An Overview of Modern Cross-platform Mobile Development Frameworks

Jelica Stanojević, Uroš Šošević, Miroslav Minović, Miloš Milovanović
Faculty of Organizational Sciences
University of Belgrade
Jove Ilića 154, 11010 Belgrade, Serbia
{jelica.stanojevic, uros.sosevic, miroslav.minovic, milos.milovanovic}@fon.bg.ac.rs

Abstract. *Interest and usage of Cross-platform mobile development frameworks (CMDF) is growing over the years. This paper outlines trends in the field of mobile development, specifically in CMDFs area which is rapidly evolving with new emerging approaches for creating mobile applications. The most popular cross-platform mobile frameworks today are Flutter, React Native, Cordova, Ionic and Xamarin, respectively. Mentioned frameworks cover both predominant mobile operating systems (Android and iOS) and also provide support for desktop and web applications. The aim of this paper is to shed light on currently (2022) most popular CMDFs in order for academia to follow the rapid industry development in this field.*

Keywords. Mobile development, Cross-platform, Flutter, React Native, Cordova, Ionic, Xamarin

1 Introduction

In dynamic environments where the proper timing of application market entry is of great importance building applications with a Write Once, Run Anywhere mobile application tools (Shah, Sinha, & Mishra, 2019) could be beneficial in contrast to building native apps targeting each platform separately which will increase time, effort and cost in order to satisfy predominant operating systems. This is especially important in the case of early stage startups where the efforts are focused towards building the Most Viable Product in a short time span. In such situations where time, effort, cost and cross-platform compatibility are important, developers tend to choose CMDF approach rather than native approach for developing applications (Shah et al., 2019). On the other hand, developers need to understand if and which CMDF could meet their needs.

An analysis from 2015 shows that 3.73% out of about 12000 mobile applications available on Google Play Store, were created using CMDF hybrid approach (Malavolta, Ruberto, Soru, & Terragni, 2015). A study from 2022, which was done on a dataset consisting of about 650 thousand mobile applications, shows a higher value of 8,67%. This could indicate a growth in interest in the hybrid development approach (Biørn-Hansen, Grønli, Majchrzak, Kaindl, & Ghinea, 2022). Regardless of the approach, all mobile applications made with CMDFs occupy 15% of the dataset which was analyzed by (Biørn-Hansen et al., 2022).

According to the worldwide survey conducted by JetBrains, and which included over 32000 software developers, predominant cross-platform mobile frameworks in 2021 were Flutter[1], React Native[2], Cordova[3], Ionic[4] and Xamarin[5] (JetBrains, 2022). These results are in line with Google Trends search queries for these technologies over the past few years, specifically 2020-2022 ("Google Trends Comparison (React Native vs. Flutter vs. Ionic vs. Xamarin vs. Cordova)," 2022).

Android and iOS represent the two most dominant mobile phone operating systems with a total market share of about 70% and 25% respectively (Laricchia, 2022). Mentioned frameworks cover both predominant mobile operating systems and also give a solution for desktop and web applications.

There aren't many research papers that include and compare the majority of the top 5 most popular CMDFs nowadays which are previously mentioned in this paper. Deciding from various options of CMDF with different underlying paradigms that offer different functionalities and performance based on use cases isn't always straight-forward. That is why it is important to conduct their in-depth analysis and comparison in order to include more important parameters in making this decision.

2 Methodology

The aim of this paper is to shed light on currently (2022) most popular CMDFs in order for academia to follow the rapid industry development in this field. Explanation of

[1] https://flutter.dev/
[2] https://reactnative.dev/
[3] https://cordova.apache.org/
[4] https://ionicframework.com/
[5] https://dotnet.microsoft.com/en-us/apps/xamarin

different types of applications that CMDFs produce will be given and mostly used frameworks will be categorized on this basis. Different possible taxonomies will be mentioned. Main characteristics of predominant CMDFs will be described. Lastly, discussion regarding already mentioned CMDFs based on existing research in this area will be realized and conclusions will be made. Google Scholar was the main database used in research of existing work on this topic. Papers included in the Related Work section are relatively new Journals, Magazines and Conference Proceedings published during 2018-2022 with a focus on review and comparison of modern CMDFs.

