2019 5th International Conference for Convergence in Technology (I2CT)
Pune, India. Mar 29-31, 2019

Analysis of Cross-Platform Mobile App Development Tools

Kewal Shah
B. Tech 3rd year, Computer Engineering
MPSTME, NMIMS University
Mumbai, India
kewalshah077@gmail.com

Harsh Sinha
B. Tech 3rd year, Computer Engineering
MPSTME, NMIMS University
Mumbai, India
h.sinha.588@gmail.com

Prof. Payal Mishra
Computer Engineering Department
MPSTME, NMIMS University
payal.mishra@nmims.edu

***Abstract*—With the emergence of different cross-platform alternatives, there is a need to explore the various approaches the developer must take. We perform this study in the hopes of shedding more light on their differences and highlighting the critical aspects which make them unique. The examination of these WORA (Write Once, Run Anywhere) mobile App tools is done on the basis of different approaches, i.e., Native apps, Web apps, Hybrid apps, Interpreted apps and Widget-based apps. The study performed shall enable us to illustrate the results on the basis of these categories.**

Keywords—cross platform tools; mobile development; mobile frameworks; evaluation; Angular; React Native; Cordova; Flutter; native apps.

I. Introduction

There has been no denying the rapid growth in the mobile industry as well as the services being tailored for it. By 2020 the expected amount of smartphone users may rise to 2.87 billion [1]. With the rising number of mobile users, comes an opportunity for tactful marketing and new business strategies owing to its increasingly pervasive nature in society. The early adopters are amongst the IT industry, retailers, enterprises, and even e-commerce startups. With the hectic and fast-paced nature of big corporates and conglomerates, it has become crucial to manage time and workload efficiently. Computer programs specially designed to run on a mobile device, commonly referred to as mobile applications or “apps” play a major role in catering to the needs of users. Before a developer starts making an app it is important to decide what mobile operating systems is he or she willing to target. In this paper we use the terms mobile operating system and platform interchangeably.

According to StatCounter [2], as of September 2018 the market share stands at 77% for the Android operating system developed by Google, 21% for iOS developed by Apple Inc. and 2% for others. A developer requires completely different skill sets for developing apps that are specific to these platforms, commonly referred to as native apps. Therefore an extensive knowledge of Java or Kotlin will be required for Android, while Objective-C or Swift will be required for iOS. For software developers that wish to deploy their apps on both platforms the native app development process would require a significant amount of investment with respect to time, money and effort. While native app development has its own advantages, it lacks the ability to run on multiple platforms without re-writing the code to run on different runtime environments, which is why many professionals are turning to cross-platform application development. Cross-platform mobile development tools can compile the application source code for multiple sources. The approaches to creating mobile apps using cross-platform alternatives are numerous and somewhat confusing as the right tools need to be chosen judiciously. There are multiple factors to be considered, i.e. choice of SDK, user experience, stability of framework, ease of updating, cost of development and time to market an app.

The structure of this paper is easy to follow. In Section II we have conducted a literature survey based on the related works as a form of prerequisite for our extensive study. Section III describes each cross-platform tool, highlighting our findings of their features and structure. Section IV forms a quantitative analysis on the types of apps based on certain parameters and proceeds to draw inferences from it. Finally, in Section V, we reach to a conclusion based on our studies and findings from our review.

II. Literature Survey

While our studies sometimes delve into technological know-how from an implementation standpoint, we do not directly discuss the techniques to build an app. The analysis of mentioned frameworks and tools is inspired by the research conducted by N. Huy and D. VanThanh [3] on the criteria for evaluation of apps. The work done by M. Palmieri, I. Singh and A. Cicchetti [4] goes through the architecture and the working of each technology, and then proceeds to draw conclusions based on certain parameters. This paper seeks to perform side-by-side comparison of some of these tools.

Our approach on cross-platform mobile development evaluation is different as it takes an example from each of the cross-platform approaches – web, hybrid, interpreted, and widget-based apps and compares them to a native development environment. Many papers offer an in-depth analysis of the cross-platform tools like Wenhao Wu’s [2] thesis on The Flutter framework however there are very few that compare cross-platform tools like web apps and hybrid apps with relatively newer ones like React Native and Flutter.

