Mobile App Technology Report 2022

Mobile has always been about innovation, modernisation and enablement for consumers and brands alike. It has not-so-slowly but surely changed the way we live and interact with the world around us for the better.

At Kin + Carta, we've consistently been at the forefront of digital for over 20 years now. We leverage data, technology and experience to put more and more brands on the path towards sustainable digital transformation; the Mobile App Technology Report 2022 encapsulates that to help you get more from mobile for good and not just for now.

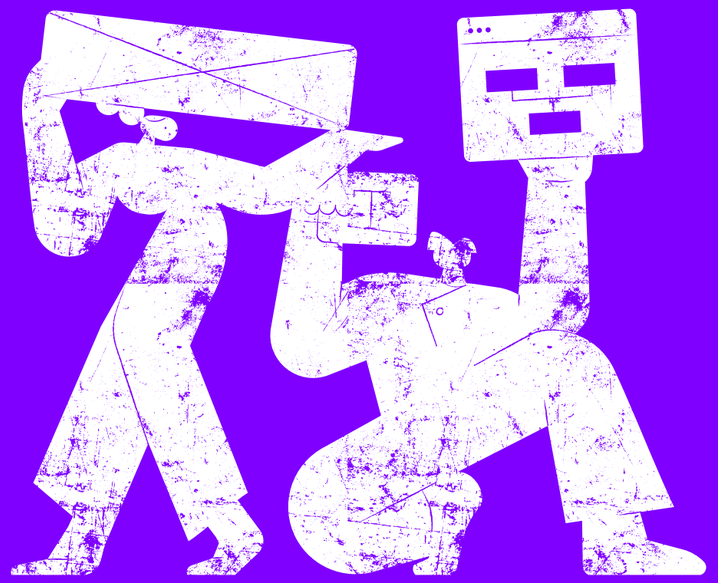
Building a mobile world that works better for everyone

The potential for mobile innovation to improve the lives of end-users and the fortunes of organisations is phenomenal; the challenge of finding the right technology for the job is unique. What will your users engage with? How much will your budget allow? How will your technology choices affect your team processes?

There are so many angles of consideration when it comes to pinpointing the perfect mobile technology for your specific set of requirements, but that's exactly the point: everyone has a specific set of requirements, so everyone has a different set of answers to the questions they'll ask along the way.
That's why this report exists: to help you ask yourself the questions that matter when it comes to finding the right mobile technology for you.

KIN+CARTA

A brief history of mobile app technology



Number of Apps at the launch of App Store 2008

500

Number of Apps on App Store 2021

2m

Let’s take a look at the backstory first. Things have come a long way since the App Store launched with around 500 apps back in 2008.

Today, there are around two million apps available on the Apple App Store and just shy of three million on the Google Play Store — all built using a wide range of tools, frameworks and programming languages.

Choosing from the countless options can be overwhelming. Before exploring which mobile app technologies are best suited to your requirements, it helps to consider the specific problems each technology aimed to solve when it was created.

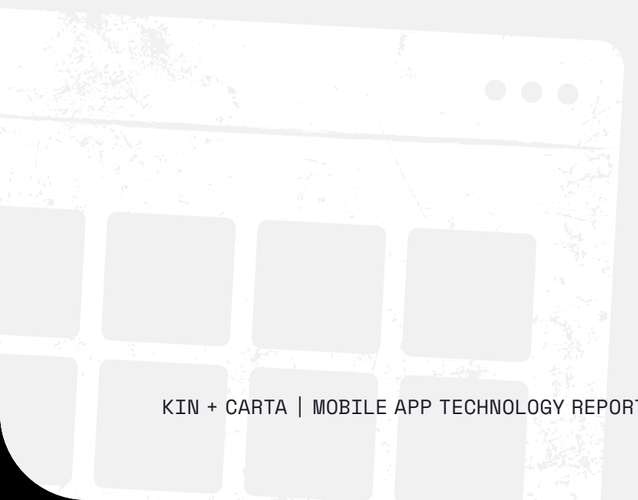
2007

In the beginning, there were web apps

HTML/CSS/JS

Apple

The launch of the iPhone saw Apple initially withhold a software development kit (SDK) from developers, with Steve Jobs declaring that so-called apps would simply be websites developed, distributed and updated using modern web standards (Web 2.0 and AJAX). Apple eventually conceded and released an SDK later that year.



