



Trends in Web Based Cross Platform Technologies

Ruben Smeets¹, Kris Aerts²

¹ES&S, DTAI, KU Leuven Technology Campus Diepenbeek,

Agoralaan gebouw B bus 8, 3590 Diepenbeek, Belgium

¹ruben.smeets@kuleuven.be; ²kris.aerts@kuleuven.be

Abstract— Cross platform mobile application development using web technologies has traveled a long way since its inception back in 2008-2009 [1]. Frameworks such as Apache Cordova/Phonegap [2] or Appcelerator [3] enabled web developers to reuse their exiting development skills to build applications for a wide variety of mobile platforms and opened up app development to a massive community. Since then many technologies have been developed: some failed, some succeeded. In this paper we provide a comprehensive overview of cross platform tools using web development technologies and identify trends that emerged from this landscape.

Keywords— Review paper, cross-platform tools, hybrid applications, Phonegap/Cordova, mobile frameworks.

I. INTRODUCTION

Mobile apps have become an important part of our society. The time we spend on our mobile device is increasing every year [4]. However, the time spent on developing mobile apps tends to increase even more, especially when each mobile platform must have a dedicated, native application. The platform diversity introduced by the mobile vendors and mobile operating systems, as well as the steady improvement of hardware and software capabilities of the mobile devices is causing increased fragmentation across platforms and even within a specific platform's ecosystem. Separate development teams are required to produce and maintain different codebases for a single application and this for each supported mobile platform, unless a cross platform development strategy is applied.

A popular form of cross platform development is web based, not only because the technologies are similar, but also because there is a vast availability of web developers. These developers can safely rely on their web development skills to build mobile apps. A typical web based mobile app is called a hybrid app which can be divided into two separate parts: a web portion that contains the elements that are shown to the user; and a bridging mechanism that provides access to advanced features of the native platform, such as hardware sensors, network information, and GPS location. In this paper we refer to these types of hybrid apps as WebView-based hybrid apps. Another form of a web based mobile app can be achieved by simply embedding an existing website without using the bridging mechanism. However, this type is not the focus of our paper. Other approaches for cross platform development exist as well, which require different programming skills [5], [6].

In this paper we focus on web based approaches, starting from the early days with simple libraries and microframeworks that quickly matured to scalable mobile frameworks in Sect. II, followed by the state of

WebView-based hybrid apps in Sect. 0. In Sect. III we discuss the counter movement of blended hybrid apps who try to increase performance by mixing native with web. Subsequently, we analyse the abandonment of WebViews by runtime-based hybrid frameworks in Sect. IV. Finally, we conclude the paper with insights on "what might come next" in Sect. V.

II. FROM JAVASCRIPT LIBRARIES TO SCALABLE MOBILE FRAMEWORKS

As mentioned in Sect. 0, a WebView-based hybrid app consists of two parts: the web portion and the bridging mechanism. This section discusses how the web portion evolved from simply supplying access to the mobile platform's WebView to large scale (opinionated) frameworks.

The first WebView-based hybrid mobile apps were merely nothing more than a Single-Page-Application (SPA). This is a web application that requires only a single page load in a web browser. Interaction with the application typically involves dynamic communication with the web server to update the current page view. The web portion of such a hybrid app lives inside the mobile platform's WebView, which allows it to use any HTML, CSS and JavaScript that is supported by the WebView version. However, the WebView of mobile devices was severely lagging behind desktop browsers. Therefore, technological advances were first seen on web development for desktop browsers, slowly dripping through to mobile app development.

The introduction of Google's Gmail in 2005 can be seen as the starting point of feature rich SPAs. At that time there were only a handful of more or less mature JavaScript frameworks and libraries, most of them focusing on similar tasks such as utility functions that enhance JavaScript (e.g. Prototype [7]), cross-browser Document Object Model (DOM) manipulation (e.g. jQuery [8]), and scaffolding for user interface (UI) components (e.g. Dojo [9]). People managed to build great apps with these tools, but gradually the need arose for more architectural solutions.

The arrival of Github (2008) [10], which provided an easy cooperative platform for developers, in combination with the rapidly changing browser environment (HTML5), resulted in a diverse expansion of the JavaScript ecosystem. This led to a new era of frameworks called microframeworks: a term used to refer to minimalistic web application frameworks. The microframeworks started to enforce a style of coding and/or an architectural style that allowed the creation of increasingly complex SPAs. This evolution demanded the creation of crucial tools that helped shape the modern SPAs of today. Examples of these tools are listed in Table 0 [11].

TABLE I
JAVASCRIPT TOOLS THAT HELPED SHAPE THE MODERN SPAS [11]

Tool category	Description	Examples
Code organization	Splitting code in a set of highly decoupled, specific pieces of functionality, called modules. Facilitating increased maintainability of an application.	AMD, RequireJS, CommonJS
Code sharing	Providing a packaging format to encapsulate reusable pieces of code and the utilities required to share these pieces with other apps	Bower, NPM
Build automation	Automate building processes associated with complex apps, e.g. syntax checking, code minification, running tests, packaging, etc.	Grunt, Gulp

The continuous improvement of the mobile browser [12] and the release of platforms like Apache Cordova/Phonegap that can bridge to native features of the mobile devices, led the microframeworks and libraries to the mobile environment. Here they were faced with new specific challenges of mobile such as: feature support, less processing power and memory but high expectations regarding performance, touch interaction, and screen size diversity. This demanded the creation of very lightweight and efficient solutions for the mobile web. It is out of these solutions that the modern mobile frameworks and libraries were born.

