DevTools:

1. Select the hotdog menu  in the upper right then select **Tools -> Developer Tools**
2. Right click on any page element and select **Inspect Element**
3. Find the **Sources** panel under the sources tab of the DevTools



4. Browse down into the **init.js** and click on the **line gutter** to set a breakpoint on **Line 59**



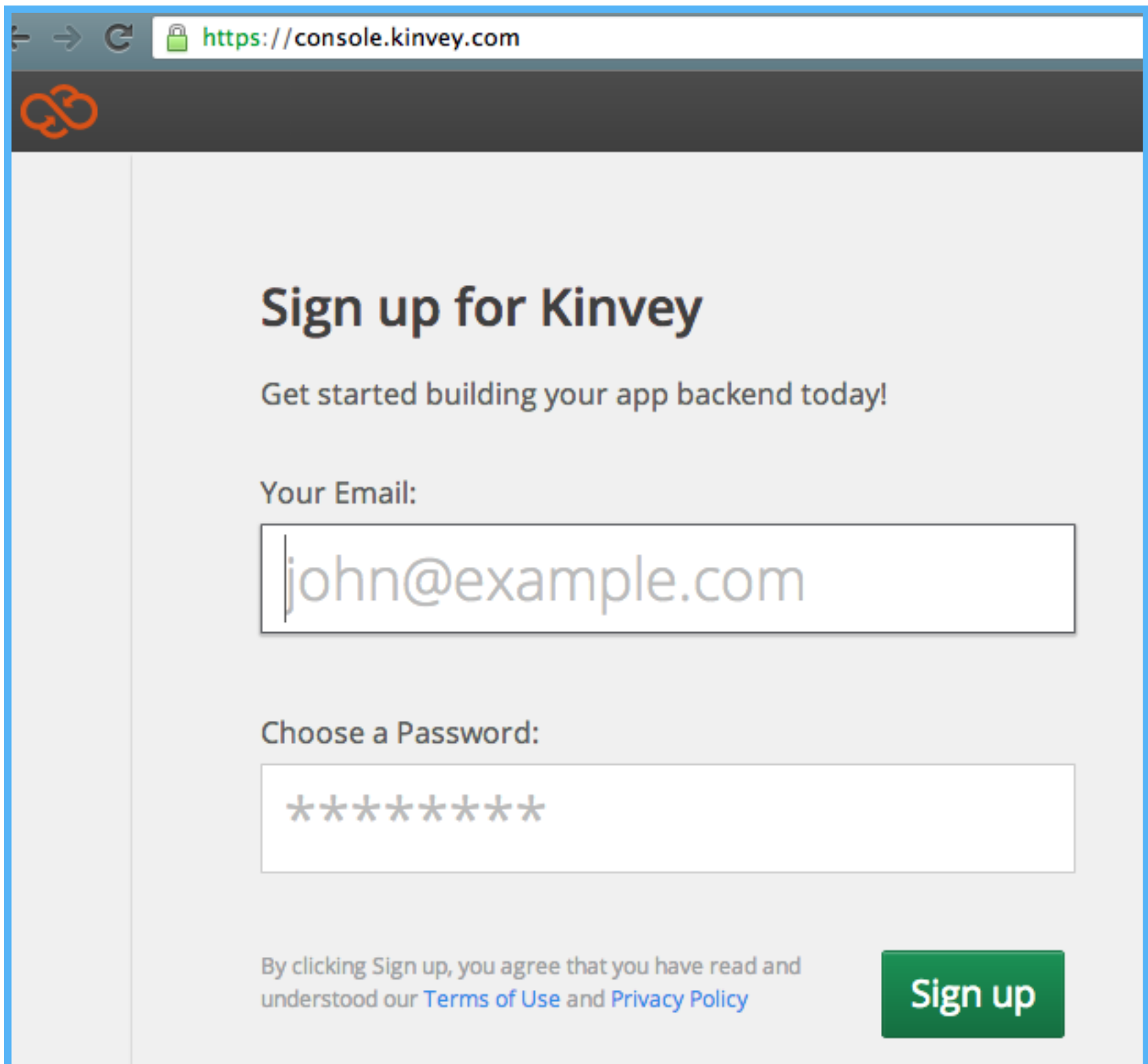
Step 2: Get a Kinvey Account, Add a New User and Configure the App

This step will establish a cloud backend to run the application against. Many applications need the ability to login a new user and store data in shared collections.