2008

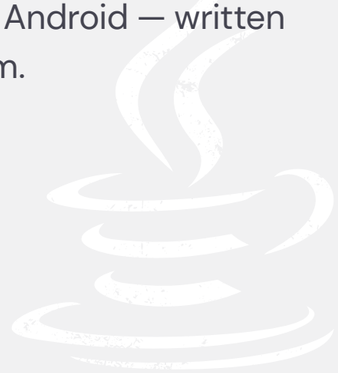
The birth of native mobile

Objective-C/Java/Swift/Kotlin

Apple, Google

Native apps came to mobile for the first time with Apple's SDK allowing third-party developers to build specifically for iOS, with integration capabilities unique to the host platform. Google followed suit in 2009 with the Android SDK.

The term "Native" is now used to describe app development that is conducted in platform-specific languages — Swift and Objective-C for iOS, and Kotlin and Java for Android — written separately for each platform.



2009

PhoneGap, Cordova and Ionic

HTML/CSS/JS

Adobe

Cross-platform technology came into play with PhoneGap, which later evolved into the open-source project, Cordova, via Adobe and the Apache Software Foundation. A separate Adobe commercial product, PhoneGap Build, enabled developers to get around the blocker of not having access to a Mac and build hybrid apps (neither web nor native per se) in the cloud. Adobe ended investment in PhoneGap and Cordova in 2020, but the open-source Cordova remains a foundation for versatile frameworks like Ionic today.

2009 continued...

Appcelerator Titanium

JavaScript

Appcelerator

Appcelerator's Titanium also went mobile in 2009, allowing developers to write apps using JavaScript only and opening doors to advanced integration with device OS and hardware. In 2021, it was announced that the whole Titanium framework would be handed over to the community, while Axway, which acquired Appcelerator in 2016, would end its support in March 2022.

2010–onwards

The rise of no-/low-code platforms and white label offerings

Throughout the 2010s, building an app was opened up to a greater number of small businesses and entrepreneurs through no-/low-code platforms and white label offerings. No-code platforms provide drag-and-drop app builder functionality without the need to write any code at all, while low-code platforms provide additional flexibility for those willing to write a small amount of code.

These services enable anyone to build an app and deploy to multiple platforms without the expense of hiring developers. Heavyweights Microsoft, Google and Amazon introduced no-/low-code developer platforms (PowerApps, AppSheet and Honeycode respectively) to make building mobile apps more accessible, while a number of startups (Bubble.io, Appy Pie, BuildFire) vie for a piece of the pie. However, the lower barrier to entry comes at the cost of customisation for some.

There are also many services offering white label apps. These vary in customisability, but usually allow the owner to add custom images, brand colours and text. They tend to be designed for common use cases such as shopping, media, booking and restaurant apps, although some are flexible enough to suit other needs. They come in at a very low cost compared to bespoke app development.

2013

Xamarin

C#

Microsoft

As a suite of frameworks — evolved from MonoTouch and Mono for Android — that extend the Microsoft .NET developer platform, Xamarin came into being in 2013.

It provides a common API for building native apps in iOS, Android and other operating systems. Written in C#, Xamarin opened the door of cross-platform mobile development to teams that already had .NET/C# experience.

2015

React Native

JavaScript

Facebook

Developed by Facebook to take advantage of its already hugely successful React web framework, the versatile JavaScript-based React Native quickly became the most popular cross-platform technology in the market.
It was created with its own layout syntax to allow developers to create apps for multiple platforms, from mobile and desktop to web and Smart TVs.

The term native is used because the UI is rendered using platform-specific views rather than as web content.

2015

The advent of Progressive Web Apps (PWAs)

PWAs also emerged in the latter half of the 2010s as a viable alternative to traditional approaches and promised to narrow the gap between web and native user experiences. Features like caching for offline use and support for push notifications meant such websites could behave very much like native apps. While they can be ‘installed’ to Android and iOS home screens, they remain unmistakably web-like in use. Advances in mobile browsers, however, along with UI libraries that mimic native components, have paved a promising path ahead for PWAs.

2017

Flutter

Dart

Google

Flutter was launched with the intention of rendering UI consistently at 120 frames per second (FPS) — twice as fast as the benchmark of 60 FPS — and it achieved this by using its own rendering engine and low-level graphics libraries. This allowed it to render views on multiple platforms that looked like their native equivalents, while actually bypassing the native layer.

