

Choosing the Right  
**Cross-Platform  
Framework**  
for Mobile Development



# Abstract

In this white paper, GlobalLogic's Mobile Practices Team rates the various features of cross-platform mobile frameworks to help readers choose the best technology stack for developing or expanding a mobile application. We also provide recommendations on when to apply which technologies for different types of mobile applications.

**Authors:**

Oleksandr Furdylo

Ajay Chaudhary

Denys Bratchuk



# Contents

4	<b>Introduction</b>
5	<b>Common Mobile Development Approaches</b>
6	Native Toolkits vs. Cross-Platform Frameworks
8	<b>Technology Stack Assessments</b>
9	Native App Toolkits Assessment
10	Cross-Platform Framework Assessments
10	<i>PhoneGap</i>
11	<i>Xamarin Native</i>
12	<i>React Native</i>
13	<i>Flutter</i>
14	<i>Progressive Web App (PWA)</i>
16	<b>Technology Stack Recommendation</b>
16	Mobile Application Only
17	Mobile and Desktop Applications
19	Mobile and Web Application



# Introduction

As the world becomes increasingly mobile, businesses must ensure that their digital solutions can continue to evolve in terms of scalability and extensibility. Users expect their mobile apps to continuously offer new features and services, as well as be available on any device, any place.

When building a new mobile solution, or evolving an existing one, businesses must carefully consider which technology stack and development approach to use. Using a cross-platform framework to develop apps across multiple platforms (versus developing individual app instances via native iOS and Android toolkits) has become an especially attractive option.

Each cross-platform framework option has its advantages and disadvantages, and choosing which technology to use depends on a huge variety of factors, including cost, industry trends, and future product roadmaps.

As a digital product engineering specialist, GlobalLogic has developed hundreds of mobile products for industry-leading businesses. We also have a dedicated Mobile Practices Team that explores the latest technologies around mobile development.

Based on this expertise, we have assessed the most popular cross-platform frameworks across a variety of factors to help readers select the best technology stack for their unique project. We have also provided recommended strategies for how to use these technologies to support mobile applications across web and desktop formats, as needed.





# Common Mobile Development Approaches

Below is a quick overview and comparison of the three most common approaches to mobile app development.

Native app toolkits support single platforms only. They provide faster performance, extremely handy navigation, experience optimized for the target platform, and full access to native capabilities.

Cross-platform frameworks enable the creation of mobile applications tailored for different platforms and contain cross-platform code written in JavaScript, C#, or C++, as well as platform-specific code wrappers for iOS, Android, and other platforms. Cross-platform frameworks save development efforts while coding applications.

Browser-based solutions provide a universal platform support. These are normally based on well-established web technologies and require centralized application hosting, but lack access to native features. Browser-based solutions save development efforts while coding, compiling, and deploying applications.

For the purposes of this paper we will focus on cross-platform frameworks, using native toolkits simply as a means of comparison.

	Code	Compile	Deploy
Native	many	many	many
Cross-Platform	once	many	many
Browser-Based	once	once	once

# Native Toolkits vs. Cross-Platform Frameworks

Below is a quick overview and comparison of the three most common approaches to mobile app development.

Native app toolkits support single platforms only. They provide faster performance, extremely handy navigation, experience optimized for the target platform, and full access to native capabilities.

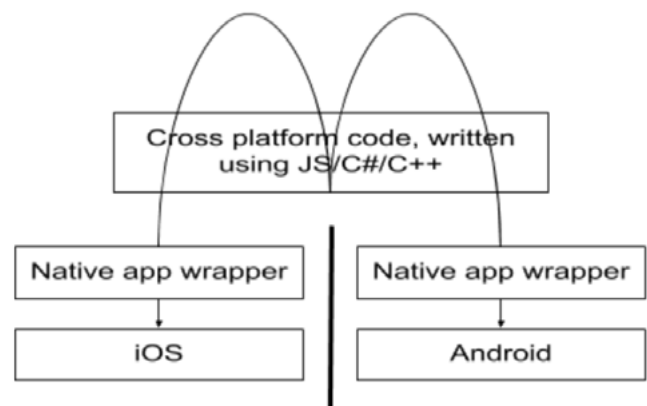
Cross-platform frameworks enable the creation of mobile applications tailored for different platforms and contain cross-platform code written in JavaScript, C#, or C++, as well as platform-specific code wrappers for iOS, Android, and other platforms. Cross-platform frameworks save development efforts while coding applications.