3 Cross-platform Mobile Development Frameworks

Beside native approach in which Android apps are created using Java or Kotlin and iOS apps using Swift or Objective-C programming language, one of the approaches for developing mobile applications is using cross-platform mobile development frameworks. Types of applications produced by CMDF could be grouped into following categories (Xanthopoulos & Xinogalos, 2013):

- Web Apps,
- Hybrid Apps,
- Interpreted Apps and
- Generated Apps.

The new taxonomy presented by Nunkesser in 2018, expands previously noted groups, categorizing mobile applications as:

- Pandemic, which are subdivided as
    - Web App
    - Hybrid Web App - Cordova and Ionic
    - Hybrid Bridged App - React Native and Flutter
    - System Language App,
- Endemic (native apps) and
- Ecdemic/Foreign Language App (like Xamarin), which can be
    - Interpreted App,
    - Generated App,
    - VM App.

Hybrid Web Apps is a match for Hybrid Apps in (Xanthopoulos & Xinogalos, 2013) categorization and Ionic and Cordova belong to this category without any disagreement in academic papers or industry. That isn't the case for the rest of the top 5 CMDFs. This is mainly because the term hybrid, in industry and academia, is used for approaches that utilize web technology but without building a strictly defined hybrid app (Martinez, 2019; Singh & Shobha, 2021; Vishal & Kushwaha, 2018).

In 2018, two predominant approaches were Hybrid and Interpreted (Biørn-Hansen & Ghinea, 2018), which can now be appended with Cross-compiled, Widget-based approach like Flutter.

M-site is another approach for creating mobile apps which results with web applications, applications that are created using widespread web technologies (e.g. HTML, CSS and JavaScript) and users access them through a web browser (Zohud & Zein, 2021). Since the main goal of this paper is to research CMDF that produce mobile applications which can be downloaded and installed on mobile devices from Google Play Store or App Store, and interact with native devices APIs, this type of apps isn't going to be in the focus of this paper.

CMDF which produces hybrid types of applications, leverages the common feature of different mobile operating systems, the mobile browser accessible from the native code (Charland & Leroux, 2011). Hybrid applications are built based on web technologies, HTML, CSS and JavaScript, which are browser-supported languages, and are wrapped as native apps inside a special native container (Que, Guo, & Zhu, 2016; Zohud & Zein, 2021) called WebView[6] in Android OS and WKWebView[7] in iOS. The name hybrid comes from a combination of native and web applications features (Que et al., 2016).

The WebView, which deals with user interface and gives better access to device capabilities, and the Plugins, which are intended to deal with native device features (e.g. storage, camera etc.) could be considered as main characteristics of hybrid apps CMDFs (Shah et al., 2019; Zohud & Zein, 2021). Plugins bind device APIs to the application, which makes it easy to be invoked simply by using JavaScript that provides developers with a generic API for each feature to bridge all the service requests from the web-based code to the corresponding platform API, abstracting away device specific APIs (Pinto & Coutinho, 2018; Shah et al., 2019)**.** Examples of Hybrid CMDF that are currently popular (2022) are Cordova and Ionic.

Logic in interpreted applications is implemented in a platform-independent way, while their native code is automatically generated to implement the user interface with platform-specific native UI elements (Nunkesser, 2018; Xanthopoulos & Xinogalos, 2013). The native features are provided by an abstract layer that interprets the code on runtime across different platforms to access the native APIs (Latif, Lakhrissi, Nfaoui, & Es-Sbai, 2017). These types of apps provide the look and feel of native applications because of native user interfaces (S.Thakare, Shirodkar, Parween, & Parween, 2014). On the downside, new platform-specific features, like new user interface features of a new android version, can be available to apps only when supported by the development environment so there is the complete dependence on the software development environment (S.Thakare et al., 2014). React Native is nowadays (2022) an actual CMDF that produces interpreted apps (Biørn-Hansen & Ghinea, 2018; Huber, Demetz, & Felderer, 2020; Shah et al., 2019).

