Step-by-Step Guide to Publishing HTML5 Mobile Application on App Stores

If you're reading this you would probably already have a decent idea of what an HTML5 mobile application is, but you might not know **how to actually get one on the Apple App Store or Google Play**.

You can submit an application built entirely with web technology to both of these stores just like you would a native application (even if you don't have a Mac), and for the majority of use cases **it can look, feel and perform just as well as native applications** - as long as the application is designed well.

Users of mobile applications have come to expect a certain level of quality and design from the applications they use. Android and iOS applications have a set of norms that - if you stray to far from them - can make your application look amateurish and bad (or as many people might say "not native"). A problem with web tech is that it has a low barrier to entry and it will let you do anything, even if what you end up building is nowhere near what a mobile application "should" look like. This has given HTML5 mobile applications a bit of a bad rap, since it is so easy to create mobile applications with HTML5 web technologies, naturally there are a lot of bad ones out there.

If you pay attention to design/quality and spend an appropriate amount of time learning *how* to build good mobile applications with web tech, then you can create high quality applications indistinguishable from standard native applications in most cases (another problem with the reputation of HTML5 mobile applications: people assume the good ones are just "normal" native applications!).

There are a few steps between *here* and *there*, though. There are a few conceptual differences to understand when creating an HTML5 mobile application, and you can easily get stuck somewhere along the way.

I've created this step-by-step guide to point you in the right direction - no matter where you are along your app store journey, this article should help give you the context you need.

1. Understand the Difference Between Native, Hybrid and Web Applications

Before you even get started, you should understand *why* you're building a mobile application with web tech and what the difference is exactly between a **web** app, and what is commonly referred to as a **hybrid** app, and a standrd **native** app.

If you're a little short on time, here's a little infographic I made that highlights the appeal of the hybrid approach in a **very simplified** way. Please note that there are **a lot** of factors not included in this infographic, and a hybrid approach is not always the best solution:

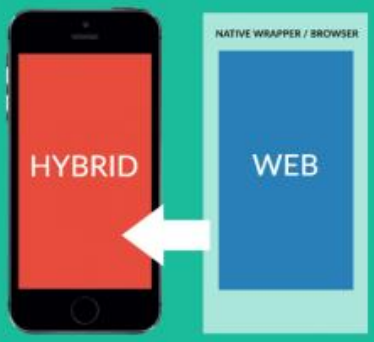
The Difference Between NATIVE, WEB & HYBRID MOBILE APPLICATIONS

Native applications are coded in the native language of the device (e.g Objective C for iOS, Java for Android). They are run directly on the device.

- ✓ Access Native APIs
- ✓ Distribute through App Stores
- ✗ Run on multiple platforms

Web applications are coded in HTML, CSS and JavaScript. They are served through the Internet and run through a browser.

- ✗ Access Native APIs
- ✗ Distribute through App Stores
- ✓ Run on multiple platforms



Hybrid aplications are coded in HTML, CSS and JavaScript*. They are run through an invisible browser that is packaged into a native application.

- ✓ Access Native APIs
- ✓ Distribute through App Stores
- ✓ Run on multiple platforms

*There are other ways to create hybrid applications, but this is the most popular

joshmorony.com

Keep in mind that the purpose of this image is to highlight the benefit of the hybrid approach, I'm not attempting to say "Look! The hybrid approach has 3/3 ticks in these areas and is, therefore, the best solution possible".

It's easy enough to understand what a **native application** is: an application that was coded using the native language of the platform. This means using Objective-C/Swift/XCode for iOS, or Java/Kotlin/Android Studio for Android. A native application has immediate access to everything that platform has to offer with no restrictions whatsoever. The downside to this approach is that separate applications need to be built for each platform.

A **web application** is an application that is powered by a browser (typically loaded through the Internet). A web application is typically **built using HTML, CSS and JavaScript** and can be served either through a desktop or mobile browser. A web application can be built to mimic the feel and behaviour of a normal native application, but instead just runs through a browser. Using this approach means you won't have access to all the bells and whistles a native application does (i.e. Native APIs) and you **can't distribute your application through most app stores**. However, **building Progressive Web Applications** – which basically just means a modern/offline-capable mobile web application – is becoming a very popular distribution method.