Browser-based solutions provide a universal platform support. These are normally based on well-established web technologies and require centralized application hosting, but lack access to native features. Browser-based solutions save development efforts while coding, compiling, and deploying applications.

For the purposes of this paper we will focus on cross-platform frameworks, using native toolkits simply as a means of comparison.

When cross-platform development frameworks appeared, they became very popular due to the fact that most of the code is shared, and therefore development efforts can be optimized. In addition, only one core team is needed to support all platforms, and applications can have a common set of documentation (including SRS and design specifications).

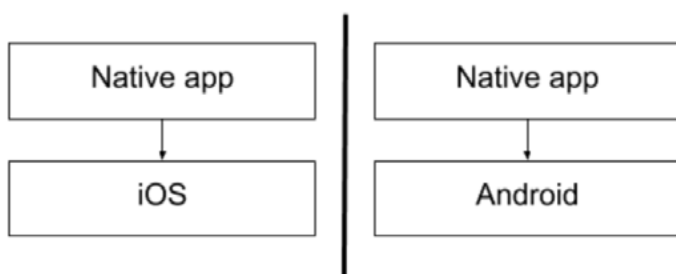
## Shared Code More Than 80%



However, cross-platform mobile applications have drawbacks as well:

1. Application performance will be lower compared to native applications due to an introduced intermediate level (a bridge between the common and native code).
2. Some device-specific functionality cannot be easily implemented using cross-platform instruments (e.g., push notifications, behaviour in sleep mode, multithreading, data-storages etc.). These functions can be supported by cross-platform frameworks to some extent, but in this case the system will include an additional abstraction level, which makes the system more complicated and harder to maintain.

## Shared Code 0%



## Native Toolkits vs. Cross-Platform Frameworks (cont.)

3. Plugins and libraries used during cross-platform development are normally less stable than native tools and may contain bugs and issues that are hard or impossible to fix on the developer's side.
4. User experience in different platforms varies. For instance:
  - Typical iOS applications include a TabBar at the bottom to switch screens (Home, Profile, Search etc.), whereas the navigation bar is normally at the top and it includes Title, Back, and Action items.
  - Even though Android also has a navigation bar, it is used differently and normally doesn't contain buttons.
  - There is a SegmentedActivity element supported by Android, which looks similar to iOS TabBar but has totally different mechanics (e.g., iOS TabBar doesn't enable you to swipe screens, whereas it is a common practice in Android).
5. Cross-platform applications require large regression testing cycles, as the changes in an application can affect multiple platforms.

# Technology Stack Assessments

In this section, we will assess each mobile toolkit across a variety of factors, with each factor being ranked on a scale from 1-5 ("1" being the lowest value and "5" being the highest).

Below is an overall comparison of each of these technologies. Continue reading for additional details and recommendations about each unique technology stack.