2020

Kotlin Multiplatform Mobile

Kotlin

JetBrains

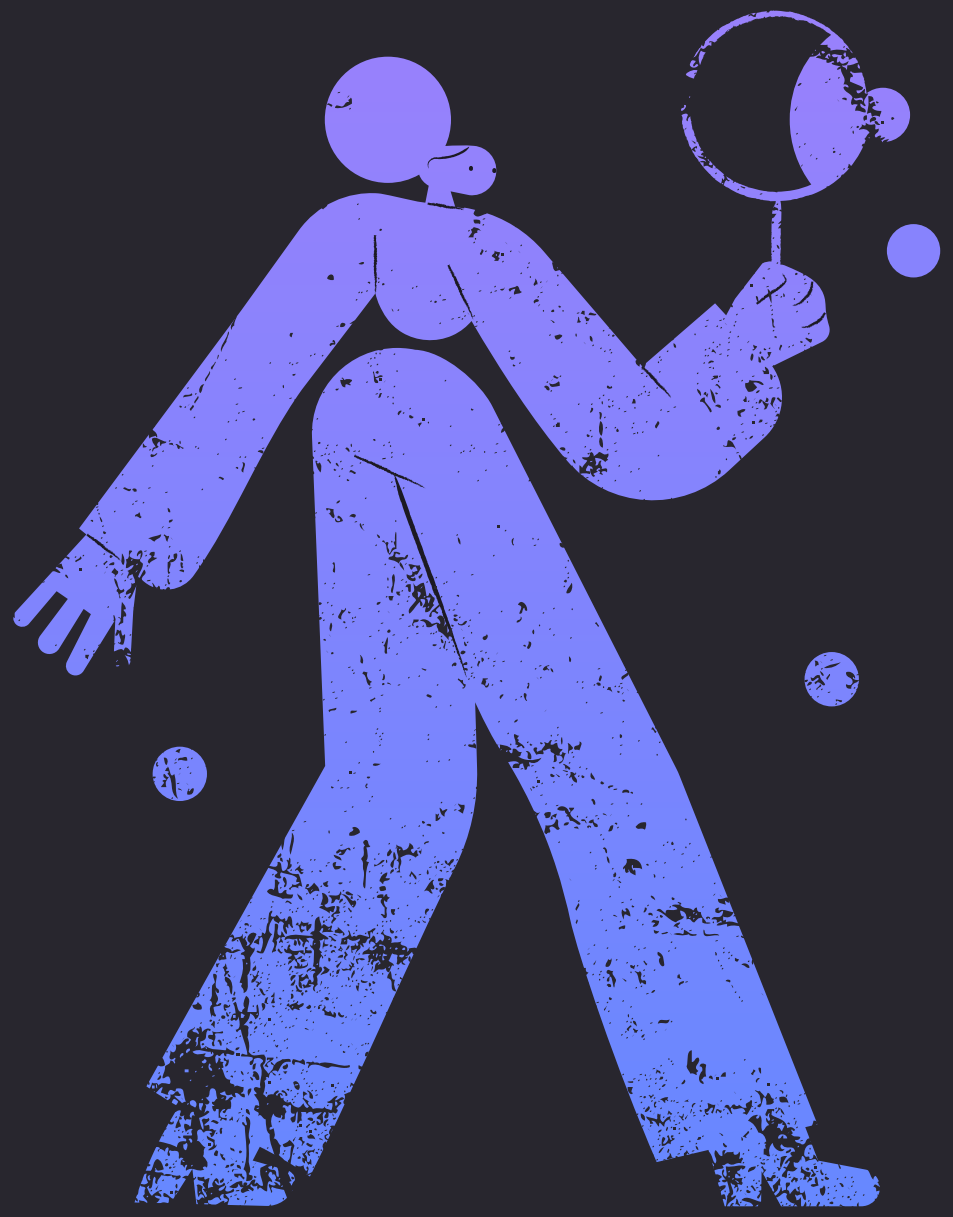
Kotlin Multiplatform Mobile (KMM) changed the way we think about sharing code between mobile platforms. Instead of providing a common syntax to render native views on each platform, all UI logic can be deferred to the native part of the codebase. From a UI perspective, these are fully native apps. KMM also allows module-by-module implementation for iOS, giving developers the capacity to choose where they use it and where a native iOS approach would be better suited.

Understanding the context behind developments in mobile provides a solid foundation on which to build a solution to your own mobile-specific business challenge. In the sections that follow, we'll take a look at some myths surrounding mobile technology and some of the questions you can ask to identify the right technology for you. →

Myths in mobile

The world of mobile is full of misconceptions that can lead you down the wrong path when it comes to choosing a technology.

Here, we've busted the most common myths to help you make the right choice to meet your objectives.