Get a Kinvey account

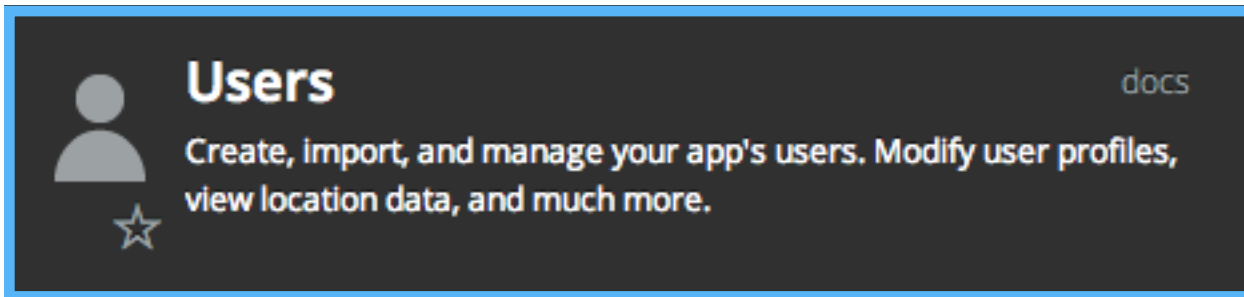
If you don't have a Kinvey account you can sign up for one for free.