A **hybrid application** is a mix of the native and web approach that allows you to build the application once and submit it to multiple app stores. A hybrid application uses a native packager like **Cordova** or **Capacitor** which essentially wraps up a browser web view into a native application and displays your web application through it. With this approach, users of the application will no longer be able to see the browser the app is being run through (so it will just appear like a normal native application to them), you have access to native functionality through Cordova/Capacitor and you can distribute your application through app stores.

The most obvious question that might arise from this is:

If Hybrid applications are so great, and provides the best of both worlds, why isn't everybody using them?

To drill down into all of the differences between a hybrid and native approach would take a massive undertaking. But in short, the main benefit of building native is that it offers the best possible performance and feature set. You aren't dealing with additional abstractions, tools, or limitations of any kind.

This doesn't mean that if you build natively instead of with a web based approach that your application will inherently just be better. In most cases, the choice just comes down to personal preference, and the skills that are available in your team. Most requirement sets could be satisfied by either a native or hybrid approach. Whilst native applications have the potential for much higher performance, in most cases, it doesn't make any noticeable difference (again, assuming that the application is designed well). There are circumstances where a hybrid approach is not viable, and some where it isn't ideal/feasible, but these are few and far bettern. Just about any business style application could be built with web tech. If you are looking into designing an application that relies heavily on processing power, games, 3D graphics, or that integrates heavily with native features you would need to be more careful about your decision. Most "typical" applications can easily be satisfied with web tech.

I often use the analogy of purchasing a high-performance supercar for city driving. If you're never driving faster than 60km/ph it will hardly make a difference that your car *could* travel at 300km/ph. It never needs to, so it doesn't matter. People often fall into the trap where they build something that doesn't perform well, and think that if they had access to more power (e.g. if the application was fully native instead) then it would perform better. It doesn't matter how much performance you have, if something is designed poorly it will still perform poorly - no matter whether you are using web tech or native code.

If you are interested in more of my thoughts on the web based approach to building mobile applications, take a look at **my home page** or my article on **the performance of Ionic**.

You will find a lot of opinions on what approach is the best for creating mobile applications between options like:

- Native iOS/Android
- Ionic
- React Native
- Xamarin
- Flutter
- NativeScript

...and more. Ultimately, all are perfectly capable of creating great applications in the right circumstances and with the right people. The only reasonable answer to which approach is the best is always "it depends". This article is specifically about using a hybrid/HTML5 approach to building mobile applications, so I will be focusing on aspects related to that.

2. Design Your Application

Before committing to a particular approach, make sure you have your **requirements defined** and that your approach is going to meet those requirements.

If you would like to create a **progressive web application** (an application that runs entirely through the Internet, rather than being distributed through native app stores), then you only have access to features that are available to the browser. Make sure all of the functionality you require is available **here**. Don't worry if you later decide to include native functionality, it's easy to convert your PWA to a hybrid app and distribute it through the native app stores.

If you want **access to some native API's** or want to **submit your application to app stores** (or both) then a **hybrid approach** is going to be your best bet. The same general process for creating the application is the same as a normal web application/PWA, except that we wrap it up in a native package. We'll talk more about integrating Cordova/Capacitor to access native API's and wrap your application later, but make sure the functionality you require is available as a **Plugin API in Cordova** or **Capacitor**, or that there is a 3rd party Cordova/Capacitor plugin available that provides the functionality you need (anybody can **create their own Cordova/Capacitor** plugin that accesses native functionality).

When designing your application keep in mind the **iOS Human Interface Guidelines** and **Android Design guidelines**. Keep in mind that some of the information in these guidelines will be specific to native applications, so don't worry too much about it. Just make sure you follow good design principles that are prevalent in most mobile applications - your application should look/behave/feel similar to standard native applications.

If you're building an application to be distributed through the Apple and Google Play app stores, **you need to play by their rules**. Make sure you comply with the **Apple App Store Review Guidelines** and the **Google Play Developer Program Policies**. This is where **the benefit of distributing your application as a Progressive Web**

Application comes in – using the open web means we don't need to jump through somebody else's hoops.