	Native (Standard)	Phonegap	Xamarin Native	React Native	Flutter	PWA
Performance	5	3	3	4	4	3
Popularity	5	2	3	4	5	5
User Experience	5	3	4	4	4	4
Cost of Development	3	5	5	5	5	5
Bug Fixing	5	3	3	3	4	3
New Features Development	2	5	5	5	5	5
Availability of Resources	4	3	3	2	3	4
Code Reusability	1	4	3	3	4	5
Development Feasibility	5	3	4	4	4	3
Synergy with Backoffice	4	5	4	5	5	5
Additional Costs	5	5	4	5	5	5
Security	5	3	4	4	4	3
Application Bundle Size	5	4	4	4	4	4
Support for Platform Updates	5	2	4	3	3	2
Risks	5	2	4	4	2	2
Total Score	64	52	57	59	61	58



# Native App Toolkits


## Assessment

Although they are costly to develop, native apps are undoubtedly more efficient and maintainable from a technical standpoint. They also provide better performance and user experience for animation-heavy applications. To provide a base level for comparison purposes, below is our assessment of native app toolkits.

However, cross-platform mobile applications have drawbacks as well:

1. Application performance will be lower compared to native applications due to an introduced intermediate level (a bridge between the common and native code).

2. Some device-specific functionality cannot be easily implemented using cross-platform instruments (e.g., push notifications, behaviour in sleep mode, multithreading, data-storages etc.). These functions can be supported by cross-platform frameworks to some extent, but in this case the system will include an additional abstraction level, which makes the system more complicated and harder to maintain.

		Native App Toolkit Assessment (iOS, Android)
Performance	5	Native applications support all types of available optimization.
Popularity	5	Native development is, has always been, and always will stay in demand
User Experience	5	Native applications enforce the most up-to-date usability standards specific for each platform, which provide the best user experience.
Cost of Development	3	Cost is higher due to duplicated efforts to implement the same functionality for iOS and Android.
Bug Fixing	5	In terms of testing and fixing platform-specific features, native applications are preferable.
New Features Development	2	New features need to be designed and developed separately for every supported platform.
Availability of Resources	4	Native mobile developers are generally easier to onboard. 
Code Reusability	1	Source code of mobile application on one platform can't be reused on another platform.
Development Feasibility	5	All currently available features can be implemented in native applications
Synergy with Backoffice	4	While native apps can be integrated with MS-based backend, synergy will remain the same if other backoffice platforms are used (Google, Amazon etc.).
Additional Costs	5	No major additional costs involved
Security	5	Provides maximum application security
Application Bundle Size	5	Optimal application size
Support for Platform Updates	5	Native applications get updates in sync with the latest Android and iOS features.
Risks	5	No known risks, high predictability
Total Score	64	

# Cross-Platform Framework Assessments

## PhoneGap

Applications based on PhoneGap use WebView and have a simple implementation: create a small, native application that displays the built-in web browser and single-page HTML. There are no native controls or direct access to the API, as all the interface elements inside the web page are simply stylized as native ones. The application has access to the system functionality using special plug-ins that add JavaScript methods to the inside of the web browser and associate them with native implementation on each platform.

For application development on PhoneGap, engineers need experience in HTML, JavaScript, CSS, and also native instruments such as Objective C and Java. Additionally, engineers must have good engineering knowledge about integrating native and cross-platform parts.

Our opinion is that using PhoneGap or similar technology would not provide any significant benefits for mobile application development.

	Phonegap Assessment	
Performance	3	Performance is worse due to two additional layers: browser and wrapper.
Popularity	2	Phonegap / Apache Cordova and similar approaches are losing their popularity.
User Experience	3	If a mobile-first approach is being used, web pages may look pretty similar to mobile applications, but this will require additional efforts. JavaScript pages are not mobile-friendly by default.
Cost of Development	5	Cost is significantly lower than native app development.
Bug Fixing	3	Bug fixing is more complex due to complicated application structure and additional "layers" between the original code and final front-end.
New Features Development	5	New features can be created for all platforms simultaneously.
Availability of Resources	3	JavaScript resources are easy to find, but platform-specific resources will be needed as well.
Code Reusability	4	Source code can be reusable for different platforms and even for web pages.
Development Feasibility	3	All typical features, including drag-and-drop and graphical elements, can be implemented within the Phonegap application, but the final user experience may vary due to the browser specifics.
Synergy with Backoffice	5	Can be integrated with any backend
Additional Costs	5	No major additional costs involved
Security	3	It requires additional security considerations when developing an app that uses a non-native framework. Also, since the browser layer is additionally introduced, the application may be more open to potential vulnerabilities
Application Bundle Size	4	The size of the application will probably be higher due to additional wrappers needed.
Support for Platform Updates	2	Hybrid applications are held back by the lack of a particular plugin that supports a new feature.
Risks	2	Higher risks due to lower popularity of the approach and less support
<b>Total Score</b>	<b>52</b>	