1. Go to <http://console.kinvey.com>
2. Sign up. It should look something like this:

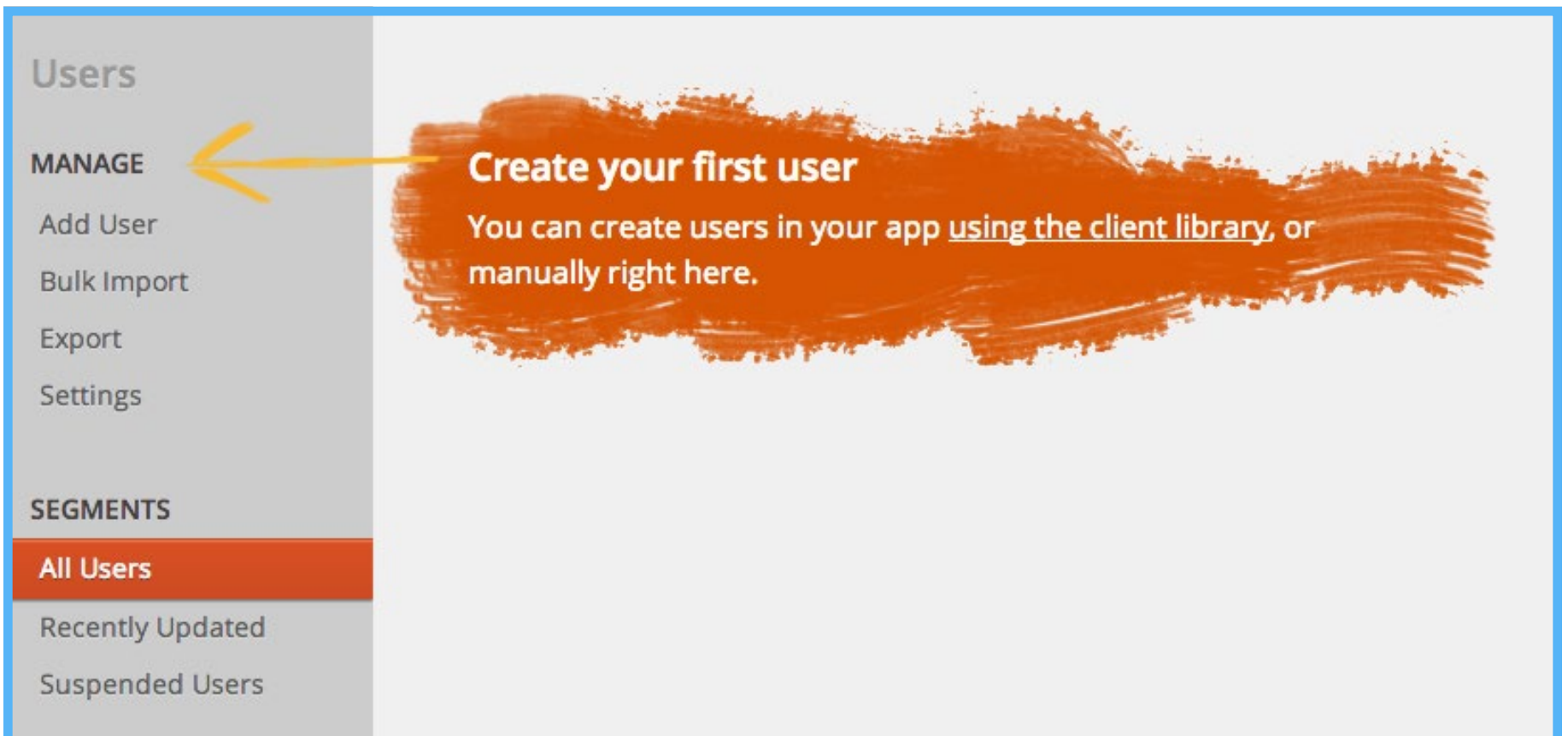
A screenshot of a web browser showing the Kinvey sign-up page. The browser's address bar displays 'https://console.kinvey.com'. The page has a dark header with the Kinvey logo (an orange infinity-like symbol). The main content area is light gray and contains the heading 'Sign up for Kinvey' in bold black text, followed by the subtext 'Get started building your app backend today!'. Below this, there is a form with two fields: 'Your Email:' with a text input containing 'john@example.com', and 'Choose a Password:' with a password input showing eight asterisks. At the bottom left, a line of text states 'By clicking Sign up, you agree that you have read and understood our Terms of Use and Privacy Policy', with 'Terms of Use' and 'Privacy Policy' as blue links. To the right of this text is a green rectangular button with the white text 'Sign up'.

Add a new user

1. Click the **Addons** menu at the top left of the Kinvey [web console](#). Under **Core** menu find the **Users** menu item