III. Comparing Cross-Platform Applications with Native Applications

The first step for app developers is to decide which platform to tailor their work for. Rarely, are there any cases in which a developer decides to develop apps for a single platform, like iOS. In which case, native app development is the way to go. But for all other cases, one must go for a cross-platform development alternative if time and effort are of the essence.

The option for choosing the method for developing a mobile app lies solely with the developer. Sometimes the approach is straight-forward, but at other times, the solution for the approach is complex. Here, we mention some of the finer points which enable developers to set forth on the right path.

A. Advantages:

i. Cross-platform tools are also referred to as WORA tools (Write Once, Run Anywhere) which is the primary advantage of such a method. The app is implemented using a single code base but can be deployed to multiple platforms, i.e., it is not restricted to a single platform.
ii. Cross-platform tools usually use well-known programming languages and syntaxes, which becomes an easy and quick means for development. Additionally, there is the added benefit for not needing to rewrite and customize code for separate platforms.
iii. Major tools have a large community-driven platform which is open-source in nature. New features and modules are continually being added, updated and revised as the need arises. It is not tied down to a select group of developers.
iv. Cost-cutting can be done effectively in businesses owing to the fact that development is simpler and a single developer is required, instead of multiple developers for multiple platforms.
v. Subsequent updates and bug fixes can be rolled out for every platform affecting all platforms at once.

B. Disadvantages:

i. Native apps are designed to work flawlessly for the specified platform. There is a minimal lag with faster CPU render time. The performance is much improved compared to its cross-platform counterparts.
ii. Native apps have a deeper integration with the device. APIs for all device features like accelerometers, location-based services, cameras, and sensors work seamlessly with the native code.
iii. Native apps provide a richer user experience. Rendering of high-end graphics is only effectively possible with native app development.
iv. Deployment to the app store of the respective platform is usually frowned upon in the case of cross-platform approaches. They have a lower priority in the list of recommendations. Some apps which simply inject HTML and JavaScript code
v.
vi. inside a thin native container are rejected from getting published.
vii. The more complex an app becomes, app freezing and crashes become commonplace. This is because support for all devices is still practically infeasible with cross-platform solutions.

IV. General Categorization

Cross-platform mobile applications can be classified into four different categories based on their implementation:

A. Web Apps - A web app is an application that uses an Internet browser to run and has limited access to the underlying hardware of the mobile device [7]. HTML and JavaScript are the main technologies that are used to develop these, but they are usually accompanied by frameworks like Angular, jQuery, Django, Ruby on Rails and so on.
B. Hybrid Apps - These apps, like web are essentially developed using HTML and JavaScript but are embedded inside software called containers. Apache Cordova, previously known as PhoneGap is a very popular container used to develop Hybrid Apps
C. Interpreted Apps - These apps emulate native apps by allowing users to interact with the user interface components that are specific to a particular platform. These applications can be developed using Ruby, XML, and JavaScript etc. React Native, a technology open-sourced by Facebook in 2015 is a frontrunner in this category.
D. Widget Based Apps - All components in these apps are widgets which serve as the building blocks to form the application user interface. These apps use their own graphics engine to generate native application code. Flutter, which was initially released in May 2017 is becoming increasingly popular among cross-platform application developers.

V. Evaluating Cross-Platform Alternatives

This section will go through each of the mentioned categories, taking an example from each category and expanding on its characteristics and the architecture which make them unique from the other approaches. A comprehensive evaluation of the tools will provide the means for drawing inferences from our study.

A. Web apps:

Web apps are much like websites in the sense that the source code exists on a remote server, it does not utilize the device's CPU memory or RAM directly. Their content is rendered onto the browser of the device. This also means that this type of app does not require installation. Web apps follow a three-tiered architecture – first comes the thin client layer of the device, then the working application layer and lastly, the database layer. [6]