Today we can divide the modern mobile frameworks and libraries into three categories: the UI frameworks/libraries; the architectural frameworks; and the combined frameworks. The first group focuses on providing UI widgets or components that are optimized for mobile platforms (e.g. Ionic [13], Kendo UI [14]). The second group provides structure to applications in the form of design patterns, like Model-View-Controller (MVC). They increase the maintainability and scalability of applications and allow the creation of complex SPAs. Additionally, they improve the code quality by incorporating established best practices (e.g. Angular [15]). The last group combines both the UI libraries and the architectural frameworks in a single all encompassing solution (e.g. Sencha [16]). Note: The first and second categories are often used in combination, e.g. Ionic+Angular.

A. Emerging Trends of Modern Mobile Frameworks

Most of the modern mobile frameworks and libraries contain all the core elements for creating complex SPAs for the mobile web: offline storage abstraction, MV* design patterns, routing, templates, dependency injection, scaffolding, and others. Yet there are still some limitations that restrict the complexity and scalability of larger applications [17]. The combination of these limitations and the support of new features in the ES2015 standard [18] of JavaScript and new browser APIs (e.g. web components) sets the stage for the next generation of frameworks. The following list discusses trends in these frameworks that are gaining adoption.

1) *Self-contained Components*: Self-contained components describe a UI component that manages its own view, styles, and data logic. This enforces a greater separation of concerns by encapsulating view elements that prevent other code from arbitrarily reading and changing properties of the component. Data communication with the component is only possible if explicitly approved. As a result, the components are easier to test and easier to reuse. ReactJS [19] and Angular2 [20] are two candidates that apply this concept.

2) *DOM Optimisation*: DOM manipulation is used extensively throughout WebView-based hybrid apps. The introduction of a virtual DOM in ReactJS from Facebook, an in-memory minimalistic representation of the actual DOM, provided significant performance improvements [21], [22]. A similar feature can now be found in Angular 2, and many others will probably follow.

3) *Universal Rendering or "Isomorphism"*: Web applications can be rendered on the server and on the client from the same code base. Combining these two mechanisms can speed-up initial load times of the application while increasing the discoverability by search engine optimisation (SEO) [23]-[25]. However, not all apps can benefit from this approach as explained by M. Ubl [26].

4) *One-way Data Flow and Functional Programming*: SPAs are becoming increasingly complicated causing developers to manage more state. This state can be seen as dynamic application data that is constantly changing over time. With one-way data flow we can make state mutations predictable by imposing certain restrictions on how and when updates can happen. Doing so, greatly simplifies the application's flow and consistency, which results in code that is easier to test and debug [27]. This trend towards a more functional programming style can also be seen in other languages such as the lambda's in Java 8 and .NET. Some even advocate JavaScript as a functional programming language [28].

5) *Semantically Structured JavaScript*: JavaScript is dynamically typed with lots of coercions going on. The type of a variable is generally not known at compile time, not by inference nor by declaration. This allows for a flexible and dynamic coding style, but is considered an improper coding technique by many because the code becomes harder to understand and refactor, and errors harder to track. A solution for this is TypeScript [29], a superset of JavaScript that adds optional static typing and provides planned features from future JavaScript editions. It increases the code quality by catching errors at compile time instead of runtime and enhances the readability. Additionally, it lowers the barrier of front-end development for developers coming from strongly typed backgrounds like C++ and Java [30].

It is interesting to see how these current developments will shape the next generation of frameworks that help developers create increasingly complex front-end SPAs.

B. Some Concerns...

The constantly changing web environment indicates a continued investment in web technologies, which is a good thing. However, there are downsides to this fast evolution. The following list describes a few.

1) *JavaScript framework fatigue*: The continuous evolution of the web in general causes the creation of new frameworks, libraries and tools almost every other day. Web developers are paralyzed by the amount of choice and loose sight of which frameworks are best suited for their application. There are tools and resources that help bring sanity in this chaos e.g. the TodoMVC project [31]. Nevertheless, this phenomenon will likely continue to cause frustration for many beginners and experienced developers alike [32], [33].

2) *Vendor lock-in*: The specific syntax, design pattern implementations and architectural choices of frameworks cause developers to invest large amounts of time in learning a particular framework. Additionally, in most cases there is little to no code-sharing between various frameworks or even subsequent versions of the same framework, e.g. AngularJS and Angular2.

3) *Complex development stacks*: Complex development stacks are a result of the previous concerns combined with performance implications caused by some frameworks (e.g. large framework sizes). This drives many experienced web developers to opt for dedicated libraries instead of monolithic frameworks. They combine numerous libraries and tools into complex development stacks that work for their particular

needs. Beginners are often overwhelmed by this wide offer of development stacks causing more choice paralysis [34].

