# Cross-Platform Development

Suitability of Current Mobile Application Frameworks

Jan Christoph, Daniel Rösch, Thomas Schuster
Hochschule Pforzheim
Tiefenbronner Straße 65, Germany
{jan.christoph | daniel.roesch | thomas.schuster}@hs-pforzheim.de

***Abstract* — Mobile device adoption has increased dramatically within the last decade. In addition to smartphones, wearables and various sensors are among the most utilized devices. At the same time, the multiplicity of devices increases, the number of platforms to consider when developing applications increases as well. It is therefore desirable to be able to generically develop applications and deploy them to all relevant target platforms. A typical approach is given by frameworks, which generate platform specific code. In this article, we examine the suitability of these frameworks. Central questions are access to native system functions, sensors of devices and support for upcoming platform developments. To evaluate the frameworks, we defined a reference application and implemented tests for different mobile devices and platforms. A final framework comparison reveals opportunities and limitations. This, in turn, serves as a foundation for future work on improvements of promising approaches.**

***Keywords-cross-platform; app development; Web engineering; component-based software architectures.***

## I. INTRODUCTION

Mobile devices have become an important platform for today's software applications. Especially, the utilization of smartphones increased rapidly within the last couple of years [1], [2]. Since smartphones are often utilized to consume or orchestrate services, this process includes a vast range of applications; they also connect to other domains such as the Internet of Things (IoT) and utilize smart cloud-based services.

The introduction of smartphones rapidly increased the need and development of mobile software. The development of mobile software applications is a special case of software engineering. Mobile applications are often also referred to as apps, which implies that the application is intended to be used on a smartphone or wearable device [3]. Thus, development must cope with specific aspects such as: short application lifecycles, limited device capabilities, mobility of users and devices, availability of network infrastructure as well as security and privacy issues [4]. While developers are enacted to create and distribute applications in a large scale, they also have to deal with these inherent limitations of mobile devices (i.e. battery life or small displays). Furthermore, it is necessary to address different operating systems (especially for smartphones, and, to a limited extent, for feature phones as well). Since the market for smartphones has consolidated recently, some operating systems (i.e. Windows Phone, BlackberryOS and other OS hold a market share of 0.2%) vanished again. Still, to address the smartphone market, applications for both, Android (market share: 85%) and iOS (market share: 14.7%) need to be provided. In addition, Android is split into different versions, manufacturers and various system customizations. Currently, the most widely used Android version is Nougat (7.0 and 7.1 with 28.5%), while little use is made of the newest Version Oreo (8.0 and 8.1; with 1.1%) [5].

To reach as many users as possible, all major platforms and versions need to be supported [4], [6]. This introduces the need to either develop platform specific or platform agnostic applications. Platform specific implementations (native apps) require as many application implementations as platforms are intended to be addressed. Therefore, this approach generates correspondingly high development expenditures without additional added value. On the other hand, with a more generic approach, a single application or some core components could serve as the basis for multiple platforms. Besides reduced developments efforts, a generic approach also strengthens reuse of code and components.

Currently, generic approaches can be further subdivided into Web and hybrid applications (see Fig. 1). Web applications can be used virtually under any platform, as a Web browser is preinstalled on almost all devices. The most salient advantage is application portability, which basically comes at no cost. Web apps are typically optimized by means of Hyper Text Markup Language (HTML5), Cascading Style Sheet (CSS) and JavaScript [7]. Numerous frameworks (such as Angular, Bootstrap, React or Vue) provide additional functionality on top of Web standard technologies and help to speed up development of Web apps. Major disadvantages of Web applications are that they do not possess platform specific look and feel and often are restricted in functionality – especially access to system functions and device sensors. Furthermore, they must be interpreted and suffer performance losses compared to native applications [8].