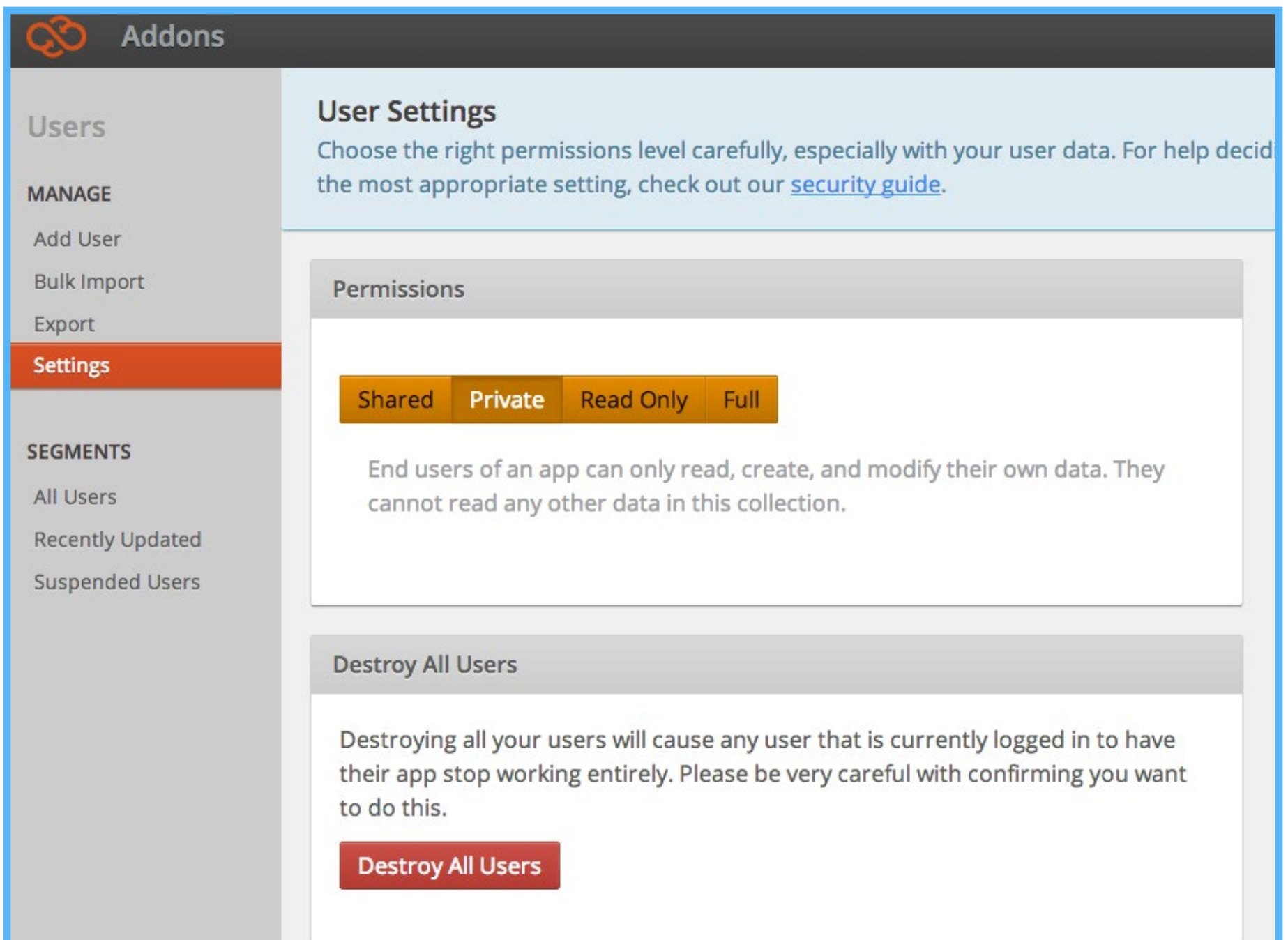


Add a new user on this screen. Make sure to use an email address for the **Username** when creating your user.



continued on next page

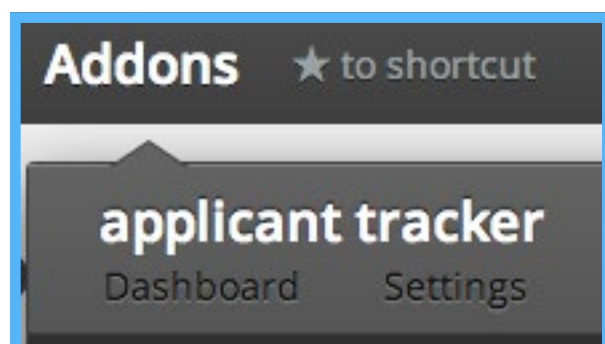
2. Set the permissions for users to “Read Only” (under the [Settings](#) page)