4) *Security risks*: The open source nature of frameworks and libraries causes trust issues amongst users and developers. Users are uncertain of the origin of the code and the intentions of the creator [35]. A recent event concerning the JavaScript package manager "npm" has brought this issue to the surface [36].

C. Key Take-aways

The next generation of frameworks and libraries incorporates many lessons learnt from years of SPA development. It is clear that the focus of these newer frameworks lies on scalability, maintainability as well as performance of larger and more complex SPAs.

The framework/library diversity encourages innovation, yet the abundance of choice causes many frustrations. Nevertheless, it remains important to thoroughly evaluate your options before making a decision. Projects like TodoMVC can help you narrow down the right tool for the job.

The continued investment in JavaScript is causing the language to become a universal programming language, see also Sect. IV.C. JavaScript is being picked up in other areas besides mobile and web: for instance, JavaScript for the Internet-Of-Things [37] or projects like the Electron Javascript framework [38] for building cross-platform desktop applications. The state of WebView-based hybrid apps

In this section we move from the web portion to the underlying WebView and bridging mechanism, the other part of WebView-based hybrid apps as introduced in Sect. 0. It is precisely this part that provides the traditional web technologies with access to advanced native functionality of the mobile platform, allowing web developers to create tightly integrated SPAs using their existing development skills. There are numerous articles explaining the advantages and disadvantages of WebView-based hybrid apps [39]-[41]. However, the focus of this section lies on the evolution of WebView-based hybrid, and in particular Phonegap [42] the most popular web-to-native wrapper, alongside with Apache Cordova, the open-source version of Phonegap released by Adobe.

A. Dealing with Problems from the Past

The evolution of WebView-based hybrid development has known many ups and downs since its popularization in 2011 with the acquisition of Phonegap by Adobe. An important down event was the abandonment from the likes of Facebook and LinkedIn in 2012. TJ VanToll has written a comprehensive article explaining their motives and describes the current solutions that address their concerns [43]. In this work he concluded that the main reasons for parting from WebView-based hybrid were the lack of tooling and the poor performance of the WebViews. The solutions to these problems, as described by TJ, are summarized below:

Solution for the lack of tooling problem

1) *Remote Debugging*: Since iOS 6, developers can use the full Safari Web Inspector to debug hybrid iOS apps. For Android, the Chrome DevTools are available as of version 4.4. These tools provide powerful debugging capabilities including memory management, DOM inspection, and JavaScript step-by-step execution.

2) *Cloud-based Builds*: remove the pain of manually managing multiple software development kits~(SDKs) for each supported platform. Additionally, they can provide a mechanism that allows automatic updates to be pushed to the user without passing through the app stores.

3) *Live-reload*: increases the development workflow by live updating the JavaScript, HTML, and CSS of your application during development. Changes are immediately reflected without requiring a new build.

4) *UI Frameworks*: are web frameworks that specialize in mobile UI widgets. They focus on a native look and feel and are optimized for performance on mobile WebViews. Examples are: Ionic, Framework 7, and Kendo UI.

5) *Backends*: Hybrid apps require the back end to be exposed over HTTP. Setting up such a back end can be time consuming because it involves setting up a database, an API for interaction, and require deployment on a server. For this reason, there are providers who offer these back ends as a service~(BaaS) [44].

Solution for the performance problem

The performance of WebView-based hybrid apps greatly depends on the underlying platform and the browser of that platform. Recently, there were some big improvements to the latter. These are shown in table II.

TABLE II
OVERVIEW OF ANDROID AND IOS WEBVIEW COVERAGE PER VERSION [48], [49]

Platform	WebView	Improvements	Coverage (%)
Android			
prior v4.4	Android Default (webkit)	Poor performance	27.3
v4.4	Chromium (blink)	Performance improvements across the board [45]. Better HTML5 support [46].	34.3
V5.0+	Chromium (blink)	Independent updates from the OS providing constant performance improvements.	38.4
iOS			
prior v8	UIWebView (Default JS engine)	Poor performance	5
v8	WKWebView (Nitro JIT engine)	Estimated 4x performance improvement compared to UIWebView [47].	16
v9+	WKWebView (Nitro JIT engine)	Fixed critical bug that prevented hybrid apps from loading local files.	79

B. New Concerns Arise that Require Attention

The solutions presented in the previous paragraph indicate that most of the tooling and performance problems have currently been addressed. Nowadays, we can create WebView-based hybrid apps with an acceptable degree of performance, while maintaining a rapid development cycle [50]. There are, however, some concerns that still require attention. We summarized the most important ones here.

1) *WebView Fragmentation*: The WebView fragmentation is a result of the platform fragmentation introduced by the mobile ecosystems. This causes WebView-based hybrid apps to have inconsistent feature support and performance across multiple version of the OS. As shown in table II, Android suffers the most from WebView fragmentation with its user base almost evenly distributed over the three WebView options. For iOS the situation appears better. However, a critical bug in iOS8 prevents WKWebView from being used properly in hybrid apps [51]. There are workarounds available, yet they do not offer a stable solution [52].