Hybrid applications are built on frameworks such as Apache Cordova or Adobe PhoneGap. Often they rely on Web technologies also, and enact access to native device functions and sensors [4]. Hybrid apps utilize a specialized browser to present the user interface (UI). This results in a presentation layer which is identical or very near to widgets used in native apps. While hybrid apps overcome some issues of Web apps (such as access to system functions and sensors), they still experience a loss of performance compared to native applications. However, it is notable that performance of hybrid apps has improved a lot with latest developments [4], [7]. Comparing the short development lifecycles of devices and operating systems on the one hand to that of hybrid app frameworks on the other, it is noticeable that the latest developments are implemented with delays by the frameworks. As a result, access to new functionalities can be gained earlier when development is based on native apps.

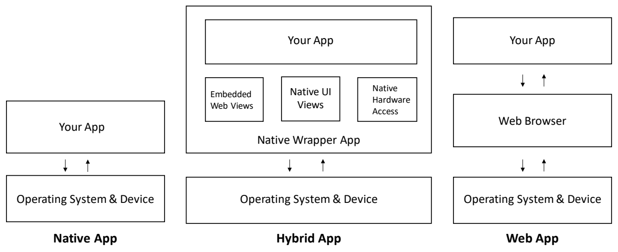

Figure 1. Mobile App Technology Stack

Issues of supported functionality, performance and the generic question of maintenance of cross-platform applications lead us to the evaluation of multiple cross-platform frameworks. The remainder of this article is structured as follows: Section 2 provides an overview of current mobile app development. In Section 3, a reference architecture is presented and three framework-based implementations of this architecture are discussed. The reference implementations are being evaluated in Section 4. Finally, Section 5 provides our conclusion and outlook on future work.

## II. Related Work

### A. *Cross-Platform Development*

As stated above, there are several approaches for cross-platform development. This type of development is subject to typical challenges of ubiquitous computing. In addition, further challenges are typical to cross-platform development [4], [9], the most important being associated with:

1. UI
2. Limited Resources
3. Device Management
4. Application Maintenance

The design of UI is associated with questions of simplicity and intuitiveness. For mobile cross-platform development, this is extended by design guidelines defined by the different operating systems. It is further restricted because of different device capabilities (e.g., screen sizes and resolution) [10]. Limited resources is a typical issue in mobile software engineering; for cross-platform development the application size and resource consumption (especially power and memory management) is a typical issue [4], [11]. Since cross-platform development addresses a vast variety of devices, their management in terms of appropriate usage of hardware and sensors (i.e. cpu, memory, bluetooth, or camera) becomes another typical challenge. Furthermore, different operating systems must be handled as well. Finally the application has to be maintained by following short lifecycles of devices, operating systems and frameworks [4], [10].

A lot of different methods that address cross-platform development can be observed in science and industry. Some are based on model-driven software engineering [12]. The advantage of model-driven methods is, that developers and users which are less familiar with specific programming paradigms are enabled to efficiently implement applications. As Object Management Group (OMG) standard, the Interaction Flow Modeling Language (IFML) offers model-based and platform-independent development of applications for different types of devices. Following the Model-Driven Architecture (MDA) it is based on a meta-model and it is built upon Web Modeling Language (WebML). A Web-based and an eclipse-based modeling environment is provided for IFML. Furthermore, extensions for Apache Cordova and PhoneGap are provided [12]. An open challenge is to keep the extensions up-to-date. Other solutions, such as WebView, utilize native code and combine it with Web technologies. Native components are used as containers to render Web pages that contain application logic and presentation layer definitions. Native components serve to access device-specific functions (i.e. push notifications or sensor data). Although WebView is a native application, it can internally use Web technologies without switching to a standard browser. WebView also supports CSS and JavaScript for custom interface development [8]. However, WebView does have two main drawbacks: 1) custom styling is necessary to gain a native look and 2) its performance is below average [13]. In summary, we observe three general approaches to cross-platform development:

1. Native Application
2. Transformation- or generator-based Application
3. Interpreted Application (Parser-based)

With native development, an application is developed for each specific device (and operating system). Benefits include the native look and feel, the ability to use all platform-specific features and a comparatively high performance of the app. The most prevalent disadvantage is high efforts for development and maintenance. The latter is a result of redundancy in code and support because each platform has to be served by a separate application [8], [9].

