IONIC Whitepaper

Building cross-platform apps with Capacitor

IONIC Whitepaper

CONTENTS

INTRODUCTION

Capacitor was first released in 2019 as a new and revolutionary way to deliver native mobile apps and Progressive Web Apps (PWAs) from the same codebase, using web technology in place of native programming languages.

The Ionic team created Capacitor as a spiritual successor to Apache Cordova and Adobe PhoneGap, with inspiration from other popular cross-platform tools like React Native and Turbolinks, but focused entirely on enabling modern web apps to run on all major platforms with ease.

At Ionic, we have a long history with these solutions. For years, Cordova ran underneath the hood of every app shipped on our platform. While Ionic focused primarily on the UI or frontend mobile experience, we left the native layer to Cordova. But over time, our aspirations and visions for a native runtime outpaced what Cordova was capable of. That's due in no small part to the fact that many of the new, modern APIs were not available previously.

Eventually, we reached a fork in the road, and decided it would be better to forge our own path, rather than iterate on top of what Cordova had already built. It was a tough decision; but looking back, I have no doubt that it was the right one.

While Capacitor shares many similarities with Cordova and other native runtimes - and owes a tremendous debt to what they pioneered - we've made some very different decisions at several key points, such that the experience of the projects are very different.

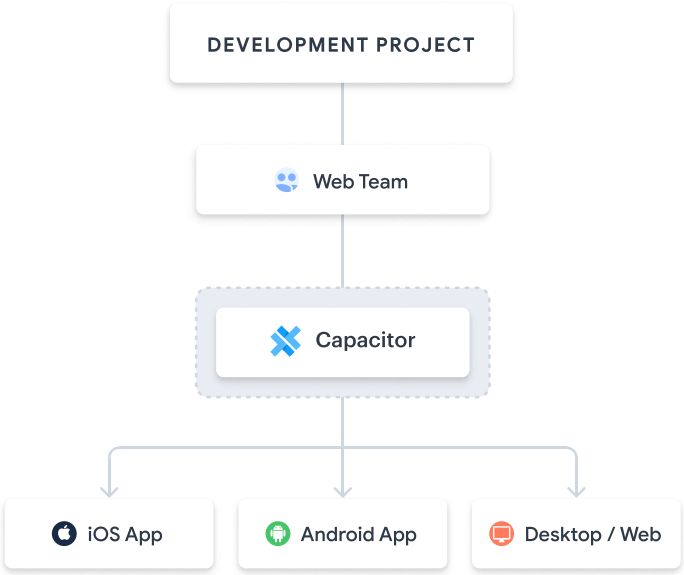


Figure 1 - Project flow with Capacitor

For example, Capacitor embraces npm to manage plugins, Swift for iOS, Java for Android, TypeScript, and has full support for Progressive Web Apps. Capacitor makes it possible to build one app for iOS, Android, and the Web, all with one code base. Capacitor has first-class support for building plugins; in fact the plugin authoring experience has been a focus from the very beginning. Capacitor treats native projects as source-artifacts, meaning native code can quickly be added without worrying about changes being lost. Capacitor does away with device-ready or async plugin loading, a common source of confusion and bugs with Cordova developers, so all plugins are available as soon as the page starts loading. And that's just the tip of the iceberg.

Today, Capacitor is the most cost effective path to deploying a single app to multiple platforms.

Don't just ask us! Since its launch, Capacitor has grown to more than 250K monthly installs and is poised to eclipse Cordova as the #1 native runtime for hybrid apps. In other words, developers really like Capacitor. We're thrilled to report some notable companies using Capacitor include Tim Horton's, Popeye's, and Sworkit, to name a few.

Building on the initial success of Capacitor, Ionic released Capacitor 2.0 in April 2020, and we have big plans for the next version - stay tuned!