2) *Cordova Plugin Quality*: One of the key benefits of hybrid apps compared to pure web apps is the ability to access advanced native functionality through the concept of plugins. Most of these plugins are created and maintained by the open-source community. This has the advantage of providing a great diversity of plugins with high availability. On the other hand, with the exception of the core Cordova plugins, the open-source nature causes the creation of many plugins with questionable code quality. Developers that release the same plugin for multiple platforms are unlikely to provide the same code quality for each supported platform. That is, if they even provide support for multiple platforms. Furthermore, many plugins lack good documentation or become unavailable due to discontinued support [53].

C. Solutions

The concerns posed in Sect. 0.B are being tackled by many. Two of the better solutions are CrossWalk [54] and the verified plugin marketplace [55].

1) *Crosswalk - a Universal WebView*: A popular solution for the WebView fragmentation is the Crosswalk WebView created by the Crosswalk Project. This WebView is available for multiple platforms including Android and iOS. The Android version is based on the Chromium browser and provides consistent feature support and performance for Android devices with versions 4.0 and above. According to the Android developer dashboard they are able to support more than 97% of the active devices and thereby effectively solving the fragmentation problem for Android. Other advantages are: a controlled upgrade cycle, simple platform integration, advanced HTML5 support, and the ability to leverage Cordova plugins. The iOS Crosswalk WebView is based on the WKWebView and allows for custom runtime extension by the developer.

2) *Verified Plugin Markets*: To overcome the problem of unreliable plugins Telerik offers the verified plugin marketplace, providing plugins that are well maintained for multiple platforms [56].

D. 60 Frames per Second with WebView-based Hybrid Apps

Beautiful apps are often packed with a set of complex UI animations and transitions. This distinguishes them from their competition, but only when the animations are smooth, fluent and feel natural. Performance is essential. Because of the costly interaction with the DOM of the WebView, this is hard to achieve in a

WebView-based hybrid app. The developer is required to have a performance minded development flow from the start [57]. Directly using the hardware accelerated HTML5 canvas element of the WebView may be a solution and has the advantage of the hardware acceleration, but the disadvantage that the canvas element does not support the typical UI controls, only low level graphics operations. Two solutions tackle this problem: React-canvas [58] provides higher-level abstractions for the hardware accelerated canvas, but it is still in its early stages. SamsaraJS [59] provides a language for positioning, orienting and sizing DOM elements and animating these properties over time. It optimizes the DOM interactions by flattening the DOM-tree and provides batched hardware accelerated animations.

III. BLENDED HYBRID APPS: MIXING WEB AND NATIVE

In the previous section we explained that PhoneGap apps currently can offer *acceptable* performance. As explained by J. Morony [60], this is mostly due to a lack of understanding of the WebView limitations. While native apps receive good performance out-of-the box, hybrid apps require some tweaking to achieve an acceptable result. For this reason, solutions like Cordova started offering ways to mix native and web and thereby increase the performance of hybrid apps. These hybrid apps are called blended hybrid apps.

A. The Embedded Cordova WebView

Consulting the Cordova documentation [61], we notice that there are two development paths specified. The Cross-platform workflow and the platform-centered workflow. The former is the most popular approach and results in pure hybrid apps¹ that we have discussed in the previous sections. The latter approach describes the required workflow for blended hybrid apps.

The typical structure of such a blended hybrid app involves the combination of custom native components and one or more embedded WebViews, see Fig. 1. The native components are mostly static parts of the application that require less updates like toolbars, navigation bars, and page transitions. On the other hand, the frequently changing content is shown in the WebViews [62]. Examples of a blended hybrid app are the Apple's Apple Store application and Basecamp by Basecamp, LLC, shown in Fig. 2.