Apple especially can be a bit fickle with their application of the guidelines, but if you break any of them (sometimes even if you don't) **your application may be rejected** and you will have to fix the problem and resubmit. Here are a few *gotchas* to watch out for that might see your application be rejected:

- Your app is more like a web page than a mobile application
- You don't handle online/offline states
- Your app is slow or unresponsive
- Your application doesn't make use of native features (i.e. there isn't really a reason for it to be distributed as a native app instead of a standard website)

Apple is generally more strict than Google when it comes to the review process.

3. Build Your Application Using a Framework and a UI Kit/Design System

HTML5 UI Kits/Design Systems (like **Ionic**) and frameworks (like Angular, StencilJS, React, and Vue) are the corner stone of developing high quality HTML5 mobile applications. Attempting to build out an entire mobile application from scratch on your own is essentially pointless and futile. You would spend an enormous amount of time trying to replicate a mobile user interface - with all of the smooth animations, interactions, and screen transitions that involves - and chances are, you wouldn't get close to the quality you can get from Ionic out of the box. They've already done a lot of the hard work for you, no need to re-invent the wheel.

Ionic provides a set of web components for building out the user interface of an application - this includes common elements like lists, buttons, tabs, navigation bars,

inputs, and so on. With Ionic, to create a typical mobile list that performs well and scrolls smoothly, all we have to do is drop something like this into our template:

```
<ion-list>  <ion-item> Apple </ion-item>  <ion-item> Banana </ion-item>
<ion-item> Coconuts </ion-item>  <ion-item> Durian </ion-item>  <ion-
list></ion-list></ion-list>
```

...and the work is done. If you were to try and recreate the behaviour of a **native smooth scrolling list with acceleration and deceleration** by yourself you would likely not achieve a good result without an enormous amount of effort – and there is a good chance your application may end up getting rejected from app stores. With Ionic, this is all built-in.

However, there is more to building out a mobile application than just the user interface. *Things* need to happen. When a user clicks a button we might want to kick off some process, or navigate to another screen. We might want to store data, and load that data in later. There is all sorts of *logic* that we want to take place in our application.

The Ionic web components are not just aesthetic, they also have a lot of functionality built into them that we can interact with. We still generally need something more than just Ionic or any other UI Kit/Design System to manage building out an application though. We need something to manage the overall architecture and logic/functionality of the application - the behind the scenes sort of stuff. Ionic and similar systems are *primarily* designed to handle the interface.

To handle everything else, we would typically combine our UI Kit/Design system with some kind of JavaScript framework.

Although Ionic was once built specifically for the Angular framework, it is now completely framework independent and can be used with just about anything.

For a lightweight approach, you can actually **build an entire Ionic application using the Ionic teams own StencilJS** - this gives you a lot of the benefits of a framework, without actually needing to use a traditional framework to build your application. StencilJS is not like a traditional framework, it is a "web component compiler", so the final build of your application mostly just runs on standard functionality built into the browser, rather than requiring a large runtime from a framework (StencilJS has just a 6kb minified and gzipped runtime).

On the other end of the spectrum, you can **use a framework like Angular to build an Ionic application**. Angular is what was originally used to build Ionic applications, and it is still the most popular choice today. Although Angular is a "heavier" and more

opinionanted framework, it provides just about everything you need out of the box and is well suited to large/complex/enterprise applications. The learning curve for Angular can be a little steep as there is quite a few Angular specific concepts you will need to learn, but once learned, you will have a very powerful tool at your disposal.

As I mentioned, Ionic can be used with just about anything - so if you have a preferred framework you can probably use it. Ionic is also now building out support specifically for **React** and **Vue**.