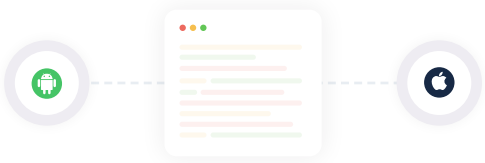
— Max Lynch, Co-Founder and CEO of Ionic

The emergence of cross-platform development

Today, about 60% of all apps in the app store are cross-platform. This means apps that run on iOS and Android from a shared codebase, using cross-platform solutions like React Native, Ionic, and Cordova. And increasingly, Capacitor.

Given the thousands of apps released in the app stores each year, and the prodigious appetite for mobile inside many businesses, cross-platform development approaches have emerged as a way to meet the demand for mobile applications, by speeding up and simplifying how development gets done. They do this in a few ways.

First, instead of writing separate code for each platform, cross-platform solutions allow you to share code across platforms. You write your code once and it works seamlessly across iOS and Android. That benefit alone can lead to time savings of 50% or more.



Secondly, by abstracting much of the native complexity, these solutions have made it so that just about anyone with a basic knowledge of JavaScript, HTML, and CSS can now build a high-quality mobile app.

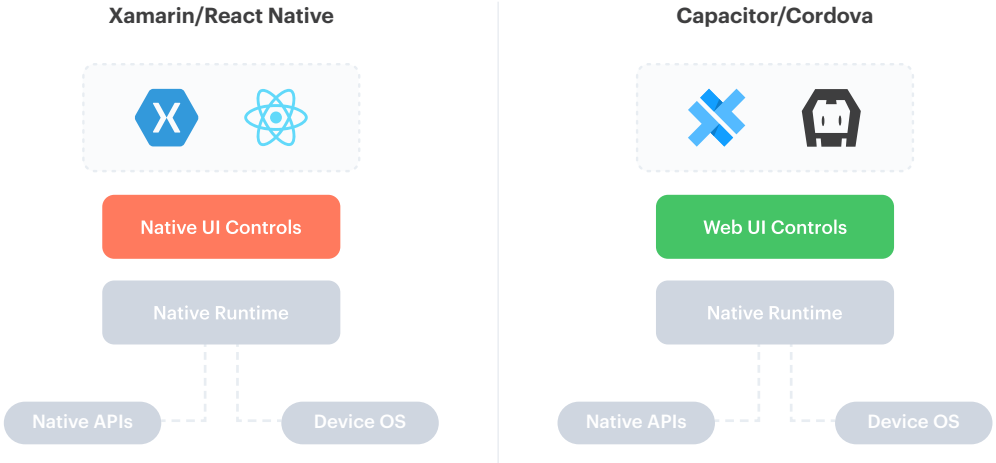
For background, check out our ebook:
Hybrid vs. Native: A Comparison Guide.

Building cross-platform apps with the web

Within the cross-platform category, there are two main categories of solutions.

So-called “cross-platform native” solutions like React Native and Xamarin are tightly coupled with the native mobile platforms. Written and controlled largely in JavaScript or C#, they boast solid performance and allow developers to build mobile experiences with native UI controls.

In contrast, Cordova and Capacitor are primarily web-based in nature — often referred to as “hybrid” solutions, in reference to the fact that they combine native runtime features with a web UI layer. Although hybrid apps run natively on the device and interact with any available native APIs, the UI layer of the app is executed primarily in a web browser, known as a WebView, that is invisible to the user.

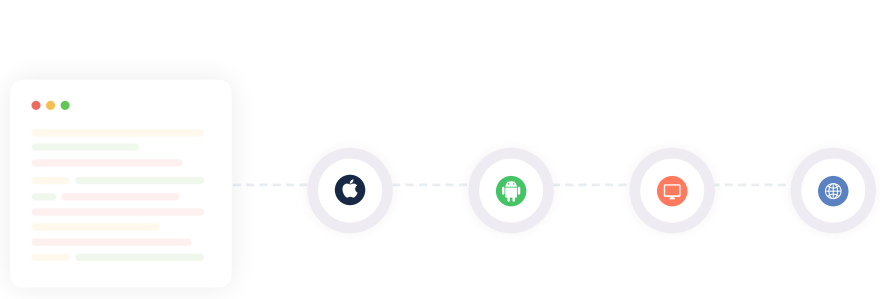


What are the advantages of a hybrid approach?