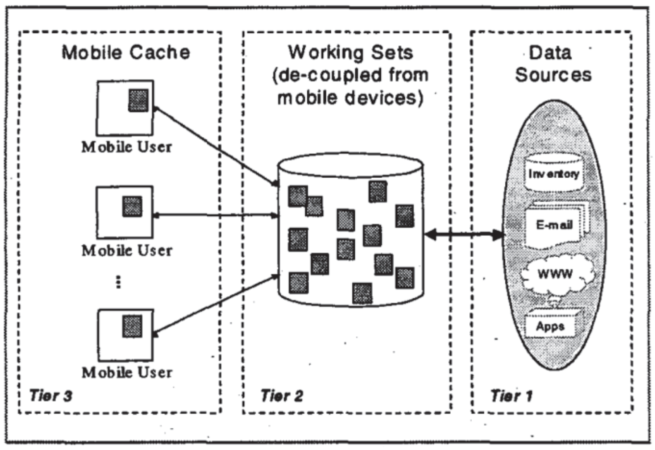

Fig. 1. 3-tiered architecture for web apps

Since it relies on a web server for the data as well as the source code, this also means that the user must always require an active Internet connection. It uses the web protocols for communication, which is why HTML, CSS, and JavaScript are its main languages for development. Recently, an emerging trend of HTML5 web apps in the fifth revision of HTML has been developed and recommended by the World Wide Web Consortium ever since its initial release in 2014. HTML5 enables developers to make web apps with a much deeper integration into the devices internal functionalities. Its use promises access to device features through API calls. As of 2018, there have been numerous mechanisms which service data storing, video presentations and complex content formatting [7]

Angular is an open-source TypeScript-based front-end web app framework. It is not to be confused with AngularJS, also known as Angular 1 which is based on the MVC (model-view-controller) design. Developed by Google and initially released in 2016, Angular is a complete rewrite of AngularJS which seeks to improve the structure and development environment. Unlike the traditional MVC design of most web app frameworks, Angular does not make use controllers and scopes, instead, it uses components and directives to control the flow of logic and the view of the DOM. Angular is intentionally built with mobile support as its objective.

The architecture follows a modular and hierarchical fashion, with the root module providing the bootstrap mechanism that launches the app. DOMs are the way the structure of a document is accessed and manipulated. It provides a means of hierarchical arrangement and interpretation of the components in a DOM. The components used in Angular 2 and Angular 4 define views and services; views are the modifiable screen elements manipulated by logic and code, whereas services function as dependencies for efficient use of code. Components include a template for how the view should look like, which is generally customizable by the developer. The templates themselves are governed by directives which provide instructions for the templates on how to behave and how DOMs are rendered. Each component follows an object-oriented paradigm – they have a class which contains the relevant data and logic. The metadata for the service classes is made ready through Dependency Injections (DI).[8][9]

Dependency Injections are a way to separate the logic of creating dependent objects from the main object in consideration thereby, taking away the developer's responsibility to explicitly include dependencies and only focus on the required flow of logic. Angular uses binding markup for connecting data to the app, this can be done dynamically either through Event Binding, i.e., user interaction with the app, or Property Binding, i.e., directly using calculated values from the app data.

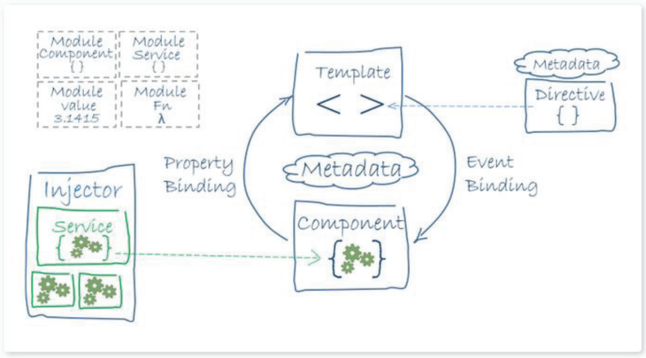

Fig. 2. A top-level view of Angular framework [9]

B. Hybrid Apps:

The defining characteristic of hybrid apps is that it combines native parts along with conventional Internet technologies such as HTML, CSS, and JavaScript. Loosely speaking, it is a cross between native apps and web apps. They usually work by injecting HTML code into a thin native container. This derives another property of hybrid apps which is, having a native app like experience. Unlike web apps, it is installable on the client's device even though the data and logic for the program come from a remote server. The development process initiates by coding in HTML and JavaScript without knowing details of the target platform. Access to the devices' underlying hardware is done through calls of specialized APIs which are not directly invoked by the programmer, rather, the hybrid cross-platform tool takes care of that - translating API calls to suit the environment of the target platforms selected.