Myth:

A cross-platform approach will garner significant time and cost savings

Building once and deploying to multiple platforms might mean fewer developers, but it doesn't mean fewer specialists when it comes to managing the final product. Each platform has its nuances and therefore complex condition-based logic is required in the codebase (i.e. if iOS do A, if Android do B). This can quickly lead to difficulties in maintaining the code.

A multi-talented team of developers, designers, testers, researchers, delivery specialists and more will still be required to make sure the product works as it should everywhere, every time. Cutting back on such diversity in knowledge and insight can end up costing more than it saves in the long run, so it's crucial to weigh up the pros and cons of a cross-platform approach before you commit.

Myth:

A native approach always results in a better experience for the end-user

There is no reason why the choice of app technology should affect user experience. The best user experiences come not from technology but from developers investing the necessary time and energy to create something remarkable.

There are many facets to UX, including performance, interaction design, accessibility, thoughtful use of animations and messaging; the myth comes from developers choosing cross-platform technology in the hope of building an app in a hurry, which, by its very nature, is optimising against UX. With the right amount of time invested, any technology can be used to create a class-leading UX.

Myth:

A cross-platform approach for mobile means it'll be easy to share resources with the web teams

Mobile development demands ways of thinking that don't apply to web development (think GPS and camera integration and one-handed navigation on a device), so cross-training web developers to make more from existing resources isn't necessarily the best route.

Opting instead for mobile specialists on mobile teams ensures that the right mindsets are embedded into the development of your product from the outset, regardless of technology.

Myth:

Cross-platform means sharing code between web, backend and mobile will be easy

Even if mobile code is written in the same language as its associated website, it's not as effortlessly shareable as you might think (or, indeed, hope). Differences in the likes of syntax and UI components need to be taken into consideration; React Native, for instance, uses different UI components to HTML, even if the way they are declared resemble one another.

Consider the user experience and the different data required to optimise it on each platform when choosing the mobile solution for you.

How to choose the right technology

From build budget to audience appetite, there's a whole host of considerations that you'll need to factor into making a decision around mobile. Not every question here will be directly relevant to you, but, by applying the ones that are to your own specific circumstances, you'll soon be able to identify the possible technology solutions that will help you make your world work better for your mobile audience.

Is an app the best solution for your audience?

A cost-effective alternative to building an app is improving your website experience for mobile users. Your first port of call, therefore, should be to decide if you need an app at this moment in time. Taking advantage of Progressive Web App technology might be a suitable alternative for serving your customers on mobile. A PWA doesn't need to be a substitute for an app — you can always develop an app later if, for instance, you find that you need more control over the device or operating system.

It's becoming best practice to develop PWA functionality to make your website sufficient for mobile users in the here and now. Whether it's native or cross-platform, a dedicated mobile app is a major investment, so the requirements need to be clear and obvious if you are to undertake it and deliver a leading user experience (UX) as a result.

Here are a few considerations to help you decide

→ On the following screens our recommendations appear in this format

Do you have an opportunity to reuse existing code?

Rebuilding an existing app may provide opportunities to reuse existing code, but it's crucial to understand the reasons behind the rebuild before choosing a technology.

If it's a case of rewriting for design or performance improvements whilst largely retaining the UI and business logic, it's likely you can keep the same technology and rewrite the parts you need.

If it's a matter of significantly revamping design and UX or regaining control over an unmanageable codebase, you don't need to be restricted by past technology choices. You can take a new direction that will provide a better overall experience for your users and your developers, while reducing ongoing maintenance costs.

When to stick with the technology that's been used before

→ Rewriting significant parts of an app, but large parts of UI or business logic can remain as they are.

When to not be restricted by the choice of technology used before

→ Rebuilding due to code becoming unmaintainable.

→ Rebuilding due to significant design/UX changes.

Enterprise or Commercial

Are you building for enterprise purposes only?

An enterprise app typically gives users tools that improve their efficiency and productivity at work. There is no competition because your customers are your colleagues and they'll have no choice but to use your app, but that's not to say user experience won't matter.

Whether you want to provide more convenient access to in-house data, integration of other in-house tools or automation of repetitive tasks, compatibility with existing technologies within your business is a crucial part of securing a high-quality UX. This is where no-/low-code app builders from Microsoft, Google and Amazon have a major advantage; if the rest of your enterprise services sit within Microsoft Azure, for example, then PowerApps will be an efficient way to build a product that integrates seamlessly with the rest of your infrastructure.