The use of generators employs a meta-implementation which is then transformed to specific platforms (e.g. as used in Cordova or Ionic). Similarly, model-driven development approaches (such as IFML) may use transformations to produce platform specific code. An advantage is that the application logic is platform agnostic [12]. Applications which are interpreted rely on some kind of parser. The parser interprets application code during runtime in order to create platform specific instructions. Fabrik19 utilizes an interpreted approach in its Mobility Suite (MOS) framework.

### B. *Cross-Platform Frameworks*

As discussed above, there are a lot of cross-platform frameworks like IFML, Cordova, Corona Software Development Kit (SDK), Appcelerator Titanium, TheAppBuilder, PhoneGap, Native Script, SenchaTouch, Framework7, Apache Weex, Flutter, Jasonette or Manifold – also see [6]. All of them utilize one or a combination of the three methods to create platform specific applications. In our comparison, we strive to evaluate the most frequently used and most progressively developed frameworks (see Fig. 2).

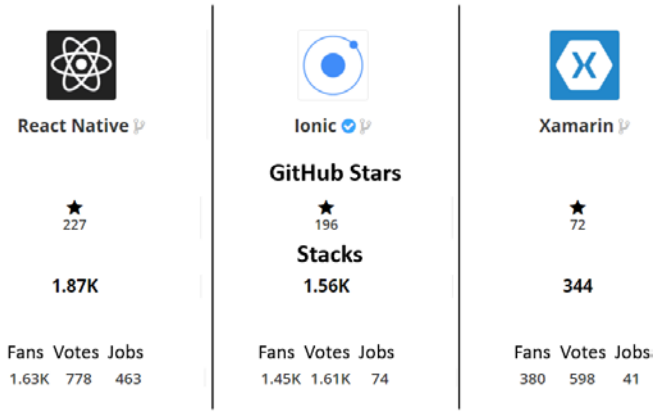

Figure 2. Ionic vs. React vs. Xamarin [14]

**Ionic** offers a generator-based approach [15]. The framework is free to use and available as open source. Additionally, several services are available via pay on demand. The generator utilizes a Web application as input. Thus, development of cross-platform applications is based on Web technologies (JavaScript/TypeScript, HTML5 and CSS; see Fig. 3). Ionic also relies on Angular [15] in order to foster component based development and reuse of templates. Ionic officially supports Android, iOS and UWP [16]. Since Ionic is based on Web applications that are generated into platform specific applications through Apache Cordova, these source applications may also be executed in any Web browser. Native operating system functions and access to sensors is only available after generation of platform specific code. The utilization device specific functionalities often also rely on plugins that have to be declared as dependency [17].

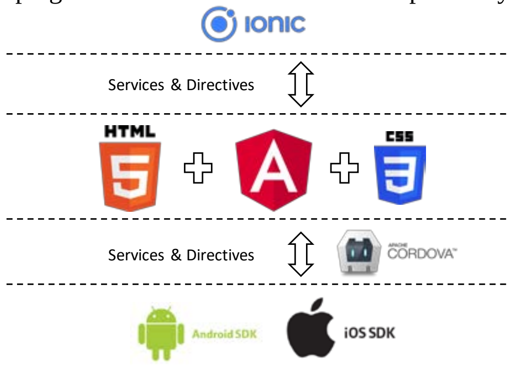

Figure 3. Ionic Architecture

**Xamarin** is another framework to develop cross-platform apps for Android, iOS and Universal Windows Platform (UWP) [18]. Other platforms such as Linux are not supported and MacOS support was recently added with the launch of Xamarin.Mac.