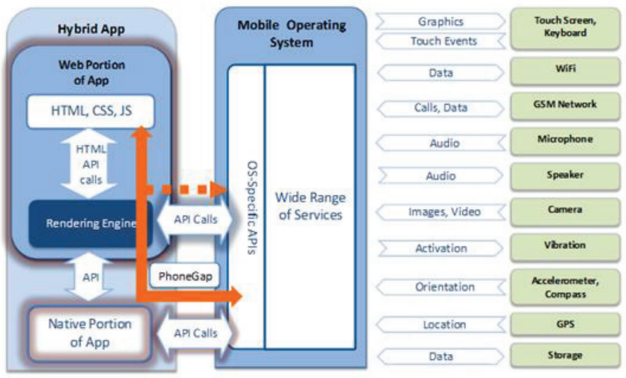

Fig. 3. Hybrid apps architecture [10]

Apache Cordova, much like other cross-platform tools, is an open-source mobile development framework which makes use of web standards such as HTML, CSS, and JavaScript. Apache Cordova is the open-source version of Adobe PhoneGap, which was acquired, modified, and rebranded by Apache. The code itself runs as a web page using web technologies. What Cordova provides is a container for running the code inside a native wrapper which is device specific. Cordova provides its own WebView for the application to run in - a sort of a placeholder for the user interface. The WebView and the Plugins are the defining characteristics of Apache Cordova. While WebView deals with the user interface, the plugins are more closely tied to deal with the native device features such as data storage, cameras, and the accelerometer. It is also interesting to note that since the code is similar to web app development, it is also compatible with the previously mentioned Angular framework for its internal logic and architecture using the MVC pattern [11].

Plugins bind device APIs to the application, which makes it easy to be invoked simply by using JavaScript. This provides developers with a universal API for each feature, abstracting away device specific APIs. Common plugins are included in the Core Plugins set, while other lesser known plugins can be developed by the community at large. To illustrate the process better, let us take an example. The plugin for Geolocation is contained inside the Core plugins set. Whenever a developer wishes to access the API for all platforms, they will explicitly include the geolocation plugin in their code and make a call to it based on a universal JavaScript API call. Internally, Cordova will translate the JavaScript API call to a bunch of other native device-specific APIs contained within the plugin - and make a call to it on the developer's behalf for all mobile platforms [12].

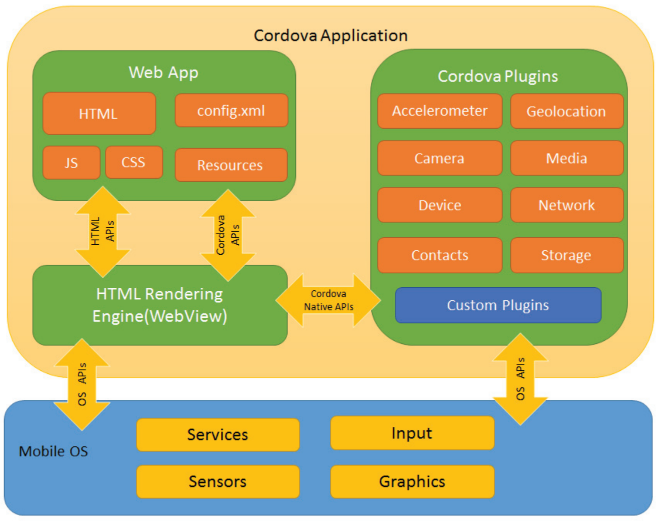

Fig. 4: Apache Cordova's architecture [13]

C. Interpreted Apps:

Interpreted apps get their name from their ability to deploy the application source code directly to the mobile device where it is interpreted using a JavaScript engine. Therefore the JavaScript engine used here is also called the interpreter and its type depends upon the platform used, for example - JavaScriptCore also known as Nitro which was developed by Apple is used for iOS whereas V8 an open-source engine managed by Google is used for Android. An application programming interface specially written to support cross-platform allows users to interact with the platform-specific user interface. The primary advantage of these apps is that native mobile apps can be created using simple JavaScript. Therefore these apps like native apps give better performance and interact better with the device hardware compared to other cross-platform development approaches in general.