[6] https://developer.android.com/reference/android/webkit/WebView

[7] https://developer.apple.com/documentation/webkit/wkwebview

As for generated applications, which are also called cross-compiled (S.Thakare et al., 2014), their code, also written in common programming languages, is compiled to a specific native code for each targeted platform (Latif et al., 2017; Xanthopoulos & Xinogalos, 2013). Because of the generated native code these types of apps achieve high overall performance (S.Thakare et al., 2014). Xamarin is currently (2022) one of the most used CMDFs which create this type of apps (Latif, Lakhrissi, Nfaoui, & Es-Sbai, 2016). Flutter is a cross-compiled framework (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020) but also belongs to a special category of Widget based apps (Shah et al., 2019).

In next chapters a brief overview of each mentioned which represents also the most popular CMDFs is going to be given.

3.1 Cordova

Apache Cordova is an open-source version of PhoneGap which was acquired by Adobe in 2011. Cordova is a hybrid CMDF so its main characteristics are a container, WebView, for running the code (HTML/CSS/JavaScript) inside a native wrapper which is device specific and Plugins which are dealing with the native device features. Plugins bind device APIs to the application, which makes it easy to be invoked simply by using JavaScript. (Shah et al., 2019)

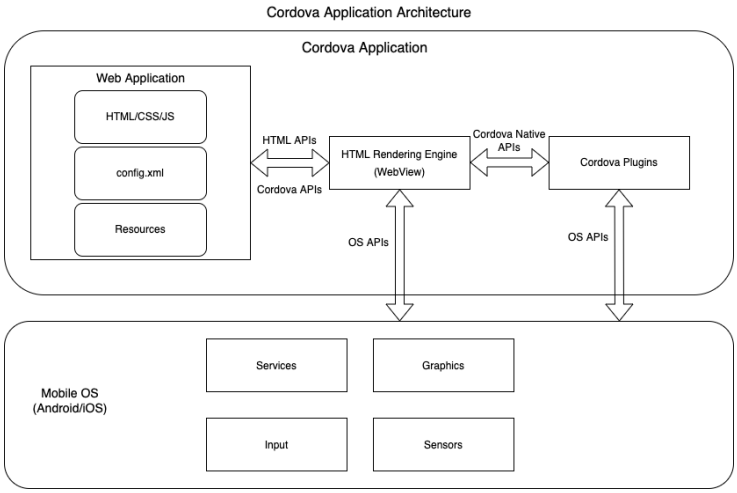

Figure 1. Cordova Application Architecture (The Apache Software Foundation, n.d.)

3.2 Ionic

As previously mentioned, Ionic represents an example of Hybrid App CMDF. First version of Ionic Framework was released in 2013 and was built on top of AngularJS[8], which is now deprecated (from January 2022), and Apache Cordova. In 2016 release of version 2 of Angular framework with drastic changes also caused version 2 of Ionic framework. Back then, Ionic components, which eased development of mobile UI, were built as Angular directives. That meant that Ionic framework
is coupled with Angular and with the rise of other Single Page Application (SPA) frameworks like React[9] and Vue[10] that became a problem. With adoption of Web Components standard which represents a suite of different technologies allowing you to create reusable custom elements - with their functionality encapsulated away from the rest of your code - and utilize them in your web apps (MDN Contributors, 2022) Ionic released version 4 in 2019 with components based on this standard. That way, Ionic was decoupled from the Angular framework, and Ionic components can be used with other SPA frameworks. Ionic team even created a tool, named Stencil[11], for creating components. Ionic 6 is the currently active version (from December 2021).

Beside components, another important part of Hybrid CMDF like Ionic, are tools like Cordova and Capacitor that help bridge the gap between web apps and native device features. The Ionic team developed Capacitor in 2018, and while it has similarities with Cordova, it is using new modern APIs which were not available when Cordova was released (Lynch, n.d.) and is considered as Ionic's official native runtime (Ionic team, n.d.).