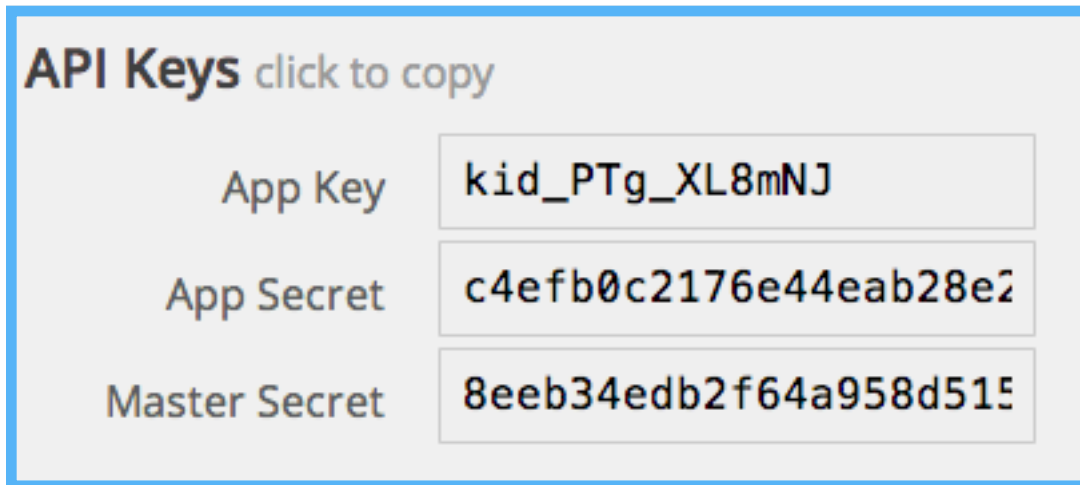
Configure the app

Replace your **app key** and your **app secret** (in js/init.js) with your application's credentials.

1. Find the app key from the Kinvey app dashboard. It should look like this.



Once you've clicked the Dashboard item on the Addons menu, you should see something like the following displayed in the bottom right hand corner of the dashboard. **Note:** *these specific app credentials will not work, you have to get your own.*

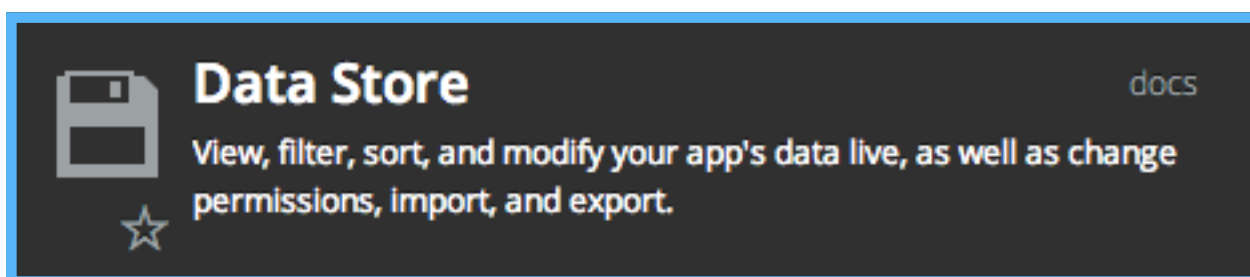


2. After you have inserted the App Key and App Secret into your code you will need to run Grunt again to rebuild the app so that these credentials are included in the build.

Import sample data

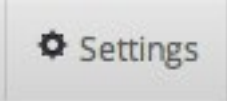
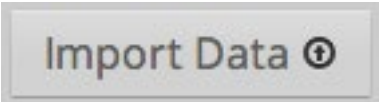
Create an applicant's collection and import **backend/applicants.json** for some sample data to get started with. (Note: You downloaded the **backend** folder from GitHub with the rest of your project back in Step 1.)