If you are building an enterprise app that will integrate with existing in-house tools

→ Microsoft PowerApps

→ Google AppSheet

→ Amazon HoneyCode

Budget

How flexible is your budget and will it generate return on investment (ROI)?

The age-old challenge of getting the most out of your budget, of course, applies to mobile technology, too.

If the solution is to be part of your long-term growth plans, the initial investment in a custom cross-platform or native app may be higher, but the future benefits will be greater when you have complete control over it. If the initial investment simply can't be high, there are off-the-shelf solutions to suit your short-term objectives.

White label products like Yapp, app builder solutions like Bubble and the aforementioned PWA route can help you make timely savings, but these decisions need to be heavily influenced by your expectations for return on investment (ROI).

Will the user experience they provide be sufficient or will you need to spend more to gain more from customisation in the future? Try not to mistake your initial project budget for the app's lifetime budget and consider how much freedom you're likely to need in the long-term.

Low budget

→ App Builder

→ White Label

→ PWA

High budget

→ Native

→ React Native

→ Flutter

What is the scope and scale of your app?

When making a technology choice, you need to consider the shelf-life of your app, the role it'll play in your business, how quickly you need to get to market and how long you'll remain there. If your app will support a one-off event or short campaign, using an app builder or a cross-platform framework with template designs might be the most cost-effective way forward, but be mindful of the lack of customisation and OS integration this will provide.

If your app is effectively your business or at least a significant touchpoint for your customers, then native, platform-specific technologies will not only help you deliver a better experience now, but they'll also allow you to become more agile when you need to adapt your approach down the line.

The reality is that many use cases are somewhere in between. A digital news outlet, for instance, might use an app to serve only 10% of its traffic, making it a relatively small portion of its business, but predicted growth as a channel would necessitate the customisation and adaptability that only a native or cross-platform solution can provide.

A technology that's fit for purpose now might not be in five years when you need to serve 10 times as many customers, so be sure to make your choice with longevity front-of-mind.

Small scope and short shelf-life

→ App Builder

→ Ionic

→ White Label

Medium-to-large scope, long shelf-life or business critical

→ Native

→ React Native

Skills

Do you have access to the right skills and resources?

Choosing a new technology is one thing. Having the capacity to harness its potential is another.

Since JavaScript has been the most commonly used programming language for eight years running, you might expect it to be relatively easy to find JavaScript developers to build your app with it. Of course, they are greater in numbers than developers with experience in, say, Dart (Flutter), but they are also in higher demand and expect higher salaries, so access to such talent is not always attainable.

You might instead look at sharing resources with other disciplines if, for example, your web team contains web JavaScript developers; they'll be able to build a JavaScript-based React Native app using their existing skills. In a similar vein, backend Java/Kotlin developers can be redirected to work on an Android app. However, it's important to note that programming language only plays a small role in app development.

Developers with experience in the target operating systems and frameworks are of far greater value.

To share resources without their learning a new language

→ React Native (JavaScript)

Worth considering:

→ Java/Kotlin sharing Backend

→ Native Android

Do you want cutting edge or tried-and-tested tech?

There's an innate risk in picking the right technology. Not all solutions gain widespread adoption, while not all established ones stand the test of time in an evolving landscape. The major advantage of choosing a cutting edge technology is that developers will be eager to work with it. Many steer clear of legacy tech because they have an appetite for something new and shiny.

Objective-C, for instance, was once the language of choice for iOS apps, but there aren't many developers left who'd be willing to work on such a codebase. Apple's traditional UIKit approach might eventually give way to the new SwiftUI framework and, even in web development, many developers now prefer TypeScript over JavaScript.

Consider the long-term stability of your preferred technology and be sure to assess the inherent risks that come with picking one at either end of the spectrum.

Legacy; obsolete or at risk of becoming obsolete

→ Legacy Native (Java for Android; Objective-C for iOS)

Mature; tried and tested

→ Modern Native (Kotlin; Swift)

→ React Native

Infancy; cutting edge

→ Flutter

→ Kotlin Multiplatform Mobile

How will you use and contribute to third-party tools?

Cross-platform technologies introduce dependencies on both the cross-platform framework itself (React Native, Flutter etc) and the community projects that aren't as typical of totally native codebases.

> *Large codebase for a native iOS grocery shopping app: 12 dependencies*
> *Small codebase for a React Native magazine app: 120 dependencies*

If you opt for cross-platform, you need to be prepared to rely on more third parties, deal with potential bugs and delays that come with community packages and contribute your modifications back to the community.

If independence from third parties is important:

Best choice

→ Native

What to avoid

→ React Native

→ Flutter

How will you integrate with the device, OS and latest platform-specific features?