[8] https://angularjs.org/
[9] https://reactjs.org/
[10] https://vuejs.org/
[11] https://stenciljs.com/

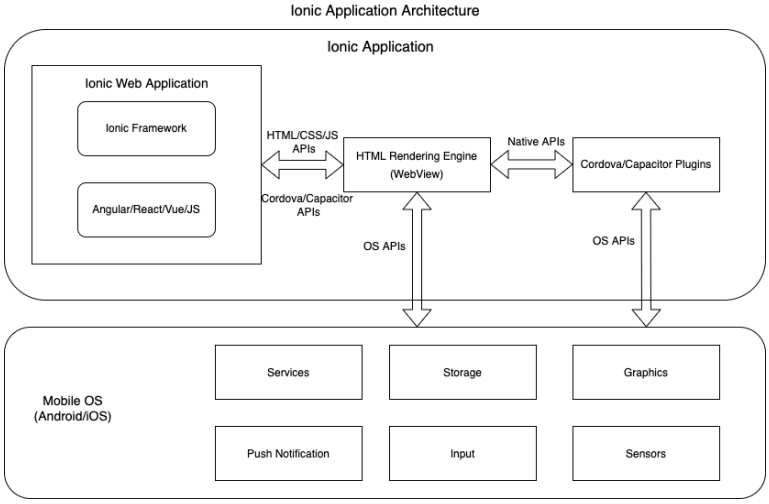

Figure 2. Ionic Application Architecture (Singh & Shobha, 2021)

3.3 React Native

React Native is nowadays the most popular Interpreted Apps CMDF. React Native started at an internal Meta Hackathon and was released in March 2015 by Meta as an open-source framework. Same as Ionic, React Native will feel familiar to web developers because they can create mobile apps using JavaScript (Pinto & Coutinho, 2018) so their learning curve is eased.

React Native is using a JavaScript engine, JavaScriptCore, in order to interpret (hence the categorization) deployed application source code directly to the mobile device (Shah et al., 2019). At runtime, React Native creates the corresponding Android and iOS views for React Native components. Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. (Meta Platforms, 2022)

The bridge layer allows to invoke platform specific APIs, Swift or Objective-C APIs for iOS and Java or Cotlin APIs for Android, needed for rendering components of the native view controller (Shah et al., 2019). Also, if an interpreted app requires access to a native feature (e.g. camera), it will pass the request down through the respective bridge handling such requests, and return a result up through the same bridge where needed, e.g. to return a value from the native-side to the app-side (Biørn-Hansen & Ghinea, 2018).

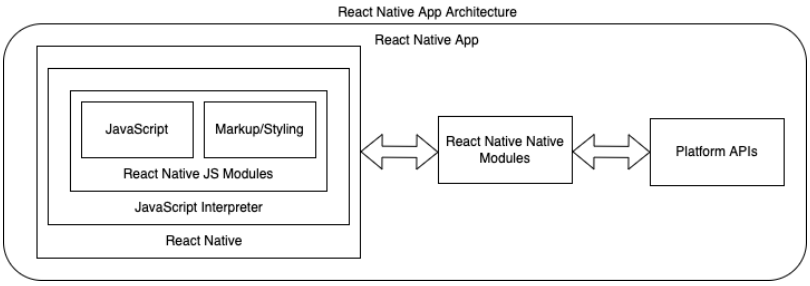

Figure 3. React Native Application Architecture (Biørn-Hansen & Ghinea, 2018)

First announced in 2018, recently, in 2022, React Native rolled out new architecture with the new native TurboModule system for interaction with the native side and the Fabric Renderer, new rendering system, meaning the bridge is splitted in two parts (Sciandra, 2019).

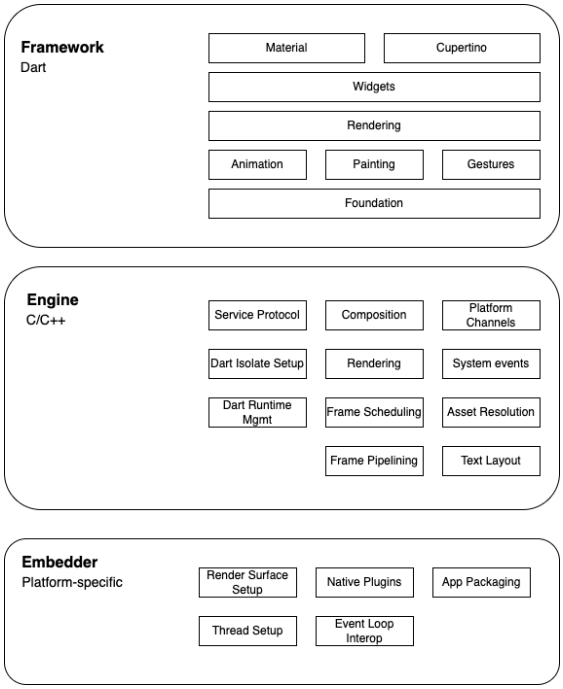

Figure 4. Flutter architectural overview (Flutter, n.d.)