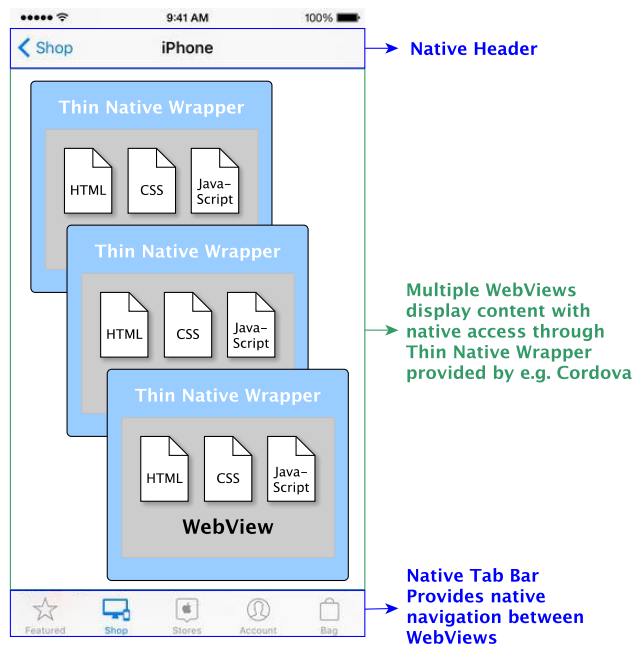


Fig. 1 Blended hybrid application structure

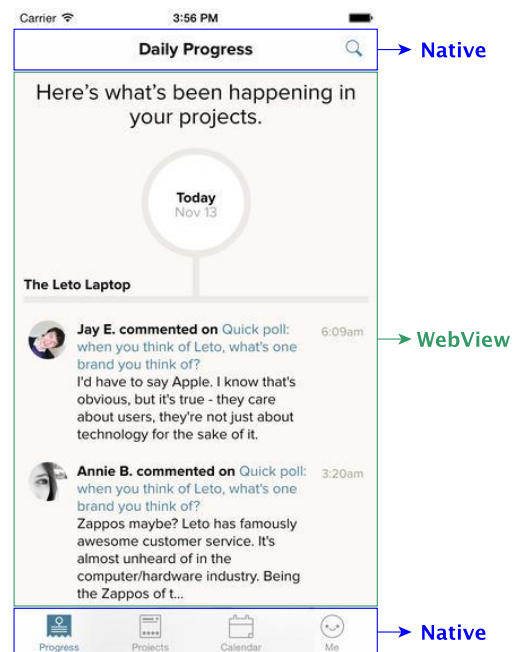


Fig. 2 Example of blended hybrid app, Basecamp App [64]

Applying this workflow combines the performance benefit of native components with the flexibility of WebViews. Frequently changing content shown in WebViews can be updated without passing through the app store approval processes and without bothering the user with having to update his or her app [63].

1) *Cross-platform Tool Support:* The embedded Cordova WebView approach requires the developer to be familiar with each supported native programming environment individually. As a result, cross-platform tool

¹ Pure hybrid app: consist of a single WebView that occupies the entire screen. All the content and navigation controls are implemented in HTML5 and a thin native wrapper (like Cordova) is used to expose native APIs to the HTML5 code.

vendors created solutions that provide abstraction over the platform specific requirements. Two of those tools are Appgyver's Supersonic UI [65] and TriggerIO [66]. By using these tools developers can focus on the hybrid part of the application without worrying about the native code for the static UI components.

B. Micro WebViews

Some apps go a step further in mixing native and hybrid by implementing the largest part of the application natively and using standard WebViews for frequently changing and/or little used content. An example of this approach is the Instagram app. Instagram's timeline is displayed inside a WebView, allowing the developers to have fast iteration and feedback of new UI designs. Another way of using WebViews as part of a native application is in the later stages of user flow in mobile commerce apps. Features like checkout and payment are displayed in WebViews because the APIs are harder to integrate with native screens [67]. Examples are the Walmart and Belk mobile applications.

C. Key Take-aways

The "pure" hybrid app approach allows the highest code re-usability across multiple platforms, whereas the blended approach (mixing hybrid and native) gives the developer a way to integrate more tightly with the native platform.

Building blended hybrid apps requires the developer to thoroughly analyse his/her application's requirements in order to identify where to use WebViews and where to use native components. Profiling tools may be used to find the performance bottle necks and help decide which parts should be native. Another perspective can be to use either standard WebViews (or Crosswalk WebViews) for parts whose layout is frequently updated, and Cordova empowered WebViews for advanced native functionality. Doing so combines the benefit of native performance with the flexibility of hybrid.

Good hybrid applications are difficult to spot. Due to the confirmation bias on the performance of HTML5-based applications, hybrid app developers are not advertising the use of WebViews inside their apps [68].


























IV. ABANDONING WEBVIEWS: JAVASCRIPT RUNTIME SOLUTIONS

Although the blended approach does provide better performance, most mobile applications that use WebViews are still not on par with native apps [50]. To circumvent the costly interaction with the WebView's DOM a different kind of hybrid technology is being created that uses web technologies without relying on a WebView for displaying content. These solutions are known as native JavaScript frameworks or JavaScript runtime frameworks. This technology is not fully mature yet, but the current achievements are very promising.

A. Possible Candidates

JavaScript runtime solutions combine native UI components with a JavaScript virtual machine (JVM) to provide a truly native application that is written with web technologies. Typically, the application logic is executed by the JVM while the native UI components and features are delegated from the JVM through a JS-to-native bridge. Table 0 lists the possible candidates and summarizes their current state, combined with the popularity metrics from social media.

TABLE III
JAVASCRIPT RUNTIMES (CONTENT CHECKED AT TIME OF WRITING) [70]-[73]

	 Titanium	 React Native	 Nativescript	 TabrisJS	 Smartface
Announced	2008	2015	2014	2014	2011
Versions	v5.3.0	v0.27	v2.0	v1.7	v4.5.0
Platforms	Android 4.0.x – 6.0.x iOS 7.1.x – 9.2.x	Android 4.1.x – 6.0.x iOS 7.0.x – 9.2.x	Android 4.2.x – 7.0.x iOS 7.1.x – 9.2.x	Android 3.7.x - 5.x iOS 6.x - 8.x	Android 4.2.x – 6.0.x iOS 7.1.x – 9.2.x
Popularity	 11963  23528  2088  2494	 11658  82092  33481  4750	 2900  12588  7142  475	 634  265  404  2	 63  1100  18  302

Although these numbers are subjective, they do indicate the amount of interest from the community. This is especially important for the open source solutions that rely on the community for providing support and continued development. Here we notice that the relatively new solutions, React Native from Facebook and Nativescript from Telerik, are gaining a lot of attention. Part of their success stems from the new concepts they introduced. For Nativescript this new concept is direct access to platform specific APIs from within JavaScript, allowing 0-day support of new APIs and platform versions. (Titanium Appcelerator is trying something similar with a project called Hyperloop, which is still experimental [69]). On the other hand, React Native brings the innovative view structuring principles of ReactJS to mobile, considering the UI to be a pure function of state and props. The new concepts provided by these solutions ensure a healthy competition that drives forth improvement and innovation.