Titanium developed by Appcelerator is a popular interpreted app development tool that uses its own Titanium Software Developer Kit or SDK to create native apps [14]. While there are many such other tools such as Rhodes, Fuse etc. React Native has been stealing the limelight since its release in 2015 and rightly so.

React Native is an open source technology released by Facebook in March 2015 and is based on their approach to "learn once, write anywhere" [15]. The apps are built using JavaScript but are indistinguishable from natively developed apps in Android or iOS. It is built upon the ReactJS, a JavaScript framework that was developed and released by Facebook in 2013. The traditional DOM (Document Object Model) is imperative, which is focusing on how the app operates and provides a way of manipulating documents with the help of programming languages like JavaScript. React however follows a declarative method which is focusing on what to do rather than how to do, thereby providing a level of abstraction for the developers and making the app development process easier. It uses Virtual DOM that stores a copy of the document object model in memory and changes only the updated portions instead of reloading the entire DOM [16]. This is very advantageous compared to natively built apps as the developer does not have to wait for the entire app to build again after making minor changes and can simply reload and see the updated version.

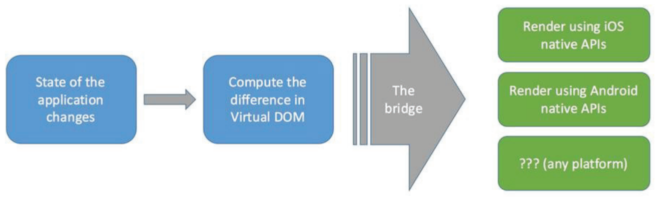
State of the application changes
Compute the difference in Virtual DOM
The bridge
Render using iOS native APIs
Render using Android native APIs
??? (any platform)

Fig. 5. A flow of events displaying an updated version of an app using React Native. [16]

The core of React Native has an abstraction layer called the "bridge" which allows it to invoke platform specific APIs needed for rendering components of the native view controller. So while building web apps with ReactJS the bridge accesses the Browser's DOM whereas while building apps via React Native the bridge will access APIs in Swift or Objective-C for iOS and APIs in Java or Kotlin for Android as shown in Figure 3. A JavaScript engine or the Interpreter is also used along with the bridge to interact with the UI elements. React Native uses JavaScriptCore for iOS and V8 for Android as an interpreter.

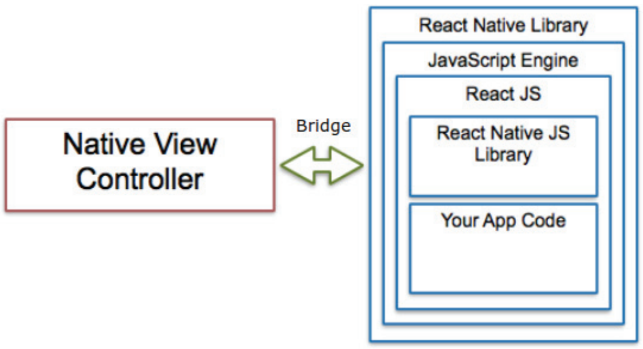

Fig. 6. Architecture of a react native app [17]

Figure 6, adapted from an online reference [17] depicts the general architecture of a react native app and its connection with the native view controller or user interface elements via the bridge.

D. Widget Based Apps:

Application widgets are anything that can define a structural element (e.g. drop-down menus or buttons). A stylistic element (e.g. theme or font color) or a layout feature (e.g. margin or padding). A widget based application treats every component as a widget thereby following a unified object model. Rather than depending on the device's native components it dynamically creates its own components using its own rendering engine by converting the application code (e.g. JavaScript or Dart) into native Code. Flutter a technology publicly released by Google in May 2017 falls into this special category.

Flutter is an open source UI framework and mobile software development kit (SDK) created by Google that not only provides cross-platform support but also enhances the application performance. The application code is written in Dart, a programming language that was created by Google [5].