3.4 Flutter

Flutter became the most popular CMDF lately. It was created by Google and the first version was released in 2017. Flutter applications are made using Dart programming language[12] which is similar to JavaScript.

Flutter framework includes a rich set of platform, layout, and foundational libraries for animation, painting, gestures and design like Material and Cupertino libraries which offer comprehensive sets of controls to implement the Material or iOS design languages. (Flutter, n.d.; Shah et al., 2019) It uses Dart for creating components using the Skia 2D graphics engine which is used for rendering the application UI (Shah et al., 2019). A platform-specific embedder, written in appropriate platform language, provides an entrypoint; coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input; and manages the message event loop. Using the embedder, Flutter code can be integrated into an existing application as a module, or the code may be the entire content of the application. (Flutter, n.d.)

3.5 Xamarin

Example of Generated apps CMDF is Xamarin, developed by Microsoft since 2016, which uses C# in order to create applications for various devices.

Xamarin.Forms, which was introduced in Xamarin 3 in 2014, represents an open-source UI framework that allows developers to create user interfaces in XAML with code-behind in C# which will be rendered as native controls on each platform (Johnson, Britch, & Dunn, 2021). The process of accessing native utilities is simplified by abstraction, a library Xamarin.Essentials, that provides cross-platform APIs for native device features (Justin Johnson, David Britch, Craig Dunn, & Hemant Arya, 2021)**.**

Microsoft's evolution of Xamarin.Forms, .NET Multi-platform App UI[13] (.NET MAUI), is a framework for building modern, multi-platform, natively compiled iOS, Android, macOS, and Windows apps using C# and XAML in a single codebase. Conversion from Xamarin.Forms to .NET MAUI is basically complete in 2022. Xamarin support will continue through November 2023 (Ramel, 2022).

[12] https://dart.dev/

[13] https://dotnet.microsoft.com/en-us/apps/maui

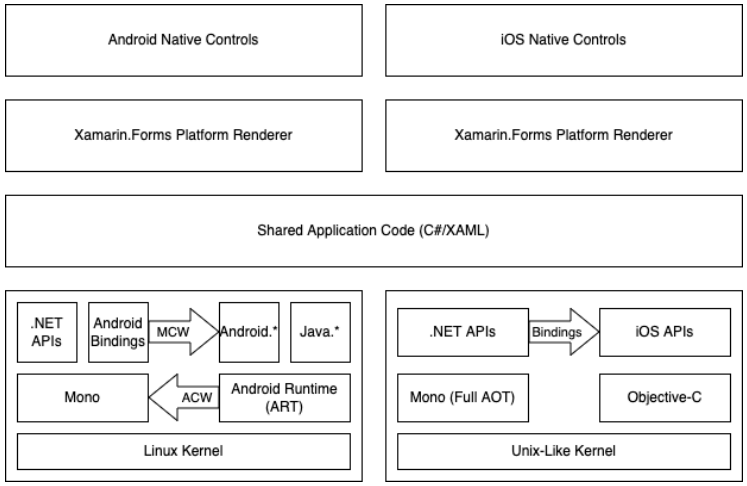

Figure 5. Xamarin.Forms Architecture (Johnson et al., 2021)

4 Related work

As for cross-platform approaches in general it can be concluded that developers need to compromise on the user experience which is of highest quality on native devices (Shah et al., 2019), but it's important to notice that it is improving in the newer releases of CMDF and that performance of mobile devices is constantly improving. For example, based on a study which was carried out by Huber, Demetz, & Felderer (2020) React Native and Android Native apps in terms of UI fluidness are comparable which results in better user experience in React Native cross-platform solution, but also comes with a consequence of high load on CPU and main memory. On the other hand Ionic has insignificant better performance compared to React Native concerning mentioned aspects, but missing the benefits of a low janky frame rate and low GPU memory usage which results in performance degradation perceived by the user (Huber et al., 2020).