B. Comparing Runtime-based Hybrid Applications with Native Apps

Runtime-based hybrid frameworks try to combine the advantages of web development with the performance of native apps. This is shown in Table III. It remains an open question whether they will succeed in this mission.

TABLE IV
COMPARING WEB, NATIVE, AND RUNTIME BASED HYBRID DEVELOPMENT

Advantages	Web	Native	Runtime-based
Use web technologies (HTML/CSS/JavaScript)	X		X
Same code and technologies across platforms	X		X
Fast iteration cycle (hot-reloading)	X		X
Scalable solutions for large teams and complex applications (frameworks)	X		X
Over the air updates (without passing through App Stores)	X		X
Fast and responsive UI		X	X
Complex gestures and smooth animations		X	X
UI is consistent with platform		X	X

C. Universal Hybrid Applications using JavaScript

The focus of hybrid application development has mostly been on the philosophy of write-once, run everywhere with hybrid solutions trying to create a cross platform application using a single code base. However, due to the underlying differences of the various platforms and the different UI conventions, sharing UI code is often not possible or results in the lowest common denominator UI. Furthermore, sharing UI between web and mobile presents a greater challenge due to the contrasting interaction conventions. For this reason, modern solutions like React Native and Nativescript+Angular2 [74] are parting from that philosophy and embrace the learn once, write anywhere paradigm. Titanium Appcelerator was the first to apply this idea in 2011 [75]. Nevertheless, React Native is the one that popularized it. Developers are able to use React Native and ReactJS to target mobile and web using a consistent developer experience based on JavaScript and their framework principles. Similarly, Nativescript in combination with Angular2 is aiming for the same goal [76].

The central idea is that the main productivity gains can be obtained from applying a single paradigm and a universal programming language, more than from code sharing. Future will tell whether this will turn out to be true.

V. CONCLUSION

In this paper we provided an overview of the past, current and possibly future state of hybrid mobile application development. We learned that WebView-based hybrid approaches have resolved many of their past issues concerning performance and tooling and are now able to facilitate complex SPA development with modern JavaScript frameworks and libraries. In addition, we gained insights on the variety of WebView based hybrid solutions that range from single WebView apps to blended multiple WebViews, increasing the flexibility of hybrid mobile application development. Finally, we discussed a new breed of hybrid application frameworks, called native JavaScript frameworks that combine native UI performance with the flexibility of web development to create truly native apps using JavaScript. The philosophy of these frameworks facilitates a consistent development experience for mobile and web, while providing the best performance for each individual platform.

ACKNOWLEDGEMENT

This work is funded by the Flemish government through the Vlaio/IWT-TETRA project 140332: Crossmos – Cost efficient development of advanced, cross-platform mobile applications.