If rapid integration with the device and OS are to be a critical part of your product, native might be your best route forward. Cross-platform requires bespoke integrations in each case and, when new features are announced by Apple and Google, it can take weeks or months for independent or community-maintained open-source projects to add support.

Integration with the likes of biometrics, Bluetooth, calendar, camera, contacts list, accelerometer and more are widely expected to be seamless now – and native can help you achieve it to make your UX better and your customers' lives easier as a result.

Best choice for integrating with host platform

→ Native

Diminished returns as platform-specific integrations increase

→ React Native (delay with getting platform-specific features)

→ Flutter (interoperability challenges)

How important is continuous integration and advanced test tooling?

Continuous integration (CI) and continuous deployment (CD) pipelines typically start builds from scratch to ensure that no artifacts from previous builds affect the new build configuration. With cross-platform codebases, these build times are longer because of the increased number of dependencies. This is impacted further by dependencies used for automated tests as part of your CI/CD pipeline.

The deployment pipeline is also affected because of the tight coupling of Android and iOS. If you have a separate codebase for each platform, the pipelines are independent and can run in tandem to save time, so technology choice can have an impact on CI and CD.
The availability of test tooling varies by technology, too. Native development toolkits include standardised mechanisms for unit and UI automation testing, but some cross-platform approaches like React Native rely on community projects instead, which can be slow and unreliable. Conversely, Flutter includes mechanisms for unit testing, automation testing and visual snapshot testing right out of the box, so it can be a great choice if end-to-end tests are must-haves and not nice-to-haves.

If advanced testing and CI/CD are important to you:

Best choice

→ Native

→ Flutter

What to avoid

→ React Native

Should your app feel at one with the host platform or is consistency between platforms the key?

A crucial part of choosing the right technology for your app involves weighing up the importance of seamless UX for the user against the need for time- and cost-efficiencies for your team.

Each platform has its own UI design guidelines and users have come to expect apps to behave in ways that are consistent with their OS, so a native approach is typically the preferred option for those who don't want to compromise on the platform-specific experience for the end-user. Developing separately for each platform naturally makes it easier to stay within the guidelines and meet these user expectations, but it does require more work.

However, when consistency is the priority, a cross-platform approach can help streamline anything from design processes to customer service because they are the same across platforms; there is no need to tailor an approach to each one, so, if it's important to you, there are savings to be made here. Due to Flutter's bespoke rendering system, it is a great choice for consistency across platforms (you can even make an Android app that looks exactly like an iOS app and vice versa...).

Best choice to feel at one with the host platform

→ Native

Best choice for consistency across platforms

→ Flutter

How will you approach accessibility as a feature of your app?

Championing accessibility not only allows you to engage a wider audience, but it also gives you a competitive advantage in a world that is (rightly) increasingly intolerant of inaccessibility. If you build for users with vision, hearing and dexterity requirements first, you'll improve the overall experience for everybody. Native is the most versatile approach to help you achieve this, but there are inconsistencies between platforms to bear in mind here; iOS offers more accessibility options than Android and is the preferred choice for users with advanced accessibility needs.

When it comes to cross-platform technologies, Flutter and React Native both support accessibility features such as adjustable contrast, text size and screen readers, but the latter can require some additional native functionality when more complex layouts are introduced.

By comparison, the Cordova-based Ionic doesn't offer as much support for accessibility and what is offered is difficult to implement.

Best choice for Accessibility

→ Native

→ Flutter

Limited in terms of Accessibility

→ Ionic

→ Cordova

Do you want flexibility and compatibility with other technologies?

An important consideration of mobile technologies is their ability to interact with each other. This is highlighted when integrating third-party frameworks that are developed in a different technology. Such frameworks might be used to handle payments, analytics or video streaming, for example. If the framework is not compatible with your primary technology choice, then bridging code must be written and this is not always straightforward.

For example, due to Flutter's bespoke, non-native rendering system, it is difficult to integrate third-party frameworks with their own UI. Depending on how the third-party framework is developed, it might even be impossible.

Kotlin Multiplatform, however, makes module-by-module usage easy. You might choose to implement all of your authentication logic in Kotlin and share that between iOS and Android, while writing all other code in the platform-specific languages.
Taking a native approach generally allows you to integrate modules that are developed in other technologies as and when necessary. React Native also renders native views, so it's fairly simple to add natively coded views to the view hierarchy and to interact with native business logic, meaning you wouldn't be as restricted as you would be with Flutter.