In a research paper by Biørn-Hansen, Grønli, & Ghinea (2019) device hardware impact and penalties caused by transitions and animations (in-app page navigation, opening side menu and complex animation - Lottie animation[14]) were studied and reports are made on following metrics: frames per second (FPS), CPU usage, device memory usage and GPU memory usage. Eight different mobile apps were created: native iOS and Android, and also Ionic, Xamarin and React Native for iOS and Android. Even though the performance of running complex animation in Native apps is much better in
contrast to Ionic Hybrid App, Biørn-Hansen, Grønli, & Ghinea (2019), from users' perspective, couldn't observe the difference between these apps. This indicates that users' experience should be taken into account while measuring performances for certain parameters (in this case FPS).

Mobile apps created using CMDFs, specifically Ionic/Capacitor and React Native, compared to natively developed applications put a higher load on CPU usage, main memory and GPU memory consumption (Huber et al., 2020). Except in the use case of building high-end gaming applications, CMDFs usually prevail over the disadvantages that come with their usage (Shah et al., 2019).

When it comes to implementation and development of communication bridges for using native device features in Hybrid (Ionic-Cordova) and Interpreted (React Native) approaches Biørn-Hansen & Ghinea (2018) conclude that these aren't complex nor tedious programming tasks in both cases. However, the Hybrid-based Ionic-Cordova app proved to be five times faster than the Interpreted app built with React Native for the functionality of fetching images stored on native devices (Biørn-Hansen & Ghinea, 2018).

On the other hand, in comparative study (Quazi & Sinha, 2018) of Ionic, React Native and PhoneGap, React Native proved to be the finest CMDF based on execution time to perform the hardware and database related operations. React Native is also the most popular from the industrial point of view based on multiple case-study (Zohud & Zein, 2021) carried out in four different software development companies in Palestine in 2021, even though Ionic is more used than React Native since its older.

Shah et al. (2019) conclude that Flutter is overall the best choice among all CMDF which combines the advantages of development tools from other approaches.

Differences regarding performances for CMDF on different platforms, iOS and Android, exist. Cordova apps exhibit significantly different patterns of rendering time, which can be attributed to the differences in the underlying HTML rendering engines and the JavaScript engines of the respective platforms, and UI response on the iOS and Android

[14] https://lottiefiles.com/72-favourite-app-icon

platform. (Jia, Ebone, & Tan, 2018) This is also in line with study of Biørn-Hansen, Grønli, & Ghinea (2019) in which Android and iOS are found to differ greatly in terms of memory consumption, CPU usage and rendered FPS for both the native and cross-platform apps.

There are some papers regarding industry perspective on CMDFs and their usage, but this area represents the potential for further exploration due to the rapid pace of development in the field, in order for academia to stay relevant with industry standards in mobile development (Biørn-Hansen, Grønli, Ghinea, & Alouneh, 2019). Based on a survey conducted in 2016 (Biørn-Hansen, Grønli, Ghinea, et al., 2019) which was targeting industry practitioners, the most popular frameworks were Phone Gap (which is now deprecated), Ionic and React Native, both in hobby and professional projects. Biørn-Hansen et al. (2019) concluded that industry is moving towards using React Native and Ionic which is in line with today's statistics (JetBrains, 2022). Obtained conclusion in another, already mentioned study (Zohud & Zein, 2021), carried out over industry practitioners, is that developers experience is the most influential factor in the development process. This could indicate that web developers who are familiar with SPA framework like Angular would prefer to work with Ionic, rather than learn new CMDF from scratch like Flutter. This is also the case with React Native since it uses JavaScript, and on the other hand for Flutter, Dart, its own development programming language must be learned (Işitan & Koklu, 2020).

Conclusion of Biørn-Hansen et al. (2020) is that certain cross-platform frameworks can perform equally well or even better than native on certain metrics. In this case metrics were measured for tasks of creating contact, reading from file system, accessing location information and usage of accelerometer for Ionic, React Native, NativeScript, Flutter, MAML/MD_2 and Native Android. No single framework scores best across all features in the study, nor do they state that a silver bullet solution exists. This indicates that well-defined technical requirements and specifications are very important before choosing between CMDF (or not choosing CMDF) since bad decisions can potentially lead to underperforming apps (Biørn-Hansen et al., 2020).