Xamarin is based on .Net and utilizes C# as programming language. Xamarin is divided into two major parts: 1) Xamarin platform and 2) Xamarin.Forms. The Xamarin platform (Xamarin.Android, Xamarin.iOS) provides APIs to share code for application logic between all platforms. The UI is written individually for each platform. Xamarin.Forms allows to create additional platform-independent UI, which are mapped into native UI in a second step. The development environment is based on Visual Studio (or Xamarin Studio for macOS) [18].

**React Native** is a parser based open-source framework for building cross-platform applications [19]. It is based on React. Both frameworks are being developed by Facebook. React Native currently supports Android and iOS. However, with a little more effort, it is also possible to deploy to UWP. Since React is built on JavaScript, this holds true for React Native as well. React Native invokes Objective-C APIs to render to iOS components and Java Application Programming Interface (APIs) to render to Android components. This means that no code generation is utilized in React Native. Facebook promises that the performance of apps would be almost as good as that of native applications. Components for React Native may either be built as functional components or class components [19].

## III. Reference Architecture and Implementation

This paper follows the constructivist paradigm of design science [20]. Thus, insights will be retrieved by creating and evaluating artifacts in the form of models, reference architectures and, in our case, specific implementation variants and efforts spent on their creation. Contrary to empirical research, the goal is not necessarily to evaluate the validity of research results with respect to their truth, but to the usefulness and feasibility of the different approaches in order to solve a common problem – here, to deploy with ease to different mobile platforms. Following this line of thought, requirements will be imposed by the definition of a reference application architecture. The reference architecture is derived using common hypotheses, practitioner interviews and literature review. The reference architecture serves as requirements model for the implementation of different alternatives and tests in a real environment.

Thus, the reference application architecture is defined to compare most utilized frameworks against each other and to identify strengths and weaknesses. To enact a comprehensive comparison [6], [21], the application should access native system functionalities and provide a platform specific UI. In short, the frameworks should generate applications, which are close to native applications. Thus, we also evaluated against platform specific UI guidelines for Android and iOS [22]. We defined the following functional reference criteria:

1. Layout: Grid
2. Layout: Tab
3. Operating System Function: Access current time
4. Sensor Function: Access current position (GPS)

5. Sensor Function: Access the phone camera

In addition to functional criteria, it is also important to measure quality aspects, such as development efforts and application performance. Therefore, we analyzed two different types of layouts mentioned in the list above, that are often used in today's apps – Mockups are depicted in Fig. 4.

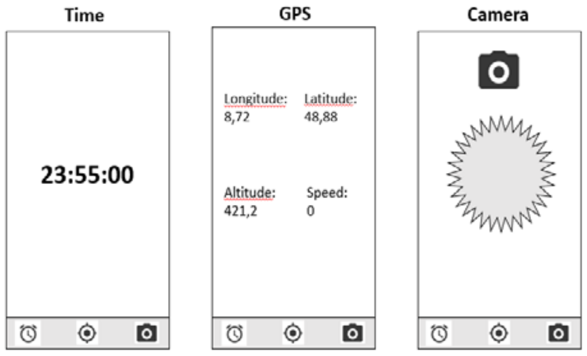

Figure 4. Wireframes

## A. Ionic

**Layout – Grid**: Ionic provides a typical Grid-View with the `<ion-grid>` component [16]. Furthermore, styling of the GridView can be set individually. **Layout – Tab**: Using Tabs in Ionic is easy as wall, it may be just derived by use of the starter template (which provides this from scratch). Precise instructions may also be found in the documentation [16].

**Access system time**: This is derived by simple and built-in JavaScript function calls (e.g., date().getHours() is used to get the current hour). **Access current position (GPS)**: To determine the position, the Cordova plug-in Geolocation has to be installed via npm. Then, it can be integrated in the project [16]. As shown Listing 1, the position can be retrieved, if the necessary sensors are available and permissions are given. **Access to the camera**: To use the camera, the Cordova plugin Camera is required and has to be integrated into the project [16].