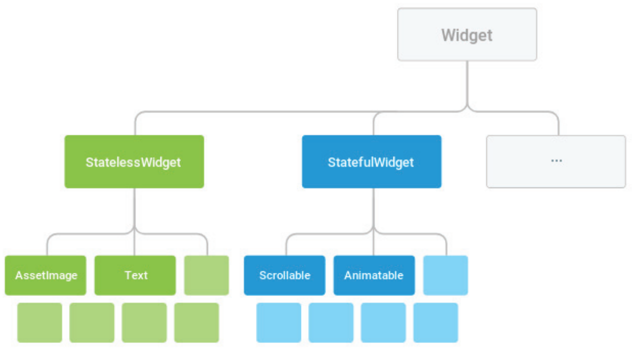

Fig. 7. Widget structure in Flutter [18]

The Flutter widgets can be categorized into two types, stateful and stateless widgets as shown in Figure 7 [18].

- A stateless widget is immutable which implies it does not respond to user's actions and cannot be overwritten preventing the views from reloading.
- A stateful widget is mutable and can change its composition dynamically while responding to the user's actions.

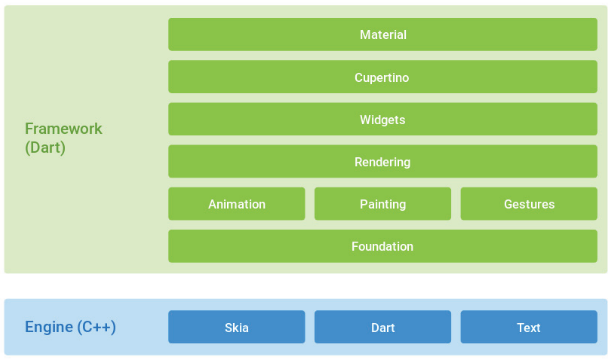

Fig. 8. Flutter framework architecture

Flutter uses Dart for creating components using the Skia 2D graphics engine. At the lowest level all the UI code uses Skia to render the application UI (Figure 3). Flutter runs its framework code and low-level engine code inside a lightweight Dart virtual machine. The framework code is written in Dart whereas the graphics engine is implemented in C++ [19]. The Dart code is compiled into native code using a feature called AoT (Ahead of Time) Compilation. The engine's C/C++ code is compiled with:

- Android's Native Development Kit (NDK) for Android
- Low-Level Virtual Machine (LLVM) for the iOS platform

Figure 8. shows how Flutter is divided into two parts. The framework that uses Dart comprises of - a library for design also called Material class, an iOS-style widget library called Cupertino and other libraries for animations, gestures etc. The Foundation library contains all lowest-level utility classes and functions used by all the other layers of the Flutter framework [20]. The Engine part mainly comprises of the 2D graphics engine called Skia and the Dart Virtual Machine.

VI. INFERENCE

Since our work is concerned primarily with the descriptive analysis of cross-platform tools, we list out certain parameters which best highlight the differences in the different app development approaches:

A. **UI/UX** - The app look and feel, considering the user interface and the user experience.
B. **Potential Users** - The crowd for which the app is targeted to, which is usually segregated based on how many platforms the app can be deployed to. Flutter only ships its apps to Android and iOS, however, this might change.
C. **Development Cost** - The time and money required for building the app, ground-up using the particular approach.
D. **App Security** - The ability for unauthorized personnel to access or "hack" into the logic or data of the app.
E. **Ease of Update** - The time required for the app to reflect its update across all platforms once deployed.

F. **Implementation Complexity** - How difficult or easy is it for the developer to create the app using a specific cross-platform tool

G. **Access to Native APIs** - This feature demonstrates the apps ability to tap into the device's hardware, such as accelerometers and sensors.

H. **Development Environment** - Native apps use development tools which are proprietary for the platform in question. Flutter, owned by Google, also uses Android Studio as its platform.

I. **License** - From a development point of view, Apple developers need to pay a fee to develop iOS apps, even its platform is source-closed in nature. For Google's platforms and Web-based frameworks, development is usually open-source in nature.

J. **Language** - Native apps use languages for development as specified by the platform, i.e, Java/Kotlin for Android and Swift/Object-C for iOS. Flutter uses Google's very own C-style language, Dart.