Based on literature study it can be observed that there aren't many studies which include the comparison of the majority of beforehand mentioned frameworks, especially Flutter as newest vs. older ones.

5 Conclusion

The Flutter and React Native ecosystems are in its rise and are constantly improving. On the other hand, Ionic, as representative of hybrid mobile applications CMDF, is going to stay relevant in the future because of the possibility that web developers with experience in SPA frameworks can quickly adapt to the Ionic framework and start building mobile applications. As for Xamarin.Forms, it is yet to be seen how its evolution .NET MAUI is going to be accepted.

The further analysis of mentioned, most recent and currently most popular CMDFs based on different aspects and criteria could be of great use when it comes to decision making of choosing between different approaches and frameworks. Since there is constant improvement of mobile devices and also CMDFs, the interesting topic for further research would be measuring real users' experience of using apps written in mentioned CMDFs (and native languages) in various use cases along with performance measurement which can affect it. This could help to draw a conclusion if the higher performance parameters are very noticeable and therefore crucial in decision making.

The fast-paced development of new versions and emerging of new CMDF and also staying updated with technical advances is a challenge for both industry and academia. Continuous research in this field is especially important due to the rapid pace at which cross-platform frameworks are being released, updated and deprecated. (Biørn-Hansen, Grønli, & Ghinea, 2019)

References

Biørn-Hansen, A., & Ghinea, G. (2018). *Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development*. 8.

Biørn-Hansen, A., Grønli, T.-M., & Ghinea, G. (2019). Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance. *Sensors*, *19*(9), 2081. https://doi.org/10.3390/s19092081

Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., & Alouneh, S. (2019). An Empirical Study of Cross-Platform Mobile Development in Industry. *Wireless Communications and Mobile Computing*, *2019*, 1–12. https://doi.org/10.1155/2019/5743892

Biørn-Hansen, A., Grønli, T.-M., Majchrzak, T. A., Kaindl, H., & Ghinea, G. (2022). *The Use of Cross-Platform Frameworks for Google Play Store Apps*. Presented at the Hawaii International Conference on System Sciences. https://doi.org/10.24251/HICSS.2022.934

Biørn-Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development

frameworks. *Empirical Software Engineering*, *25*(4), 2997–3040. https://doi.org/10.1007/s10664-020-09827-6

Charland, A., & Leroux, B. (2011). Mobile application development: Web vs. native. *Communications of the ACM*, *54*(5), 49–53. https://doi.org/10.1145/1941487.1941504

Flutter. (n.d.). Flutter architectural overview. Retrieved June 6, 2022, from https://docs.flutter.dev/resources/architectural-overview

Google Trends comparison (React Native vs. Flutter vs. Ionic vs. Xamarin vs. Cordova). (2022, June 9). Retrieved June 9, 2022, from Google Trends website: https://tinyurl.com/CMDFComparison

Huber, S., Demetz, L., & Felderer, M. (2020). Analysing the Performance of Mobile Cross-platform Development Approaches Using UI Interaction Scenarios. In M. van Sinderen & L. A. Maciaszek (Eds.), *Software Technologies* (pp. 40–57). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-52991-8_3

Ionic team. (n.d.). Cordova Community Plugins. Retrieved June 6, 2022, from https://ionicframework.com/docs/native/community#capacitor-support

Işitan, M., & Koklu, M. (2020). Comparison and Evaluation of Cross Platform Mobile Application Development Tools. *International Journal of Applied Mathematics Electronics and Computers*, 273–281. https://doi.org/10.18100/ijamec.832673

JetBrains. (2022, February 21). Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021. Retrieved May 30, 2022, from Statista website: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

Jia, X., Ebone, A., & Tan, Y. (2018). A performance evaluation of cross-platform mobile application development approaches. *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 92–93. Gothenburg Sweden: ACM. https://doi.org/10.1145/3197231.3197252

Johnson, J., Britch, D., & Dunn, C. (2021, July 8). What is Xamarin.Forms? Retrieved June 6, 2022, from https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms

Justin Johnson, David Britch, Craig Dunn, & Hemant Arya. (2021, July 8). What is Xamarin? Retrieved June 6, 2022, from https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

Laricchia, F. (2022, February 7). Mobile operating systems' market share worldwide from January 2012 to January 2022. Retrieved May 28, 2022, from Statista website: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

Latif, M., Lakhrissi, Y., Nfaoui, E. H., & Es-Sbai, N. (2016). *Cross platform approach for mobile aplication development: A survey*. 5.