Listing 1

```
getThePosition(){
   this.geolocation.getCurrentPosition().
      then((resp) =>{

         this.longitude = resp.coords.longitude;
         this.latitude = resp.coords. latitude;
         this.altitude = resp.coords. altitude;
         this.speed = resp.coords. speed;
   }).catch((error) => {
      console.log("Error getting location", error);
   });
}
```

**Debugging & testing**: Ionic offers several methods to debug and test apps. If the application is not utilizing sensor information, a clean Web test can be driven (by `ionic serve`). Web tests may be carried out as known for Web applications in general – such as debugging by means of the browser's developer console (`F12` shortcut) or employing Web driver test scripts. If sensor information is utilized the application has to be deployed to a platform specific device or an emulator. With Ionic this can be done by calling `ionic cordova build android|ios` to build the app and `ionic cordova emulate android|ios` to execute the app on an emulator. If a test device is being utilized instead of emulation (by calling `ionic cordova run android|ios`) the application may again be tested in a browser, e.g. using Google Chrome (`chrome://inspect/#devices` has to be called and the specific device has to be selected). In order to automate unit testing typical tooling as known for other JavaScript-based frameworks can be used. To test the reference implementation, we could simply employ the well-known frameworks Karma and Jasmin.

## B. Xamarin

**Layout – Grid**: In Xamarin the layout differs, depending on the chosen platform. For Android `GridView` and for iOS `uicollectionview` has to be used [18]. **Layout – Tab**: In Xamarin tabs have to be set up manually. There is no standard template available to support this layout. Typically, a tabbed page will be used to reference other content integrated as tabs.

**Access system time**: To retrieve the system time, a `ViewModel` is created, and a `DateTime` attribute tracks the current time. For updates a `PropertyChanged` event is fired. The reference is made possible by the data binding. **Access current position (GPS)**: The current position is determined by the plugin `Xam.Plugin.Geolocator` [18] (installed via NuGet). Adjustments are needed to support Android. In addition, necessary privileges for querying the position must be granted. After configuration, the logic can be implemented. Attributes for longitude and latitude have to be mapped to determine the location (see Listing 2). **Access to the camera**: Camera access is realized with the plugin Xam.Plugin.Media [18]. It has to configured by means of xml. In important step is the definition of a resources folder to determine where to store captured pictures and videos. The camera itself can be called asynchronously (`getTakePhotoAsyncCommand`).

Listing 2

```
public async System.Threading.Tasks.Task
   getLocationAsync()
   {
     var locator = CrossGeolocator.Current;
     locator.DesiredAccuracy = 50;
       if (locator.IsGeolocationAvailable &&
             locator.IsGeolocationEnabled) {
               var position = await
                 locator.GetPositionAsync();
               this.Longitude="Longitude" +
                 position.Longitude.ToString();
               this.Latitude="Latitude" +
                 position.Latitude.ToString();
       }
   }
```

**Debugging & testing**: Xamarin enables unit testing and debugging with Visual Studio. For Xamarin, Visual Studio basically offers the same mechanisms as known for any other component which is developed within Visual Studio (such as break points and live debugging). Visual also offers support for asynchronous testing and mock object creation, e.g. if the Model View Viewmodel (MVVM) pattern is applied and view models invoke service operations asynchronously. Visual Studio also provides a well sophisticated profiler, which provides monitoring of memory utilization and object allocation. Finally, Xamarin also offers also a test cloud for UI-Tests – where automated testing for native and hybrid applications is done by employing the App-Center.

### C. React Native

**Layout – Grid**: React Native does not provide a grid layout immediately. To resemble a grid-layout within the reference implementation, a ScrollView component was used and individual views had been adapted by means of CSS. Alternatively, third-party grid components could be utilized as well to resemble a grid layout. React Native Easy Grid and React Native Layout Grid are just two examples of these components, which may be installed via npm. **Layout – Tab**: React-Native Expo IDE can create a starter app which directly operates with tabs. Manual creation is not as easy as in Ionic but efforts are still considerably low.