## Xamarin Native

Xamarin Native is a cross-platform mobile app development framework based on the .NET framework. It uses C# to create applications for mobile platforms, and it is natively compiled. As a result it enables us to build high-performing applications with close-to-native design. The framework uses native libraries for cross-platform development and can access native APIs.

Xamarin Native is a popular choice for cross-platform application development. It is often considered while building mid-size business mobile applications for different platforms, especially those integrated with MS-based backend.

		Xamarin Native Assessment
Performance	3	Less flexible in terms of optimization as compared to native apps
Popularity	3	While Xamarin Native still has a developed community and visible roadmap, the popularity of Xamarin Native is decreasing in comparison to React Native (and definitely less popular native mobile approaches).
User Experience	4	User experience will be close to native app UX due to reusable UI components.
Cost of Development	5	Cost is significantly lower than native app development.
Bug Fixing	3	It's harder to fix bugs in a cross-platform application due to its more complicated structure, longer regression testing cycle, and potentially more complex nature of the issues.
New Features Development	5	New features can be designed and developed for all supported platforms at the same time.
Availability of Resources	3	We expect that cross-platform resources needed to support Xamarin Native are harder to onboard, and platform-specific resources will still be needed.
Code Reusability	3	Source code can be reusable for different platforms, but it can't be used for web pages.
Development Feasibility	4	While Xamarin Native can potentially be used to create interfaces as complex as needed, in reality the feasibility of the most complex features needs to be checked in advance.
Synergy with Backoffice	4	Since Xamarin Native is sponsored and supported by Microsoft, it must have a good synergy with MS-based backend
Additional Costs	4	Although Xamarin is a free open source platform for individual developers, the framework may cost a lot for enterprises to purchase a license for Visual Studio.
Security	4	It requires additional security considerations while developing an app using a non-native framework.
Application Bundle Size	4	Depending on their type and complexity, hybrid apps are typically larger than native ones (e.g., native app might be half the size of Xamarin app).
Support for Platform Updates	4	Non-native platforms take time to support new features for the latest platforms. Xamarin is better on this aspect than React Native, where it provides support for a new iOS SDK within 24 hour. However, Android support isn't available within 24 hours of a release due to the sloppy nature of Android rollouts.
Risks	4	No major risks related to the platform
<b>Total Score</b>	<b>57</b>	

## React Native

Similarly to PhoneGap, React Native (sponsored by Facebook) lets developers build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components. As opposed to PhoneGap, React Native enables you to render native components, not just a web view.

With React Native, you don't build a "mobile web app," an "HTML5 app," or a "hybrid app." It enables you to create a real mobile app that is indistinguishable from an app built using Objective-C or Java. React Native uses the same fundamental UI building blocks as regular iOS and Android apps; the developer just puts those building blocks

together using JavaScript and React. There is still a possibility to use native code.

For application development on React Native, engineers need experience in JavaScript and also native instruments such as Objective C or Java.

All in all, React Native can be considered as an alternative to Xamarin Native in the area of cross-platform development, especially for new applications created from scratch, as its popularity is growing (as compared to Xamarin Native), the cost of development of a new application will be lower than for native applications, and the performance will be nearly native.