On top of all the general cross-platform benefits that we noted in the section above, hybrid offers a few key advantages over the "cross-platform native" solutions:

- If you're already familiar with the traditional web languages and approaches, you'll be right at home. There are no additional languages to learn.
- If you have an existing web project that you want to bring to mobile, you can do that using a native runtime like Cordova or Capacitor.
- If you have a web UI library that you'd like to use in mobile - whether it's a corporate design system or a packaged UI library like Bootstrap - you can bring those UI components to mobile using a hybrid approach.
- If you're building an app that you'd like to deploy across mobile and web, including PWAs, then a hybrid approach makes sense.

On top of that, because the UI of a hybrid application runs entirely in the browser, it's possible to target iOS, Android, desktop and the web, all from a single shared codebase. That's in contrast to cross-platform native solutions that often require separate native code on each platform, along with a third "control" layer for any shared code across platforms.



Of course, some will debate whether running the UI in a browser adds a big performance hit, but the truth is that advances in native devices, browser technologies, and runtimes like Capacitor have rendered these issues largely obsolete. For proof, just download one of the many successful hybrid apps on the market, including Shipt, Sworkit, Sanvello — all highly rated consumer apps serving millions of users.

Capacitor: A new approach to cross-platform development

Capacitor is an open source project that runs modern Web Apps natively on iOS, Android, Electron, and Web, while providing a powerful and easy-to-use interface for accessing Native SDKs and Native APIs on each platform.

Capacitor delivers the same benefits of the native runtimes that preceded it, including React Native and Cordova, but with a modern, web-first approach.

What makes Capacitor unique?

Progressive Web App (PWA) support

Capacitor is the only native runtime to deliver first-class support for web apps and PWAs. In fact, we've taken it a step further by building out a collection of UI experiences, like Camera, that bring the native app UI experience users expect to PWAs.

Also, building plugins that offer web functionality as a fallback simply requires adding a few files to your plugin. Capacitor will only use your web implementation if the native one is not available, so users can use the exact same API when running on iOS, Android, Electron, and the web. Here is the Capacitor Camera API, for example. Notice there is no platform-specific code:

```
import { Plugins } from '@capacitor/
core';
const { Camera } = Plugins;
const photo = await Camera.getPhoto({
  quality: 100,
  resultType: CameraResultType.Uri,
  allowEditing: true,
  saveToGallery: true
});
```

Figure 2 - One API, one codebase

CLI tooling that is version managed per app

Capacitor provides a small CLI tool that is installed locally to each app. That means there are no global dependencies to manage and it's easy to use different versions of Capacitor across every app you build. This is a boon to teams that are building multiple apps with potentially differing dependency versions or version management processes.

Native and web developer collaboration

Since Capacitor apps are actual native apps and a key design consideration of Capacitor is embracing native tooling, Capacitor enables teams that have both traditional native mobile and web developers to collaborate on mobile app projects. Traditional native mobile developers can use their programming languages of choice (Swift/Objective-C on iOS, Java/Kotlin on Android) to build UI experiences or business logic and then expose them to the Web layer through Capacitor's JavaScript-to-native APIs. This also ensures an app team never gets stuck implementing the functionality it needs.

Cordova, in contrast, works through an abstraction layer that manages the underlying native platform project and source files for you, making it harder to drop down to native code or work with a traditional native mobile development process, and can result in custom changes being lost.

Capacitor takes a "web first" approach

We believe that the core of every great cross-platform hybrid app is a quality, modern Progressive Web App (PWA). This keeps your app aligned with the rapidly evolving web platform while enabling powerful native device functionality on platforms that support it. Capacitor has even convinced native developers or users of alternative platforms that see the benefits of web development on mobile, and customer feedback indicates that Capacitor is finally providing the development experience they prefer.

Easily access native device features

Developers have access to the full Native SDK on each platform, and can easily deploy to App Stores, and the web. Adding native functionality is easy with a simple Plugin API for Swift on iOS, Java on Android, and JavaScript for the web.