REFERENCES

- [1] Phonegap blog. (2008) "Unofficial announcement of PhoneGap." [Online]. Available: <http://phonegap.com/blog/2008/08/07/unofficial-announcement-of-phonegap>
- [2] (2016) Cordova: Mobile apps with HTML, CSS & JS website. [Online]. Available: <https://cordova.apache.org>
- [3] (2016) Appcelerator: The Appcelerator platform website. [Online]. Available: <http://www.appcelerator.com>
- [4] Media Buying. (2015) "Growth of Time Spent on Mobile Devices Slows." [Online]. Available: <http://www.emarketer.com/Article/Growth-of-Time-Spent-on-Mobile-Devices-Slows/1013072>
- [5] S. Xanthopoulos, S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*. ACM, New York, NY, USA, 2013, pp. 213-220
- [6] Wafaa S. El-Kassas, Bassem A. Abdullah, Ahmed H. Yousef, Ayman M. Wahba, "Taxonomy of Cross-Platform Mobile Applications Development Approaches," *Ain Shams Engineering Journal*, [Online]. Available: <http://dx.doi.org/10.1016/j.asej.2015.08.004>, ISSN 2090-4479, 2015
- [7] (2016) Prototype: a foundation for ambitious web user interfaces website. [Online]. Available: <http://prototypejs.org>
- [8] (2016) jQuery: write less, do more website. [Online]. Available: <https://jquery.com>
- [9] (2016) Dojo: A JavaScript toolkit that saves you time and scales with your development process website. [Online]. Available: <https://dojotoolkit.org>
- [10] (2016) Github: Web based Git repository hosting service website. [Online]. Available: <https://github.com>
- [11] C. Sotelo. (2014) Evolution of the single page application part 2. [Online]. Available: <http://paislee.io/evolution-of-the-single-page-application-2-of-2>
- [12] (2016) Google Chrome team: Evolution of the web website. [Online]. Available: <http://www.evolutionoftheweb.com>
- [13] (2016) Ionic: Create mobile apps with the web technologies you love website. [Online]. Available: <http://ionicframework.com>
- [14] (2016) Kendo UI: The Most Complete UI Framework to Speed Up Your HTML/JS Development website. [Online]. Available: <http://www.telerik.com/kendo-ui>
- [15] (2016) Angular: HTML enhanced for web apps website. [Online]. Available: <https://angularjs.org>
- [16] (2016) Sencha: Rapidly design, develop, and manage cross-platform web application website. [Online]. Available: <https://www.sencha.com>
- [17] A. Avram. (2014) "Facebook: MVC Does Not Scale, Use Flux Instead." [Online]. Available: <http://www.infoq.com/news/2014/05/facebook-mvc-flux>
- [18] *ECMAScript 2015 Language Specification*, Standard ECMA-262, 2015
- [19] (2016) React: a JavaScript library for building user interfaces website. [Online]. Available: <https://facebook.github.io/react>
- [20] (2016) Angular2: One framework. Mobile and desktop website. [Online]. Available: <https://angular.io>
- [21] J. Haberman (2014) "React Demystified." [Online]. Available: <http://blog.reverberate.org/2014/02/react-demystified.html>
- [22] ReactJS Docs (2016) "Working With the Browser." [Online]. Available: <http://facebook.github.io/react/docs/working-with-the-browser.html>
- [23] J. Whelpley, P. Stapleton (2015) "Angular 2 Server Rendering." [Online]. Available: <https://angularu.com/ng/session/2015sf/angular-2-server-rendering>
- [24] M. Thomas (2015) "Server-Side Rendering with React + React-Router." [Online]. Available: <https://ifelse.io/2015/08/27/server-side-rendering-with-react-and-react-router>
- [25] D. Nutels (2016) "Server-Side Rendering With React, Node And Express." [Online]. Available: <https://www.smashingmagazine.com/2016/03/server-side-rendering-react-node-express/>
- [26] M. Ubl (2015) "Tradeoffs in server side and client side rendering." [Online]. Available: <https://medium.com/google-developers/tradeoffs-in-server-side-and-client-side-rendering>
- [27] (2016) Redux: a predictable state container for JavaScript apps website. [Online]. Available: <http://redux.js.org/index.html>
- [28] M. Fogus, *Functional JavaScript*, O'Reilly Media, 2013
- [29] (2016) TypeScript: a typed superset of JavaScript that compiles to plain JavaScript website. [Online]. Available: <http://www.typescriptlang.org>
- [30] B. Ali Syed, *TypeScript Deep Dive*, [Online]. Available: <https://github.com/basarat/typescript-book>, 2015
- [31] (2016) TodoMVC: Helping you select a MV* framework website. [Online]. Available: <http://todomvc.com>
- [32] J. Breck-McKye (2014) "The State of JavaScript in 2015." [Online]. Available: <http://www.breck-mckye.com/blog/2014/12/the-state-of-javascript-in-2015>
- [33] A. Pike (2015) "A JS framework on every table." [Online]. Available: <https://www.allenpike.com/2015/javascript-framework-fatigue>
- [34] C. Elsea (2015) "The React Curve." [Online]. Available: <https://medium.com/@Connorelsea/the-react-curve>
- [35] G. Podjarny (2016) "Eliminating Known Vulnerabilities With Snyk." [Online]. Available: <https://www.smashingmagazine.com/2016/01/eliminating-known-security-vulnerabilities-with-snyk>
- [36] C. Cimpanu (2016) "Node.js Package Manager Vulnerable to Malicious Worm Packages." [Online]. Available: <http://news.softpedia.com/news/node-js-package-manager-vulnerable-to-malicious-worm-packages-502216.shtml>
- [37] P. Traeg (2015) "JavaScript Meets the Internet of Things." [Online]. Available: <http://www.universalmind.com/blog/technology/javascript-meets-the-internet-of-things>
- [38] (2016) Electron: Build cross platform desktop apps with web technologies website. [Online]. Available: <http://electron.atom.io>
- [39] S. Jones, C. Voskoglou, M. Vakulenko, V. Measom, A. Constantinou, M. Kapetanakis, "Cross-platform developer tools 2012," Survey by Vision Mobile, 2012
- [40] IBM Corporation, "Native, web or hybrid mobile-app development," Whitepaper by IBM corporation, 2012
- [41] Sourcebits Technologies, "Native, HTML5, Hybrid?" Whitepaper by Sourcebits, Inc., 2012
- [42] (2016) Phonegap: Build amazing mobile apps powered by open web tech website. [Online]. Available: <http://phonegap.com>
- [43] TJ VanToll (2015) "The State of Hybrid Mobile Development." [Online]. Available: <http://developer.telerik.com/featured/the-state-of-hybrid-mobile-development>
- [44] (2016) Telerik Platform: Develop Mobile Apps with Powerful Backend Services website. [Online]. Available: <http://www.telerik.com/platform/backend-services>
- [45] T. Roes (2013) "Old WebView vs. Chromium backed WebView Benchmark." [Online]. Available: <https://www.timroes.de/2013/11/23/old-webview-vs-chromium-webview>
- [46] M. Firtman (2013) "Android 4.4 KitKat, the browser and the Chrome WebView." [Online]. Available: <http://www.mobilexweb.com/blog/android-4-4-kitkat-browser-chrome-webview>
- [47] TJ VanToll (2014) "Why iOS 8's WKWebView is a Big Deal for Hybrid Development." [Online]. Available: <http://developer.telerik.com/featured/why-ios-8s-wkwebview-is-a-big-deal-for-hybrid-development>