	React Native Assessment	
Performance	4	Native tools can be used which makes the performance better than on Xamarin Native, closer to native app performance.
Popularity	4	Popularity of React Native grows as compared to Xamarin Native
User Experience	4	User experience will be close to native applications' one due to reusable UI components.
Cost of Development	5	Cost is significantly lower than Native app development.
Bug Fixing	3	Bug fixing time is close to Xamarin Native.
New Features Development	5	New features can be designed and developed for all supported platforms at the same time.
Availability of Resources	2	JavaScript / React Native resources are harder to find, and platform-specific resources will be needed as well.
Code Reusability	3	Source code can be reusable for different platforms, but can't be used for web-pages.
Development Feasibility	4	While potentially React Native can be used to create as complex interfaces as needed, in reality the feasibility of the most complex features needs to be checked in advance.
Synergy with Backoffice	5	Can be integrated with any backend.
Additional Costs	5	No major additional costs involved.
Security	4	It requires additional security considerations while developing an app using a non-native framework.
Application Bundle Size	4	Depending on their type and complexity, Hybrid apps (Xamarin, RN) are typically larger than native ones (Native app might be half the size of a Xamarin app).
Support for Platform Updates	3	Non native platforms are taking time to support new features of the latest platforms. Xamarin is better on this aspect then RN, where it provides support for new iOS SDK within 24 hour, however android support isn't available in 24 hours of release due to sloppy nature of android rollouts.
Risks	4	No major risks related to the platform
Total Score	59	



## Flutter

In terms of the cross-platform space the big two approaches are still Xamarin and React Native (backed by Facebook). However, good traction has recently been shown by Flutter (backed by Google) and its adoption is slowly increasing, though the number of applications using Flutter are significantly less compared to Xamarin or React Native.

Flutter and Dart are two of the fastest-growing technologies worldwide, and this stack is easy to learn for developers with native (iOS/Android) or web background. Also, Google is heavily investing in this technology.

According to the trend, we can predict that in the near future, the market will have a lot of competitive Flutter/Dart developers. The main strengths of Flutter include:

- Fast onboarding process for new developers with Web or Android background
- Performance. Flutter application is native and there are no additional layers between Flutter and hardware
- Support from the Google community. Google has developed Flutter itself, as well as a lot of different UI widgets. This allows the build of complex UI in short period of time

		Flutter Assessment
Performance	4	Great performance as Dart code is compiled into platform native binaries and widget painting is done by skia rendering engine. So there are no "bridges" to native platforms when it comes to rendering.
Popularity	5	Flutter is listed as the #2 fastest growing open source projects and Dart the #1 fastest growing language worldwide, October 2019. <a href="https://octoverse.github.com">https://octoverse.github.com</a>
User Experience	4	User experience will be close to native apps when it comes to material and cupertino widgets (but for some widgets not identical, as flutter implements those widgets). For custom UIs users will not see the difference with native.
Cost of Development	5	Cost is significantly lower than Native app development.
Bug Fixing	4	Bug fixing is not complex because of one codebase for all platforms and because of the fact that flutter is a relatively simple framework without complex component lifecycles. Although bug fixing native platform code in case of presence of platform channels may require additional effort.
New Features Development	5	New features can be designed and developed for all supported platforms at the same time.
Availability of Resources	3	No problems with Flutter/Android developers. Harder to find Flutter devs with iOS background.
Code Reusability	4	Source code can be easily reusable across mobile/web/desktop. But additional efforts required in case of use of platform native APIs.
Development Feasibility	4	Flutter was designed to build complex layouts and different varieties of animations.
Synergy with Backoffice	5	Can be integrated with any backend.
Additional Costs	5	No major additional costs involved.
Security	4	It requires additional security considerations while developing an app using a non-native framework.
Application Bundle Size	4	Depending on their type and complexity, Flutter apps are typically larger than native ones.
Support for Platform Updates	3	Non native platforms are taking time to support new features of the latest platforms.
Risks	2	Still not widely supported and growing quickly, risks are quite high.
<b>Total Score</b>	<b>61</b>	

## Progressive Web App (PWA)

PWA (Progressive Web App) is an approach which has grown more popular over the last few years. It enables the extension of the functionality of web sites and makes them closer to “applications” or even “mobile applications.” This includes the ability to work in offline mode, access the camera on mobile devices, etc.