Capacitor apps access native and web features using a series of platform APIs (AKA plugins). While the base Capacitor experience includes a series of core plugins, along with OSS contributed plugins, you can easily add your own custom device functionality by writing a plugin of your own.

Using Capacitor, developers can build one app and target a single set of APIs regardless of the platform the app is running on, as opposed to managing multiple APIs for each target platform. As illustrated in the example above, accessing the Camera uses the same code on iOS/Android as it does on Electron and on the web.

When to Use Capacitor

Capacitor is a versatile solution that addresses several key use-cases related to mobile and web application development.

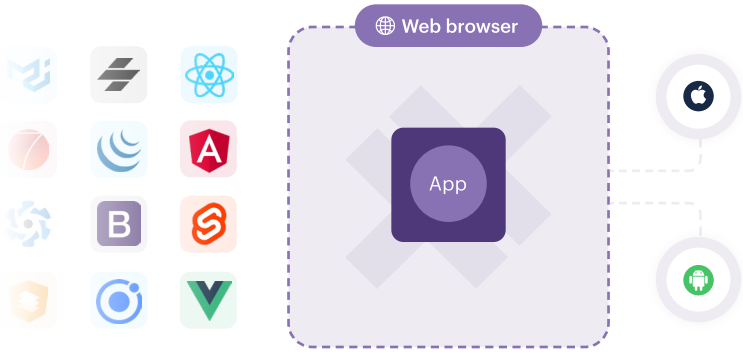
Building a native mobile app on iOS and Android

Capacitor makes it easy to build and deploy native mobile apps for iOS and Android, using familiar web languages, libraries, and frameworks, without the underlying complexity of the native SDKs and iOS and Android specific code. Mobile applications built with Capacitor are optimized for native performance and have full access to any native device features, including full access to the native SDKs when needed. The UI layer of any Capacitor mobile app primarily runs in the browser, so that any new or existing web applications can be deployed as a native mobile application.

Building a Progressive Web App

Capacitor has first-class support for Progressive Web Apps (PWAs) and native apps. That means that Capacitor's plugin bridge supports running in either a native context or on the web, with many core plugins available in both contexts with the exact same API and calling conventions. This means you'll use @capacitor/core as a dependency for both your native app and your Progressive Web App, and Capacitor will seamlessly call web code when required and native code when available.

Capacitor also offers a number of utilities for querying the current platform to provide customized experiences when running natively or on the web.



Adding a web-based experience to an existing native mobile app

Capacitor adds superpowers to the traditional WebView control available on each platform, and it was designed to be used anywhere a WebView would be used. That means it can easily be dropped into an existing native app codebase, making it possible to build certain screens of the app using web technology, without the need to make any sweeping changes to the rest of the app codebase.

This is also a great way to let other teams that may have more of a traditional web development skill set participate in the development of the app without getting in the way of a traditional native development process.

Deploying a legacy web app to mobile

Traditionally, teams looking to extend their existing web apps to other platforms were faced with a conundrum: either rewrite the app using native technologies or risk excluding new customers (the majority of which are obtained through mobile experiences these days!). Both decisions involve dropping years of time, money, and effort invested or risking future company growth.

With Capacitor, there's no need to start from scratch. In fact, it was designed to drop into any existing modern JavaScript web app - including ones using UI frameworks like Bootstrap, Material UI, or Framework7. Simply install Capacitor, add the desired native platforms, then start using any of Capacitor's 23+ cross-platform APIs. Yes, you'll need to address several mobile-specific UI paradigms - more on that below.

To see how to use Capacitor with your web app framework of choice, check out the examples in this code repository.

What about the User Interface (UI) layer?

While Capacitor gives you the native runtime environment to run your app on mobile, you still need to think about the UI layer of your app. Here are a couple options to consider.

Bring your own library or framework

It's important to note that the UI of a Capacitor app runs primarily in the browser. This small but important distinction opens up a whole world of possibilities for web developers and teams who want to bring their web UI components and applications to mobile.