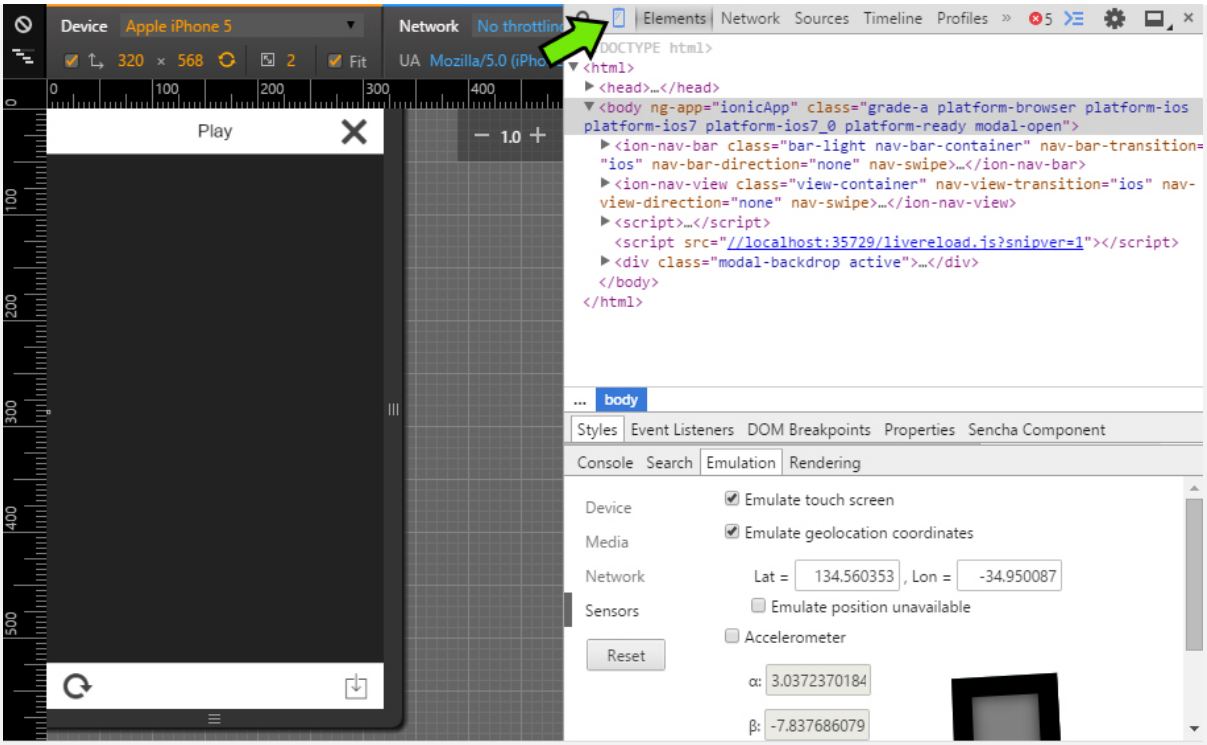
4. Test Your Application Through the Browser

The great thing about developing mobile applications with HTML5 is that **you can test it right through your browser**.

Different frameworks may be run in different ways but typically you will be able to access your application through a local web server by going to:

```
http://localhost:8100
```

in your browser. If you're using Chrome Dev Tools you can even emulate the device you are designing for by clicking the mobile icon in the top left:



From here you can debug your application to your hearts content using the various debugging tools provided by the browser.

If you're building a web application/PWA, then once you've finished this stage your app is all good and ready to go. You would just need to deploy it to the web. But if you want to create a **hybrid application** that can access native API's and be submitted to app stores, it's now time to package the application.

5. Package Your Application

You can use a technology like **Cordova** or **Capacitor** (built by the Ionic team) to wrap your web application in a native wrapper and act as a bridge between your application and the Native APIs of the device.

If you want to build iOS applications **but don't have a Mac** you can use a service like **PhoneGap Build**, or Ionic's own **Package** service. To build for iOS you need the appropriate SDKs installed on your machine, and you need a Mac for that, however, if you didn't want to use one of these build services you could also do something fancy like setting up a virtual machine to provide access to the necessary macOS software.

The basic role that Cordova/Capacitor performs is this:

- Cordova/Capacitor creates a native application that contains a web view
- All of the resources for your application are stored inside of this native application
- Cordova/Capacitor loads your web-based application into the web view (the web view cannot be seen by the user)
- The web view displays your application to the user

Our application is really still just a web application since it is being powered by the browser, but now we also have access to the device through Cordova/Capacitor, and it can be installed like a normal native application when distributed through app stores.