The main driver of PWA is Google, therefore Android and Chrome support most of the features. Apple’s coverage of PWA features is also growing.

PWA is normally used to extend the functionality of existing websites. This can be done gradually, by means of adding PWA-related features one by

one. PWA works best with mobile-friendly sites with responsive design. If a site is already mobile friendly, it only takes a day or two to convert it into a basic PWA-application.

One of the main features of PWA applications is that it can be added to the main screen on mobile phones, and moving forward, it can be used as if it were a mobile application.

The main difference between PWA and native or hybrid mobile apps is that you don’t need to download PWA from the marketplace, as it can be downloaded directly from the site.

		Progressive Web Apps Assessment
Performance	3	Performance is worse due to the additional browser layer.
Popularity	5	Popularity of PWA is growing.
User Experience	4	PWA enables developers to create pages which look similar to mobile applications, yet this requires following Mobile-first approach.
Cost of Development	5	Cost is significantly lower than Native app development.
Bug Fixing	3	Bug fixing is more complex due to complicated application structure.
New Features Development	5	New features can be created simultaneously for Mobile and Web applications.
Availability of Resources	4	JavaScript resources are easy to find, and you can use any JS framework (React, Angular, Vue etc.)
Code Reusability	5	Source code can be reusable for different platforms and web-applications.
Development Feasibility	3	All typical features, including drag-n-drop and graphical elements can be implemented within PWA, but final user experience may vary due to the browser specifics.
Synergy with Backoffice	5	Can be integrated with any backend.
Additional Costs	5	No major additional costs involved.
Security	3	It requires additional security considerations while developing an app using a non-native framework. Also, since browser layer is additionally introduced, the application may be more open to potential vulnerabilities
Application Bundle Size	4	The size of the application will depend on the size of the PWA-powered web-pages.
Support for Platform Updates	2	PWA applications are held back by the lack of a particular plugin that supports a new feature.
Risks	2	Still not widely supported and growing fastly, risks are quite high.
<b>Total Score</b>	<b>58</b>	

## PWA (cont.)

Progressive Web Apps cannot be used for transferring native mobile iOS or Android apps though. If there is a native mobile application, then a JavaScript-based PWA application will need to be created from scratch.

A PWA approach can be used either purely as is or by means of wrappers such as the Electron JS framework.

Progressive Web Applications share the following principles: progressive, responsive, faster after initial loading, connectivity independent, application-like, fresh, safe, discoverable and re-engageable. Among the others, PWA applications normally utilize one or several of the following technologies: Manifest, Service Workers, Web Storage, WebAssembly and Databases.

A PWA approach is mostly useful when creating a strategy and architecture of new massive software products and ecosystems, and combining mobile and web experience. UI/UX design of all existing web-based applications will need to be adjusted to be mobile-optimized, and all newly created products will have to follow a “Mobile-First” approach. It makes no major sense to replace an existing native or cross-platform mobile application with Progressive Web App if there is no plan to build a web-version of the platform.



# Technology Stack Recommendations

The choice of technology for building a platform heavily depends on the pros and cons of each technology, as well as the roadmap of the overall platform development. Below are GlobalLogic's recommendations on how to leverage the previously discussed cross-platform frameworks based on how you plan to build or expand your product.

## Mobile Application Only

### Recommendation: Xamarin Native, React Native, Flutter

Taking into account the scope of a typical mobile application, the technical specifics of different mobile development approaches, their popularity, and our experience in using these approaches, we came to the conclusion that Xamarin Native, React Native, and Flutter are the best technology choices for a cross-platform mobile application development.

If you want to build a mobile application from scratch and support both iOS and Android platforms, we

recommend using React Native or Flutter since they can support multiple platforms, deliver comparatively high performance, and are growing in popularity.