**Access system time**: Is achieved by simple JavaScript calls. `this.state` [19] is needed for the databinding and `new Date().getHours()` retrieves the current hour. **Access current position (GPS)**: The determination of the current position is already integrated in the React Native API [19]. The position is retrieved by calling `navigator.geolocation.getCurrentPosition`, further details can be seen in Listing 3. **Access to the camera**: Camera and access rights have to be configured and `hasCameraPermission` has to be set to zero. The `componentWillMount` method the permissions are checked and we the status is updated. The asynchronous method `takePicture` is utilized to check if the camera is available and if it was possible to take a picture.

Listing 3

```
Navigator.geolocation.getCurrentPosition(
   (position) => {
      this.setState({
         latitude: position.coords.latitude,
         longitude: position.coords.longitude,
         error: null,
      });
},
(error) => this.setState({ error: error.message }),
{ enableHighAccuracy: true, timeout: 20000,
maximumAge: 1000},
);
```

**Debugging & testing**: React Native similarly offers multiple ways to debug and test apps. Debugging mode can be activated from a developer menu. This can be called by keyboard shortcuts or, if running on a test device, by shaking the smartphone. To debug the JavaScript code in Chrome, a remote debugging session can by created when select `Debug JS Remotely` is selected from the developer menu. This will open `http://localhost:8081/debugger-ui` in a new browser tab. Other debugger implementations may be used as well and a recommendation then would be to use the standalone version of React developer tools. These can be installed via `npm install -g react-devtools` and may be called via `react-devtools`. To set up unit testing for React Native it is recommended to utilize Jest and execute tests via node. For integration testing, several different options exist. Integration testing always relies on platform specific environments; thus those have to be set up first.

## IV. Evalution

The evaluation and comparison of different frameworks is based on our test app. We selected evaluation criteria based on the evaluation framework developed by Heitkötter et al. [23]. It covers different evaluation criteria, especially for infrastructure (including the lifecycle as well as the functionality and usability of the app) and app development (including testing, debugging and developing the app). We also extended or removed some criteria (e.g. scalability) to focus the following app properties:

1. Supported platforms
2. Supported development environment
3. Access to platform-specific functions
4. Application look and feel
5. Application portability
6. Simplicity of development
7. Application performance

It is important which platforms (Android, iOS, UWP, etc.) and to which extent these are supported by each framework. The next criterion discusses all possible development platforms and environments (Windows, MacOS and Linux). With the help of our test app we intend to analyze if platform-specific functions are available. Also, an evaluation of the UI is conducted to measure platform specific look and feel. Moreover, we want to unveil if the source code is reusable and if it can be integrated into other frameworks (portability). Also, the development efforts play a major role and will be evaluated within criterion 6. In order to assess and evaluate efforts and feasibility of the frameworks, we asked five experiences developers to implement our test application according the reference architecture. The following evaluation is also based on their feedback. Finally, we conducted an assessment of the application's performance. Therefore, the test app is used to measure: start time, used memory and execution speed of internal functionalities such as GPS polling. For this purpose, three test devices (Honor 9, Sony XZ1, Samsung Galaxy S7) were used. 100 test runs were conducted for each device to stabilize results.

### A. Ionic

Configuring a system for Ionic and creating a first app only takes a few minutes. Regarding ramp up, the majority of our developers found that Ionic is the easiest framework to start with. It has to be mentioned, that it is necessary to ensure that all dependencies (to plug-ins) are installed according their declared version. This can be error prone, especially when Ionic is updated. In case of multiple app development projects, conflicts may also arise between dependencies of different projects. Hence, previously deployed Ionic projects should be removed from the test device to prevent side effects during testing. As a prerequisite to start and develop Ionic applications only knowledge in the typical Web development stack (HTML, JavaScript and CSS) is required. TypeScript as an extension of JavaScript and thus is easy to learn if JavaScript is already known. TypeScript provides additional benefits compared to JavaScript (especially type safety) – some extension have been adopted into ECMAScript-6 (such as classes, inheritance or generics) [24].