Enjoying learning about hybrid apps? Consider retweeting to share with others 😀

Read my "big picture" overview of what a "hybrid" application is and the steps required to publish them: ***https://t.co/dfqhBUOpWB***

```
<p>  &mdash; Josh Morony (@joshuamorony) <a
href="https://twitter.com/joshuamorony/status/973328711775764480?ref_src=twsr
c%5Etfw">March 12, 2018</a></p>
```

6. Access Native APIs (optional)

You can just use Cordova/Capacitor to package your application so that it can be submitted to app stores, but why stop there? We have access to all this magical native functionality now!

With Cordova or Capacitor, we can access just about everything that a native application can. Rather than interacting with the device directly, we could use Cordova to pass messages between the device and our application.

Typically, the native functionality will be available through some **global Javascript object**. We will use the **Cordova Camera API** as an example. To trigger the camera all we need to do is run the following bit of JavaScript:

```
navigator.camera.getPicture(onSuccess, onFail, {  quality: 50,
destinationType: Camera.DestinationType.DATA_URL,});function
onSuccess(imageData) {  var image = document.getElementById('myImage');
image.src = '' + imageData;}function onFail(message) {
alert('Failed because: ' + message);}
```

This tells Cordova to tell the device to launch the camera, and then we can grab the resulting image data from the **onSuccess** function. In this case the functionality is available through the **camera** object which can be found at **navigator.camera**. Sometimes though, like in the case of the **AdMob plugin**, the functionality will be available directly through an object that is available globally like:

```
admob.someMethod();
```

If you were using Capacitor, then interacting with the **Camera API** might look more like this:

```
import { Plugins, CameraResultType } from '@capacitor/core';const { Camera }
= Plugins;async takePicture() {  const image = await Camera.getPhoto({
quality: 90,    allowEditing: true,    resultType: CameraResultType.Uri  });
console.log(image.webPath);}
```

7. Test Your Application on a Device

I mentioned before that one of the great things about HTML5 applications is that you can test them through your browser. Once you start integrating Cordova/PhoneGap/Capacitor and native API's though **testing through the browser isn't so great**.

To test an application that is accessing native API's you will need to run the application on an actual device. But how do you debug without your browser debugging tools? Don't fret! You can still debug directly on your device and access the same browser debugging tools, you can check out the videos below for more information on how exactly to do that:

- **Debugging Ionic Applications When Deployed to an Android Device**
- **Debugging Ionic Applications When Deployed to an iOS Device**

It's an important step, in any case, to test on a device, whether you're using native functionality or not. The behaviour on an actual device can be different than when it is viewed through a desktop browser.

8. Sign Your Application

To distribute your applications on app stores you will need to "sign" your application, and in the case of iOS you need to sign your application before you can even install it on a device.

For an iOS application you will need to **create a .p12 personal information file and a provisioning profile**. You can more easily sign an iOS application if you have a Mac and XCode but it is also possible to sign an iOS application without a Mac (the article I linked above describes how to do that).

To sign an Android application, you simply need to **create a single keystore file**. The Ionic documentation has some examples of how to get through the **signing process**.

If you want to use Capacitor to distribute your applications, I have a complete guide to signing/distributing available here:

- **Deploying Capacitor Applications to Android (Development & Distribution)**
- **Deploying Capacitor Applications to iOS (Development & Distribution)**

9. Distribute Your Application

Assuming that you want to distribute your application in the native app stores, you'll need to sign up for the **iOS Developer Program** (in fact, you'll have to do this to create your provisioning profile and such anyway) and as a **Google Play Developer**.

If you want to publish your application as a Progressive Web Application, you do not need to do this (and you don't need to worry about signing your application either).

You will need to prepare your app store listings for both platforms and provide some details about your application. To submit to Google Play you will simply be able to upload your **.apk** file through the developer console. To submit to the Apple App Store without a Mac though you will need to use the **Application Loader** program or **XCode**, which are only available on Macs. You can **use a service like Macincloud.com to get around this**, though.

Summary

I hope that this article has been able to give you some context around the path to building and deploying an HTML5 mobile application to the Apple App Store and Google Play. There are a lot of steps in between, of course, and this has just been a very high-level overview.

Courtesy: https://www.joshmorony.com/the-step-by-step-guide-to-publishing-a-html5-mobile-application-on-app-stores/

Modified: 2021.10.06.8.05.PM

Dököll Solutions, Inc.