If flexibility to mix technologies based on use case is important:

Most flexible choice for reuse between platforms

→ Kotlin Multiplatform Mobile

Least flexible

→ Flutter

What is your monetisation and distribution strategy?

If you intend to make money from your app by either selling it in Apple's and Google's app stores or enabling in-app purchases (or both), then only a PWA might hold you back.
The Play Store facilitates discovery but not purchases of PWAs, while the Apple App Store allows neither – and there's no indication it will do so in the future because of Apple's tight control over its own ecosystem. You can distribute a PWA in the Play Store, but since it'll otherwise be available via mobile browsers anyway, it doesn't make sense to pursue this as a tactic.

Either cross-platform or native approaches will allow you to monetise and distribute your app in these stores and there is no one option that trumps the other. If you plan to use an app builder platform, though, be sure to check that it supports in-app purchases first.

What to avoid if you wish to generate revenue from app sales or in-app purchases

→ PWA

The pursuit of app-iness: Real-world examples

Our connective mindset at Kin + Carta means the right minds can collaborate on finding the right mobile solutions for our clients without restrictive barriers getting in the way. It's what's characterised our expertise for over twenty years and, if we may say so, the proof is in our track record.

Here are a few examples of the solutions we've built in that time and the considerations we took to get there in each case:

A native solution for the UK's leading bank, NatWest

Leighton Kuchel
iOS Specialist
Kin + Carta

Mobile or web?

This product needed to allow users to authorise multimillion-pound transactions — authentication with biometrics was the answer. This is not something that's possible with web apps, so a dedicated mobile app was required.

→ *Mobile*

Budget

The target audience was business banking customers managing accounts and transactions worth millions of pounds, so investment in a bespoke solution for each platform allowed trust and confidence to be woven into the fabric of each user experience.

→ *Native*

Scope

The scope of the project involved managing accounts and authorising regular transactions, but, given NatWest's status as one of the leading providers of business bank accounts in the UK, this was required to work on a huge scale. Indeed, the ability to continue to scale was of paramount importance, too, so a native approach was perfectly suited.

→ *Native*

Skills

As the mobile equivalent of an existing web application, the app could have been built using existing skills within the client's organisation, should a cross-platform approach have been chosen.

→ *React Native*

Integration

Arguably the most important feature of the app was to use biometrics for authorising transactions; a native approach allowed the necessary control over the device integration.

→ *Native*

User experience

The trust involved in authorising huge transactions meant that the app's UX had to be slick, stress-free and unambiguous. The best way to achieve this was to build a bespoke solution that was fully integrated with each host platform.

→ *Native*

Conclusion

Although the client had existing web skills that could have been utilised had we chosen React Native, a Native approach came out on top. This made advanced device integration simpler and more robust; it provided a familiar, trustworthy user experience, all while giving us the long-term flexibility to scale.

→ *Native*

A React Native solution for a revered publication

Raynelle Alphonso
React Native Specialist
Kin + Carta

Mobile or web?
With a newly built app delivering a poor audio experience compared to the legacy app it replaced, users were reluctant to migrate. That's where Kin + Carta came in to rebuild the codebase to deliver a reliable audio experience and improved overall performance. We actually considered decommissioning the mobile app and instead fleshing out the client's website to offer a better experience on mobile. However, we concluded that a mobile app would provide a better audio streaming experience due to integration with operating system features like continued background playback.

→ *Mobile*

Code reuse
As the existing app was built with React Native, we were able to lift and shift much of the business logic and directly reference the existing code when recreating the user interface. We were also able to reuse utilities and GraphQL queries.

→ *React Native*

Skills
Colleagues in the client's web team and React Native mobile team were able to rotate relatively easily, which meant that continuing with React Native was the best route forward for everyone.

→ *React Native*

Integration
The fact that the audio experience was poor on the previous version of the app was largely to do with the bridging code between React Native and the native operating system audio capability. Taking a purely native approach would avoid this bridging and allow us to interact directly with the platform-specific APIs.

→ *Native*

Conclusion:
While a native approach would have offered benefits in OS integration, the ability to utilise existing skills and reuse existing code point to React Native as the best choice of technology.

→ *React Native*

A Flutter solution for global data science company, dunnhumby

Swav Kulinski
Lead Flutter Engineer
Kin + Carta

Mobile or web?
A mobile app was an obvious choice here, as the scope included high-performance calculations, machine learning, camera usage and uploading large files. We weren't sure how much processing would be required on the device and how much could be done in the cloud. The product also had to be enticing enough in appearance for users to want to use it.