K. **Publishing to Marketplace** - Assuming that one follows the guidelines laid out by the platform, the chances of any app getting published on an application platform marketplace like Google's Play Store or the App Store for iOS, are subject to how "native" the app looks and feels.

TABLE 1: EVALUATION OF CROSS-PLATFORM CATEGORIES

Decision Parameters	Native	Web apps	Hybrid apps	Interpreted apps	Widget Based (Flutter)
UI / UX	Excellent	Moderate	Moderate	Fairly Good	Very Good
Potential Users	Limited	Maximum	Large	Large	Limited
Development Cost	High	Low	Low	Moderate	Moderate
App Security	Very High	Very Low	Low	Moderate	High
Ease of Update	Low	High	Varying	Varying	Moderate
Implementation Complexity	High	Low	Moderate	Low to Moderate	Moderate
Access to Native APIs	Yes	Yes, but only with HTML5	Yes, through plugins	Yes	Yes
Preferred Development environments	Android Studio, XCode, Momentics	Visual Studio Code, PhpStorm, WebStorm	Eclipse Hybrid Mobile Tools (THyM), Intel XDK,	Nuclide using Atom IDE, Appcelerator Studio	Android Studio, IntelliJ
License	Android: Open-source, iOS: Closed-source	Open-source	Generally open-source	Open-source	Open-source
Language	Java/Kotlin, Swift/Objective-C, etc.	HTML, CSS, JavaScript, TypeScript, etc.	HTML, CSS, JavaScript, Node.js etc.	JavaScript, JSX, UX Markup, etc.	Dart
Publishing to Marketplace	Yes	No	Yes	Yes	Yes

From Table 1 based on our own extensive study and other references [3][21], we can infer that Native apps are a better choice when it comes to performance-related parameters like access to native device APIs, ease of update, rendering UI and providing the better user experience. These apps, however, require platform specific specialization and often are cost intensive.

Web apps are essentially mobile-friendly versions of websites which are more interactive than simple websites - it is a handy approach useful for when the app itself is not too demanding of resources or complexity but requires some element of user interaction. They are a better choice if ease of implementation and the development time is the developer's primary concern but they have to keep in mind that the app cannot get published in the app marketplace.

This drawback can be overcome by Hybrid apps which also can access native device APIs but on the downside, they are not as easy to develop. Hybrid apps combine native components with the primitive WebView offered by web apps. The tools which follow this approach usually follow a component-based structure which injects HTML code into native containers based on event-driven actions. They also have a closer integration with the device as native APIs calls are managed by the tool's plugins. s. It is a compromise between truly native apps and web apps, most useful for creating interactive but simple user-friendly apps. Interpreted apps not only have better security compared to web and hybrid apps but have better access to device hardware as they essentially are native apps generated from cross-platform tools. Interpreted apps often use web development technologies like JavaScript and are therefore easier to develop compared to purely native apps. A widget based app or Flutter has the following advantages:

A. Like purely native apps it has almost complete access to the underlying device hardware
B. It uses Dart which is easy to implement close to although not exactly as simple as web apps
C. It is better than interpreted apps as it generates its own widgets compared to interpreted app tools like React Native that are dependent on the device original equipment manufacturer (OEM) widgets
D. It is developed by Google that ensures high-security standards and maintenance

Keeping the above factors in mind one can deduce that Flutter combines the advantages of almost all application development tools. Although it cannot completely offer the amount of hardware integration like purely native apps it seems to be the best choice among all cross-platform alternatives.

VII. Conclusion

The purpose of our paper is to perform a comprehensive analysis of the cross-platform applications. In this paper, we introduce cross-platform development tools and contrast it with native app development. Cross-platform approaches require developers to compromise on the user experience - a native touch and feel, can only be accomplished by the use of truly native components inside the app. A deeper integration with the device and improved performance are invaluable parameters to be considered when developing apps. However, we found that the use of cross-platforms tools usually outweighs the disadvantages that come with it, except for when, for example, creating high-end gaming apps such as Asphalt.

Reviewing different types of cross-platform approaches and taking an example from each category gave us an idea about the characteristics of each type of approach and how useful they are in different scenarios. Cross-platform frameworks despite their many advantages are still in their preliminary stages in the market. It has barely been a decade with the advent of interpreted apps and yet newer innovative approaches such as Flutter continue to be developed which propel the future of cross-platform mobile app development.