1. Click on **Addons -> Data & Storage -> Data Store** menu item.



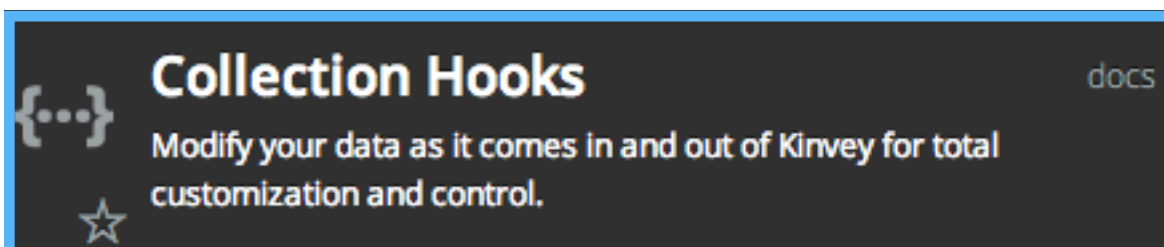
continued on next page

2. Create a new collection, call it 'applicants'.

3. Click the  button for the applicants collection you just created and  the csv file named **backend/applicants.json**.

Add business logic

Plug in some backend code to fetch an applicant's GitHub public profile information and return that with GET requests to the applicants collection.



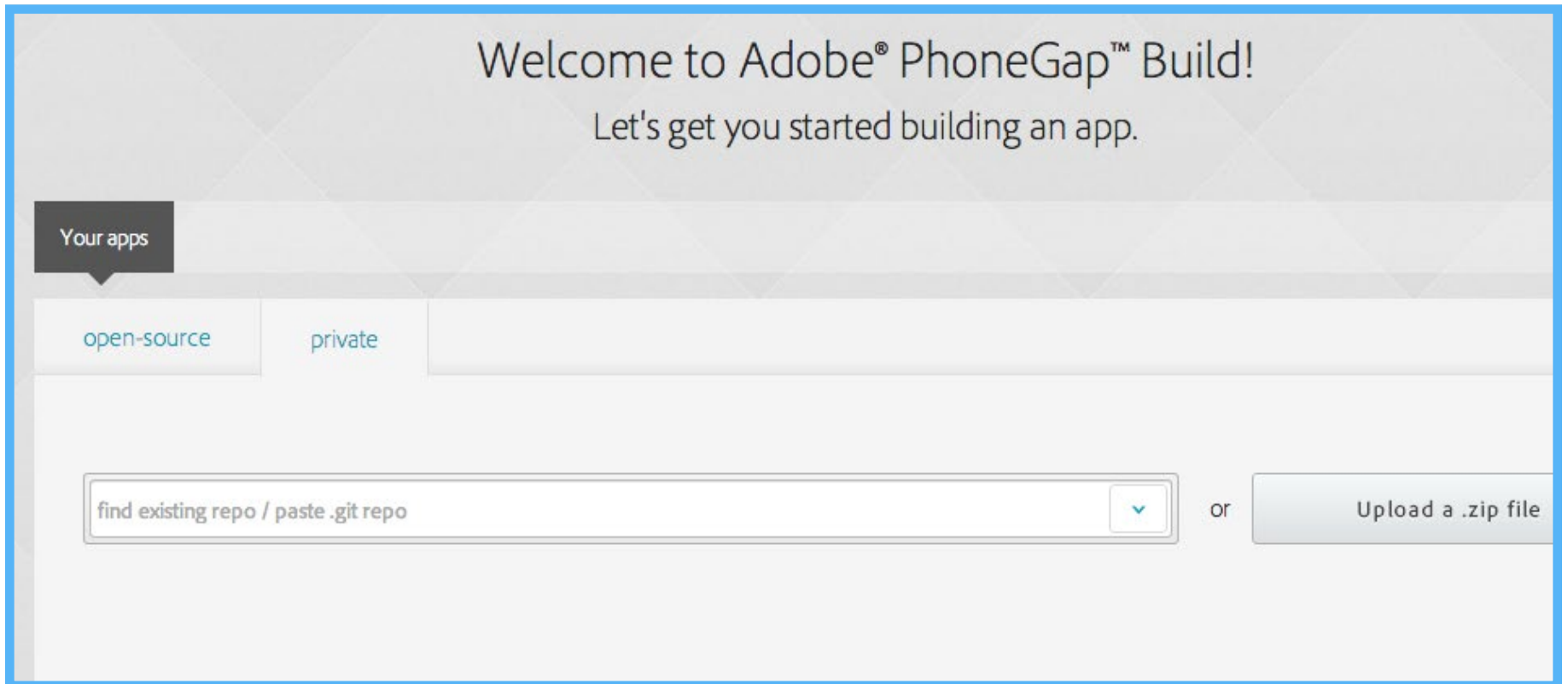
1. Click on **Addons -> Business Logic -> Collection Hooks** to add some backend code to your app.
2. Copy the contents of **backend/applicants-post-fetch.js** into the post-fetch [custom code](#) for the applicants collection, making sure to add your GitHub API keys to the top of the file. (Note: you will need to have already obtained your GitHub API keys using your GitHub account.)