It also means any popular UI framework like Bootstrap, Material, or Tailwind will work great in Capacitor. While you'll need to anticipate and address the many mobile-specific UI paradigms (see below), your existing web-based library will now run natively on any iOS or Android device, and on the web as a PWA.

Use a mobile-ready UI framework like Ionic

While Capacitor addresses the feasibility of bringing your web apps to mobile, there are other hurdles to consider. Mobile styling, navigation, and performance on mobile devices are all very tricky and can be daunting for the uninitiated.

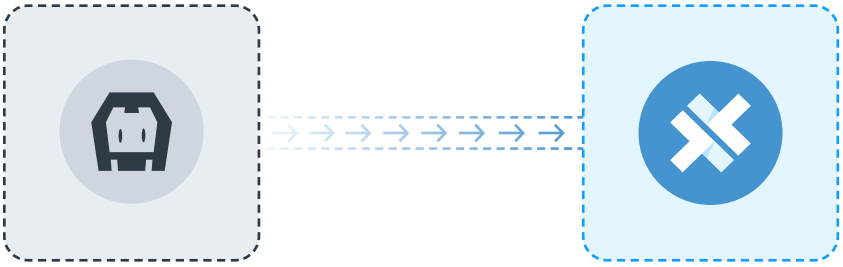
As an alternative to rolling your own solution from scratch, you can use a mobile-ready UI library like Ionic Framework instead. You can then incrementally add your own custom components as needed. Or, start with your UI library as the foundation, and incrementally add Framework components as you need them. There are a variety of options, and it's not an all-or-nothing proposition.

Using Capacitor in an enterprise environment

Capacitor is truly unique as an open source project in that it has a real company behind it. That company, Ionic, is in the business of enterprise app development. That means our sole focus is to provide teams building mission-critical apps the support and enterprise-specific functionality they need to be successful. Ionic, the team behind Capacitor, works with hundreds of enterprise customers, building everything from major consumer transportation apps, fast-food retail apps, internal B2E apps, and a whole lot more.

Ionic offers dedicated support across Capacitor and its native APIs, as well as advanced enterprise native features like secure token management, encrypted offline storage, and OpenID-powered single sign-on. Additionally, teams can build best-of-breed user experiences on top of Capacitor using the highly popular Ionic Framework.

Capacitor is a key aspect of Ionic's mobile infrastructure and the infrastructure of its customers. That means security and functionality issues are discovered, fixed, and released quickly. Capacitor has a full-time team of engineers, product management, marketing, and customer success experts dedicated to customer and project success.



Migrating to Capacitor from Cordova

If your team is currently using Cordova and curious about moving to Capacitor, then you will welcome the fact that Capacitor has backward-compatible support for a large swath of existing Cordova plugins. Migrating to Capacitor from Cordova is well-documented and uncomplicated.

If you're interested in learning more about our enterprise options, reach out to one of our team members to schedule a free Strategy Session.

Adopting Capacitor

Capacitor is a key aspect of Ionic's mobile infrastructure and the infrastructure of its customers. To date, Capacitor is installed over 1.5 million times a year, and is currently powering major production enterprise apps with hundreds of millions of users. Suffice to say, Capacitor is now a major force in the mobile development ecosystem and is a major element of Ionic's enterprise business. Even native developers (or alternative platforms) are recognizing the benefits of web development on mobile, and are satisfied with the development experience Capacitor provides.

"Capacitor's support for the latest in security, performance, and native platform capabilities, makes it easy to build compelling, modern app experiences that our users want, without having to worry about all the underlying complexity of the native SDKs and iOS and Android specific code."

— Rakesh Gadapa, Application Developer III at Blue Cross Blue Shield of Michigan.

Interested in learning more about Capacitor, enterprise support and advanced integrations, or the Ionic platform in general? Get in touch, we'd love to see how we can help your organization build market-leading apps in a fraction of time compared to traditional app development.

Book a strategy session with a Solutions Engineer today.