In addition, Ionic reuses Angular, which makes it easier to keep the code clean, separate concerns and speed up development of the application itself. The project structure in Ionic is logically well structured according to Web component architecture. Since Ionic relies on Web technologies, the user is free to choose the development environment [16]. The use of Cordova is another advantage, especially because it enables access to system specific functionality and device sensors. Furthermore, Cordova improves re-use of application components, since a single code base can be utilized for all platforms. However, since Ionic is based on Web-technologies and packaged into native wrapper applications, the performance is behind native applications. Former evaluations also indicated that the performance is behind Xamarin and React Native, especially for larger applications [20].

### *B. Xamarin*

Xamarin projects can be set up in Visual Studio. With the use of C#, Xamarin is the best choice for developers, who also work conventionally with C#. Another advantage is the native UI [18]. Users will not recognize any difference to native applications. Xamarin offers to share a single code base between platforms, to develop application logic. Platform specific extensions may be integrated with a subproject feature of Xamarin. As for all cross-platform frameworks, problems may arise with third-party plugins (installed via NuGet). We recognized several issues with outdated plug-ins. In general, our experience has shown that new device and operating features of mobile devices had been adopted very fast by Xamarin. Hence, in most cases it framework based services can be used instead of third-party plugins. Regarding testing and debugging applications, the developers stated, that Xamarin would be the most convenient framework to use. This may be the case because of extended possibilities instantly provided by Visual Studio.

### *C. React Native*

React Native is easy to set up as well. React Native is built upon React, and is also based on JavaScript. Applications developed in React Native interpreted directly and the design appears near to native. Interesting features include a well-designed live debugging. With Expo, React Native offers an open source toolchain to simplify deployment on test devices. Although this may result in some benefits, we observed that the apps that are generated by the Expo are structured differently than those set up by the console. Additionally, these apps have different access to native functions. Another disadvantage compared to the other frameworks is interface development. React utilizes a lot of specific HTML-Tags which we recognized as somewhat difficult to use and configure. This makes it more difficult to get started than with other frameworks, even if experience in Web technologies is preexistent.

### *D. Comparative Evaluation*

To evaluate all frameworks comparatively and in an objective manner, we implemented a test application according the reference architecture (as defined in Section 3). In a second step, we measured the criteria defined at the beginning of Section 4, to reason about benefits and limitations of all frameworks.

**Supported platforms**: Ionic officially supports Android, iOS and since 2016 also UWP development, although the documentation is still very limited here. Xamarin offers full support for Android, iOS and UWP. Limited support is provided for MacOS. React Native supports iOS, Android and with a little extra effort also UWP applications. **Supported development environment**: Ionic applications can be developed on Windows, macOS and Linux. The development platform for Xamarin is Visual Studio for Windows and Xamarin Studio for macOS. React Native supports Windows, macOS and Linux. **Access to platform-specific functions**: Ionic provides access to iOS, Android, Microsoft and browser-based features. Platform-specific functions can be used via various Cordova plugins. With Xamarin, all platform-specific functions can be used in a similar fashion. However, Xamarin offers different possibilities to access platform specific functions. The fastest possibility is to install corresponding NuGet packages. A second option would be the definition of interfaces with platform or devices specific implementations and expose this shared code via the dependency service. Then there is also the possibility to use native libraries, for example written in pure Java for Android, via binding. While React-Native is JavaScript-based and many native functions are not supported, it is possible to include native SDKs and libraries. However, this requires specific code for Android (in Java) and for iOS (in Swift) which results in higher development efforts. In addition, these features are currently not mature enough.

**Application look and feel**: Ionic offers its own widgets for the UI. Navigation elements (e.g. back button) are provided in platform-specific style, so the differences to native apps are small. As already described in Section 3, the use of a GridView in Ionic is very simple. Xamarin creates completely native UI, thus the interface is familiar to the user. Xamarin also supports styling with themes and the interface is not different to native apps. Xamarin Android also supports material design. React Native uses specialized widgets. Setting up a GridView it is not as easy as in Ionic or Xamarin, CSS has to be used to achieve this layout. In general, Ionic and React Native ignore style guidelines of platforms partially and some widgets break them explicitly. For example, tabs in Android are at the top of the screen in native apps, while this is not the case in apps developed with

Ionic or React Native. Xamarin, in contrast, uses tabs as expected.