Step 2 is now complete and you are ready to generate the mobile versions of the app.

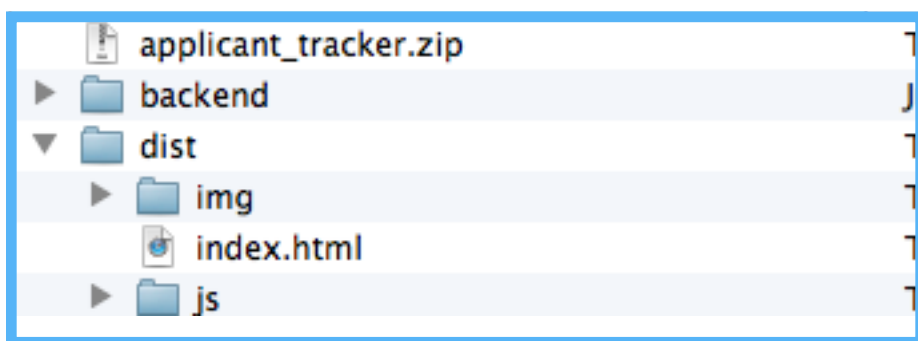
Step 3: Sign up for PhoneGap and Generate Native Mobile Apps

Create a new PhoneGap app and generate native apps

1. Create an account at the PhoneGap website <https://build.phonegap.com>
2. Upload the **applicant_tracker.zip** to create a PhoneGap app.

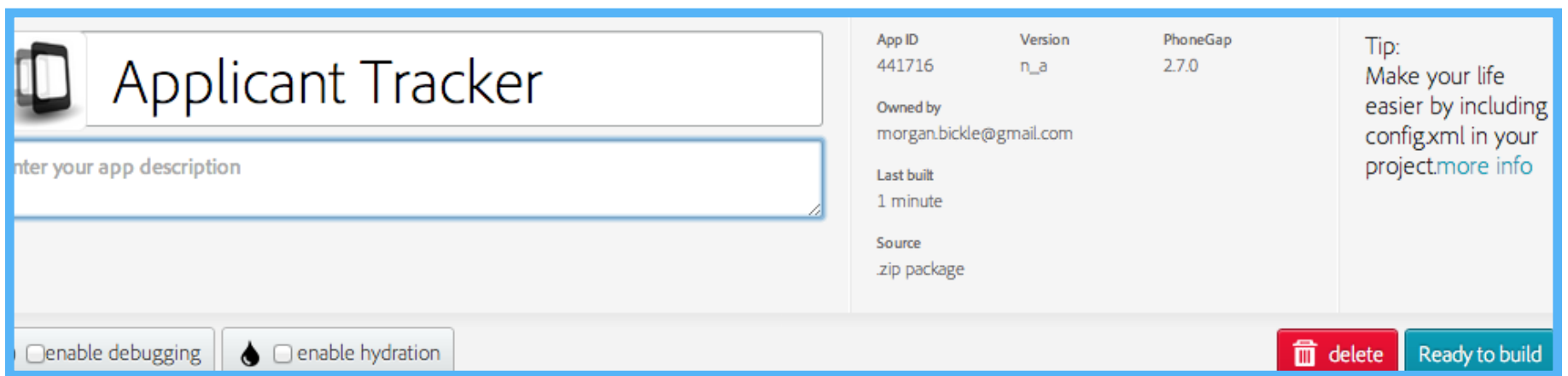


Browse and select the file from the project directory.