→ *Mobile*

Budget
We had a small budget and the client prioritised rapid experimentation over long-term investment. The ideal solution would have been a PWA, but the obvious obstacles were the performance and integrations that would come with that. Flutter was still in its infancy, but it was exhibiting good support in important areas such as performance and testing. Flutter's hot- reloading — reflecting code changes without recompiling — allowed fast feedback loops between designers and developers.

→ *Flutter*

Cutting edge
When considering a novel technology like Flutter, we had to consider the risk that it hadn't been tried and tested like other technologies. With Flutter being advocated by Google and with good uptake in the community, we found that developers were very keen to work with the technology, setting the client up for easy access to skills should the app need iterating upon in future.

→ *Flutter*

Integration
We knew that integrating with the device and operating system would be the most demanding aspect of this project. We knew from the beginning that we would need access to the camera, but we didn't know what other requirements might come up later. We knew if we chose Flutter, we would need to build device and OS plugins ourselves.

→ *Native*

User experience
The ability to test and modify UX without much effort was crucial to meeting dunnhumby's high standards. Having a UI that was almost the same on each platform allowed us to streamline designs and design reviews, which was ideal due to the low budget.

→ *Flutter*

Conclusion:
This was a tough call, but Flutter came out on top based on the low budget and the need for rapid experimentation. We knew it would be a risk in terms of integration and, if we'd taken a native approach, we could have spent less time fiddling with third-party libraries. The appetite for developers to work on an experimental project in a new, modern technology gave us the confidence to choose Flutter and it proved to be a great choice.

→ *Flutter*

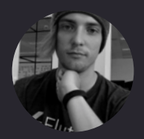
Thank You

Compiling this report has been a monumental team effort. It wouldn't have been possible without the endless insight of our subject matter experts. I'd like to thank everyone for the content they've written and for their involvement in the several workshops that fed into this report. Thank you!

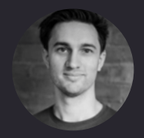
- Sam Dods, Head of Mobile Engineering

Arvind Ravi
iOS

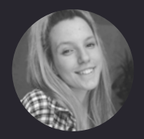
Dan Smith
iOS



Jacek Kulinski
Android & Flutter

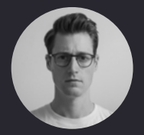


James Scott
Android



Kirsty Butler
Android

Leighton Kuchel
iOS & React Native



Jon Hocking
iOS

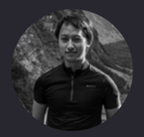
Neil Horton
iOS & Ionic

Raynelle Alphonso
iOS & React Native

Rob Morgan
Web & React Native

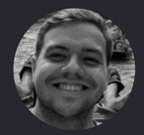
Robbie Fraser
iOS & React Native

Roberta Goodhead
Web & Flutter



Roger Tan
iOS & Flutter

Swav Kulinski
Android & Flutter

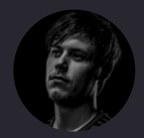


Voicu Klein
Android

Production team

Sam Dods
Head of Mobile & Editor

Boo Wallin
Creative



Andy Goodison
Creative

Emma Savory
Marketing

Creating a mobile foundation for the future

There is no one-size-fits-all approach to mobile technology – every use case is unique – but taking all these angles into consideration will put you on the path towards finding the right one for you.

The crux of the decision is in asking yourself where your product will (or should) be in three, four or five years' time.

Will you eventually need to integrate third-party frameworks even if you don't need them now? Will your app have served its purpose for a one-off event or will it need to be revived and updated every year? Will your audience crave more connected and engaging experiences as your organisation grows?

A timely solution might be perfectly suitable for some, but creating for tomorrow and turning initial outlays into long-term investments might be more prudent for others. If you want to remain agile and adaptive in an unpredictable landscape, it will be healthier to have the option to iterate than the necessity to lift and shift every time your audience demands something different...

About Kin + Carta

Kin + Carta is a global digital transformation consultancy committed to working alongside our clients to build a world that works better for everyone.

Our 1,600 stategists, engineers, and creatives around the world bring the connective power of technology, data, and experience to the world's most influential companies, helping them to accelerate their digital roadmap, rapidly innovate, modernise their systems, enable their teams, and optimise for continued growth.

In our offices across Europe, South America, and Singapore, and as a Certified B Corporation in the United States, our triple bottom line focus on people, the planet, and profit is at the core of everything we do. For more information, please visit www.kinandcarta.com

How can we make it happen for you? Get in touch