### Exceptions

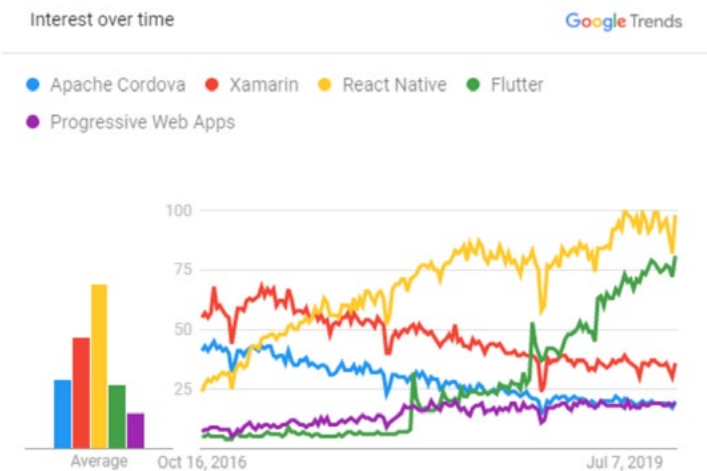
1. We recommend creating a native application under the following circumstances:
  - If you want to build a mobile application from scratch and support either the iOS or Android platform (but not both)
  - If your product has any features that are heavily mobile-specific and require complex custom implementation
  - If your product has strict performance requirements



## Mobile Application Only (cont.)

### Exceptions (cont.)

2. If you already have a mobile application, we caution switching to a new platform due to the cost of rewriting an application and the potentially negative impact on existing users. There should be a valid business reason for switching to another platform, such as major application-related bottlenecks, or significant performance or security issues.



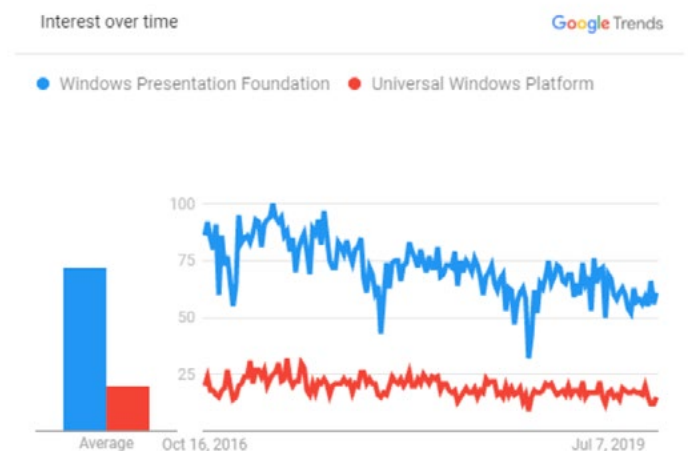
Interest over time charts based on Google Trends results (<https://trends.google.com/>).

## Mobile and Desktop Application

### Recommendation: Xamarin and WPF

If your goal is to create a desktop application in addition to the mobile app, we recommend using a .NET technology stack and, in particular, a WPF (Windows Presentation Foundation) subsystem. For the Mobile + Desktop combination, we recommend utilizing Xamarin.Native (Xamarin.Android + Xamarin.iOS) for mobile, and Xamarin.Native (Xamarin.Mac) and WPF for desktop.

The WPF application will be able to partly utilize the code of the Xamarin.Native mobile application, as these are both written in C#, and they can become a part of a larger Microsoft-based ecosystem. WPF is still popular and more stable than UWP (Universal Windows Platform), which was introduced in Windows 10.



Interest over time charts based on Google Trends results (<https://trends.google.com/>).