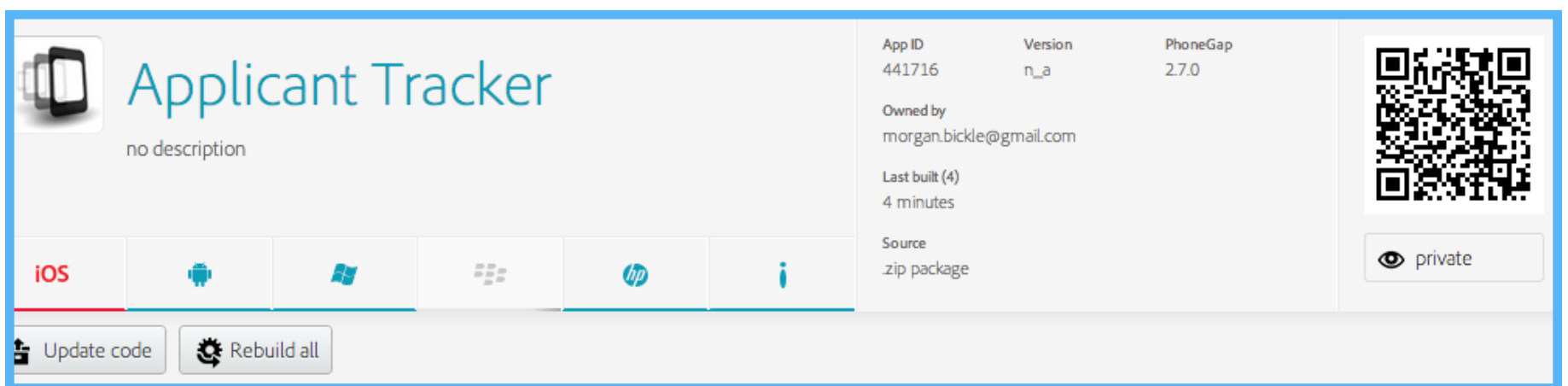


continued on next page

3. After the upload is complete, click the Ready to build button



4. To install the app on Android, ensure the Developer Options are enabled for the device and scan the QR Code with your mobile phone.



The app will now install automatically. When it is finished, you can tap the app's icon on the home screen to run it.

Congratulations! You have just built and deployed an HTML5 app on your mobile device!

Written By

[Randall Cronk](#)

Since 1990, Randall has helped over 250 high-tech companies convey the value of what they do through white papers, web content, brochures, case studies and articles. Based today in downtown Boston, he was previously a vice president with Regis McKenna where he ran the Digital Equipment account.

[Morgan Bickle](#)

On any given day you'll find Morgan programming mobile SDKs, building backend APIs, and discussing the future of web and mobile technology. As part of the core founding team at Kinvey, he owns Kinvey's technology vision. Prior to Kinvey, he wrote enterprise software for a decade. He held various technology roles at AMD, where he gained a wealth of experience designing and deploying systems in the areas of big data warehouses, web applications and real-time data intensive computing clusters.

Project Managed by

[Kelly Rice](#)

Designed by

[Jake McKibben](#)



What is Kinvey? Kinvey makes a fully-featured Backend as a Service solution, offering 3rd party data integrations, multi-platform support, push notifications, and custom business logic on a platform where it's free to get started and you only pay when your app is successful.

Build your backend today