Latif, M., Lakhrissi, Y., Nfaoui, E. H., & Es-Sbai, N. (2017). Review of mobile cross platform and research orientations. *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 1–4. Fez, Morocco: IEEE. https://doi.org/10.1109/WITS.2017.7934674

Lynch, M. (n.d.). Capacitor vs Cordova: Hybrid Mobile App Development. Retrieved June 6, 2022, from https://ionic.io/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development

Malavolta, I., Ruberto, S., Soru, T., & Terragni, V. (2015). Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation. *2015 2nd ACM International Conference on Mobile Software Engineering and Systems*, 56–59. Florence, Italy: IEEE. https://doi.org/10.1109/MobileSoft.2015.15

Martinez, M. (2019). Two Datasets of Questions and Answers for Studying the Development of Cross-Platform Mobile Applications using Xamarin Framework. *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 162–173. Montreal, QC, Canada: IEEE. https://doi.org/10.1109/MOBILESoft.2019.00032

MDN Contributors. (2022, April 27). Web Components. Retrieved June 6, 2022, from https://developer.mozilla.org/en-US/docs/Web/Web_Components

Meta Platforms. (2022, March 30). Core Components and Native Components. Retrieved

June 10, 2022, from https://reactnative.dev/docs/intro-react-native-components

Nunkesser, R. (2018). *Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development*. 5.

Pinto, C. M., & Coutinho, C. (2018). *From Native to Cross-platform Hybrid Development*. 669–676. IEEE. https://doi.org/10.1109/IS.2018.8710545

Quazi, F. U. R., & Sinha, N. (2018). Android-Platform Based Determination of Fastest CrossPlatform Framework. *International Journal of Computer Science and Mobile Computing*, *7*(9), 1–12.

Que, P., Guo, X., & Zhu, M. (2016). A Comprehensive Comparison between Hybrid and Native App Paradigms. *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, 611–614. Tehri, India: IEEE. https://doi.org/10.1109/CICN.2016.125

Ramel, D. (2022, May 24). .NET MAUI Reaches General Availability, Replacing Xamarin.Forms. Retrieved June 10, 2022, from https://visualstudiomagazine.com/articles/2022/05/24/net-maui-ga.aspx

Sciandra, L. (2019, April 9). The New React Native Architecture Explained: Part Three. Retrieved June 10, 2022, from https://formidable.com/blog/2019/fabric-turbomodules-part-3/

Shah, K., Sinha, H., & Mishra, P. (2019). Analysis of Cross-Platform Mobile App Development Tools. *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, 1–7. Bombay, India: IEEE. https://doi.org/10.1109/I2CT45611.2019.9033872

Singh, M., & Shobha, G. (2021). Comparative Analysis of Hybrid Mobile App Development Frameworks. *International Journal of Soft Computing and Engineering*, *10*(6), 21–26. https://doi.org/10.35940/ijsce.F3518.0710621

S.Thakare, B., Shirodkar, D., Parween, N., & Parween, S. (2014). State of Art Approaches to Build Cross Platform Mobile Application. *International Journal of Computer Applications*, *107*(20), 22–23. https://doi.org/10.5120/18868-0389

The Apache Software Foundation. (n.d.). Apache Cordova Documentation—Overview. Retrieved June 17, 2022, from https://cordova.apache.org/docs/en/11.x/guide/overview/index.html

Vishal, K., & Kushwaha, A. S. (2018). Mobile Application Development Research Based on Xamarin Platform. *2018 4th International Conference on Computing Sciences (ICCS)*, 115–118. Jalandhar: IEEE. https://doi.org/10.1109/ICCS.2018.00027

Xanthopoulos, S., & Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. *Proceedings of the 6th Balkan Conference in Informatics on - BCI '13*, 213. Thessaloniki, Greece: ACM Press. https://doi.org/10.1145/2490257.2490292

Zohud, T., & Zein, S. (2021). Cross-Platform Mobile App Development in Industry: A Multiple Case-Study. *International Journal of Computing*, 46–54. https://doi.org/10.47839/ijc.20.1.2091