## Tools Matrix

The tools used to develop and test the next generation of mobile and desktop applications will certainly evolve over time in collaboration with the implementation team. However, we have listed a potential set of tools here as a starting point for

discussion, with the understanding that many of these tools will be common. This enables the use of similar — or even the same — development resources to build and maintain desktop and mobile applications

Architectural Component	Xamarin Native	WPF
Build Tool	msbuild	msbuild
IDE	Visual Studio 2019 Professional + Visual Studio for Mac (XCode + Android Studio optional)	Visual Studio 2019 Professional
Monitoring Tool	AppCenter	Microsoft Store
Static & Dynamic Code Analysers	StyleCop/ReSharper	StyleCop/ReSharper
Document Repository	any	Sharepoint
Wiki	any	any
Test Coverage	Visual Studio (Enterprise)/ReSharper's dotCover	Visual Studio (Enterprise)/ReSharper's dotCover
Source Control	Git	TFS / Git
CI	AppCenter/Bamboo	AzDo
ALM	Azure DevOps Server	AzDo

# Mobile and Web Application

## Recommendation: PWA

If your goal is to build a web application in addition to mobile apps, we recommend considering the Progressive Web App (PWA) approach. Unlike traditional applications, Progressive Web Apps are a hybrid of regular web pages (or websites) and a mobile application. This new application model attempts to combine the features offered by most modern browsers with the benefits of a mobile experience. More importantly, the PWA approach can be used to build desktop applications, as well.

Mobile users can access the PWA through their mobile browser by URL. On the first use, the PWA will suggest adding a shortcut on the home screen, so users can access it as a regular mobile app

later on. Desktop users can open the PWA in their browser and use it as a website. Additionally, users can install the PWA onto their desktop through the Chrome browser, and the application will open in a separate window and mimic a desktop app.

If you already have a legacy mobile application (e.g., Xamarin Native app), GlobalLogic recommends that you start developing a new PWA as though it were a regular web application, with a mobile-first UI/UX design approach. The estimated effort to add PWA-specific features is approximately an additional 20-30% on top of what is needed to create a usual web application with the same functionality.

	PWA	Native App
Platform Compatibility	Can be used on desktop, tablets, and smartphones; source code can be reused	Only developed for a specific platform
Connectivity Independence	Can work both offline and on low-quality networks	Can work offline
App-Like Interface	Mimics navigation and interactions of native apps	Navigation and interaction of native apps
Push Notifications	Sends push notifications	Sends push notifications
Updates	Self-updated, no need to re-install or download new versions	Updates rolled out via app marketplace and should undergo its policy
Safety	Due to updates strategy, the app is always using the latest stable and secure version	Safety is controlled by the marketplace
Discoverability and Easy Installation	Can be accessed via a link or URL; no need to install via app marketplace	Needs to be installed via app marketplace

## Pros and Cons of Using PWA

As per Comscore's report, the reach of the mobile web is 2.5 times more than that of apps when considering the top 1,000 sites and apps. Each step to download an app reduces 20% of users. PWA reduces the steps between the discovery of an app and getting it on the home screen, which thereby eliminates the friction of getting an app installed.

For example, AliExpress is using PWA with great results:

- 104% for new users across all browsers
- 82% increase in the iOS conversion rate
- 2x more pages visited per session per user across all browsers
- 74% increase in time spent per session across all browsers

Pros	Cons
<ul style="list-style-type: none"><li>• Can be used for web, mobile, and desktop applications</li><li>• If an application is mostly offline, PWA supports offline mode in all types of applications</li><li>• Provides easier delivery to devices, with no need to submit to the marketplace</li><li>• Easier testing of features common for all platforms</li></ul>	<ul style="list-style-type: none"><li>• Lacks access to native mobile features</li><li>• Comparatively new technology; not fully supported by all browsers</li><li>• Increased battery use compared to native apps</li></ul>



# About GlobalLogic

GlobalLogic is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating strategic design, complex engineering, and vertical industry expertise—we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses.

Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the communications, automotive, healthcare, technology, media and entertainment, manufacturing, and semiconductor industries.

[www.globallogic.com](http://www.globallogic.com)