While the ultimate choice of cross-platform tools rests with the developer, we tried to shed some light on the methodology to adopt when making such apps. In the near future, we look forward to carrying out a formal assessment by testing a simple version of a cross-platform app developed from each framework and henceforth comparing our inferences with the quantitative values obtained.

Acknowledgment

We would like to thank Prof. Krishna Samdani for his useful comments and guided assistance that helped us follow a proper methodology while conducting our review.

References

[1] "Number of smartphone users worldwide 2014-2020." Internet: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Oct. 2, 2018].

[2] "Mobile Operating System Market Share Worldwide." Internet: http://gs.statcounter.com/os-market-share/mobile/worldwide [Oct. 5, 2018].

[3] N. P. Huy and D. vanThanh, "Evaluation of mobile app paradigms," in *Proc. MoMM '12 10th International Conference on Advances in Mobile Computing & Multimedia*, 2012, pp. 25-30.

[4] M. Palmieri, I. Singh and A. Cicchetti, "Comparison of cross-platform mobile development tools," *2012 16th International Conference on Intelligence in Next Generation Networks*, Berlin, 2012, pp. 179-186.

[5] W. Wu "React Native vs Flutter, Cross-platforms mobile application frameworks." B.E. thesis, Metropolia University of Applied Sciences, Finland, 2018.

[6] S. Helal, J. Hammer, J. Zhang and A. Khushraj, "A three-tier architecture for ubiquitous data access," *in Proc. ACS/IEEE International Conference on Computer Systems and Applications*, Beirut, Lebanon, 2001, pp. 177-180.

[7] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications*,"* in *Proc. BCI '13 6th Balkan Conference in Informatics*, 2013. pp. 213-220.

[8] N. Jain, P. Mangal and D. Mehta. "AngularJS: A Modern MVC Framework in JavaScript." *Journal of Global Research in Computer Science*, vol. 5, pp. 17-23, Dec. 2014.

[9] *"*Angular Docs." Intenet: https://angular.io/guide/architecture [Oct. 10, 2018].

[10] "Mobile App Services." Internet: http://www.directiontech.com/dt7/mobile-apps [Oct. 10, 2018].

[11] E. Palme, "How to wrap an Angular app with Apache Cordova." Internet: https://medium.com/@EliaPalme/how-to-wrap-an-angular-app-with-apache-cordova-909024a25d79, Feb. 1, 2018 [Oct. 12, 2018].

[12] S. Bosnic, I. Papp and S. Novak, "The development of hybrid mobile applications with Apache Cordova," *2016 24th Telecommunications Forum (TELFOR)*, Belgrade, 2016, pp. 1-4.

[13] "Architectural overview of Cordova platform - Apache Cordova." Internet: https://cordova.apache.org/docs/en/latest/guide/overview/index.html [Oct. 12, 2018]

[14] C. Brousseau. (2013, October 25). *Creating Mobile Apps with Appcelerator Titanium*. [On-line]. Available: https://www.packtpub.com/application-development/creating-mobile-apps-appcelerator-titanium [Oct. 15, 2018].

[15] W. Danielsson, "React Native application development : A comparison between native Android and React Native." M.S. Dissertation, Linköping University, Sweden 2016.

[16] J. Warén. "Cross-platform mobile software development with React Native." B.B.A. thesis, Haaga-Helia University of Applied Sciences, Finland, 2016.

[17] B. Khichar, "Awesome React Native: Building Android App with JavaScript." Internet: https://codebrahma.com/awesome-react-native-building-android-app-javascript/, Jun. 13, 2018 [Oct. 16, 2018].

[18] "Technical Overview." Internet: https://flutter.io/technical-overview/ [Oct. 16, 2018].

[19] J. Fayzullaev "Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter." B.E. thesis, South-Eastern Finland University of Applied Sciences, Finland, 2018.

[20] "Foundation Library." Internet: https://docs.flutter.io/flutter/foundation/foundation-library.html [Oct. 16, 2018].

[21] I. Dalmasso, S. K. Datta, C. Bonnet and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Sardinia, 2013, pp. 323-328.