- [48] (2016) Google's Android: developer dashboard website. [Online]. Available: <http://developer.android.com/about/dashboards/index.html>
- [49] (2016) Apple's iOS: app-store support website. [Online]. Available: <https://developer.apple.com/support/app-store>
- [50] M. Willocx, J. Vossaert, V. Naessens, "A Quantitative Assessment of Performance in Mobile App Development Tools," in *proceedings of the IEEE Int. Conf. on Mobile Services*, New York, 2015
- [51] Kiosk Pro (2016) "WKWebView: Known Issues." [Online]. Available: <http://docs.kioskproapp.com/article/840-wkwebview-supported-features-known-issues>
- [52] Cordova (2016) "Cordova WKWebView Engine with http server (localhost) support." [Online]. Available: <https://github.com/apache/cordova-plugins/tree/master/wkwebview-engine-localhost>
- [53] K. Erickson (2016) "Phonegap, be wary of plugins." [Online]. Available: <http://www.agingcoder.com/phonegap/2016/03/20/phonegap-be-wary-of-plugins>
- [54] (2016) Crosswalk Project: build world class hybrid apps website. [Online]. Available: <https://crosswalk-project.org>
- [55] (2016) Telerik: Enter Verified Plugins Marketplace website. [Online]. Available: <https://plugins.telerik.com>
- [56] S. Witalec (2014) "Building Cordova Apps with the Verified Plugins Marketplace." [Online]. Available: <http://modernweb.com/2014/07/21/building-cordova-apps-verified-plugins-marketplace>
- [57] B. Satrom (2013) "How to build a hybrid app that performs like native - yes, it can be done." [Online]. Available: <http://venturebeat.com/2013/11/07/how-to-build-a-hybrid-app-that-performs-like-native-yes-it-can-be-done/>
- [58] M. Johnston (2015) "60 FPS on the mobile web." [Online]. Available: <http://engineering.flipboard.com/2015/02/mobile-web>
- [59] (2016) SamsaraJS: a library for animating layout website. [Online]. Available: <http://samsarajs.org>
- [60] J. Morony (2015) "Why Do HTML5 Mobile Apps Have a Bad Reputation?" [Online]. Available: <http://www.joshmorony.com/why-do-html5-mobile-apps-have-a-bad-reputation>
- [61] Apache Cordova Docs (2016) "Embedding WebViews." [Online]. Available: <https://cordova.apache.org/docs/en/latest/guide/hybrid/webviews/index.html>
- [62] H. Schinsky (2015) "Choosing a Mobile Strategy: Part 2." [Online]. Available: <http://phonegap.com/blog/2015/08/03/mobile-choices-part2>
- [63] Apple Inc. (2015) "Apple Developer Program License Agreement." [Online]. Available: <https://developer.apple.com/terms>
- [64] (2016) Basecamp website. [Online]. Available: <https://basecamp.com>
- [65] (2016) Supersonic: Build beautiful data-driven apps with real native performance website. [Online]. Available: <http://www.appgyver.io>
- [66] (2016) TriggerIO: The simplest way to build amazing mobile apps website. [Online]. Available: <https://trigger.io>
- [67] I. Anand, D. Wasmer, "Native vs Web vs Hybrid," Kinvey and Moovweb ebook, 2013
- [68] K. Ormandy (2015) "Your favourite app isn't native." [Online]. Available: <http://kennethormandy.com/journal/your-favourite-app-isnt-native>
- [69] R. Pot (2016) "Titanium - An introduction to Hyperloop by Hans Knoechel." [Online]. Available: <https://medium.com/all-titanium/titanium-an-introduction-to-hyperloop-by-hans-knoechel>
- [70] (2016) React Native: a framework for building native apps using React website. [Online]. Available: <https://facebook.github.io/react-native>
- [71] (2016) Nativescript: Build truly native mobile apps using JavaScript website. [Online]. Available: <https://www.nativescript.org>
- [72] (2016) TabrisJS: Native mobile apps in JavaScript. Simple website. [Online]. Available: <https://tabrisjs.com>
- [73] (2016) Smartface: Create High Quality Native Apps, Easily without any need for Mac OS website. [Online]. Available: <https://www.smartface.io>
- [74] B. Green (2015) "Building Mobile Apps with Angular 2 and NativeScript." [Online]. Available: <http://angularjs.blogspot.be/2015/12/building-mobile-apps-with-angular-2-and.html>
- [75] K. Whinnery (2011) "Best Practices for Cross-Platform Mobile Development." [Online]. Available: <http://www.slideshare.net/appcelerator/kevin-whinnery-best-practices-for-crossplatform-mobile-development/4>
- [76] N. Walker (2016) "Code Reuse in Angular 2 Native Mobile Apps with NativeScript." [Online]. Available: <http://angularjs.blogspot.be/2016/03/code-reuse-in-angular-2-native-mobile.html>