**Application portability**: Since Ionic represents a hybrid approach, portability of the source code is given and further supported through Cordova. Since Ionic modules are well-structured and based on Web technologies, they can be transferred to other Web frameworks. However, as many other frameworks, Ionic uses specific HTML tags that may not be supported in other frameworks, thus there is limited transferability of this module part. Since Xamarin separates application logic and UI related code, it offers the best portability and reuse of the logic. Furthermore, Visual Studio offers a portability analyzer to transfer the UI related parts as well. Of course, it has to be said that this is restricted to .Net and mono frameworks. The UI (defined by eXtended Application Markup Language (XAML)), could in principle be transformed into HTML or similar languages, which, however, requires further manual efforts in a second step. Similarly, React Native offers portability to different platforms. React-Native code is relatively easy to transfer to other frameworks that use JavaScript, HTML, and CSS. Comparable to Ionic, specialized HTML tags have to be XAML handled manually. However, since as React Native Logic, UI and CSS are typically implemented in a single file, this tends to be tedious.

**Simplicity of development**: Through a lot of documentation (tutorials, community discussion, API documentation, quick start and programming templates) a quick and efficient start in development of Ionic Apps is possible. Because of the short development lifecycle, confusion may occur through different version documents and some outdated plugins. Occasionally, the framework reveals unexpected behavior (some builds end up with broken apps, while a rebuild without code change is successful). We intend to examine this further. Currently we believe that this is related to generator issues. In principle, the development with Xamarin is fast as well, since the framework also possesses a very good documentation (tutorials, sample projects and a very precise API documentation). The programming language underneath (C#) also is very sophisticated and in our opinion much better then JavaScript. In terms of simplicity, Visual Studio or NuGet may pose a certain barrier for developers not used to it in the beginning. The entry into the development with React Native is comparable to Ionic. The use of the framework-specific UI elements is different from the other frameworks but does not impose an obstacle. The ability to see and debug all changes in real-time eases troubleshooting. A larger issue is related to external libraries and modules. Since many of these modules and libraries are not officially supported, regular maintenance and support is not guaranteed. In addition, we observed that the installation of node modules consumes much more time compared to Ionic.

**Application performance**: (100 test runs of the test app, with three test devices) The required start time of the Ionic test app is between 2s and 2.5s, while Xamarin requires 3 to 3.5s, and React Native 4 to 5s. The size of the Ionic app is 10 MB, while Xamarin requires 24 MB and React Native requires 11 MB. The time the Ionic app takes to retrieve the current location (with high signal strength of GPS) is approximately 0.2s, while Xamarin needs 3.2s and React Native 0.5s. Based on upon the evaluation criteria presented and measured above, the overall results are summarized in Table 1 (“++”=very good , “+”=good, “0”=neutral, “-“=poor, “--"=very poor).

TABLE 1: EVALUATION OF CROSS-PLATFORM FRAMEWORKS

| **Evaluation Criteria** | **Ionic** | **Xamarin** | **React Native** |
|---|---|---|---|
| Supported platforms | + | ++ | + |
| Supported development platforms | ++ | 0 | ++ |
| Access to platform-specific functions | + | ++ | 0 |
| Application Look & Feel | + | ++ | + |
| Application Portability | 0 | 0 | 0 |
| Simplicity of development | ++ | + | 0 |
| Application performance | ++ | + | 0 |

## V. CONCLUSION AND OUTLOOK

Development of cross-platform applications is supported by different approaches: native development, transformation- or generator-based and interpreted. With the exception of native development, all approaches share in common that some layer of abstraction is introduced to encapsulate platform specific details. The latter enables platform-independent development in combination with platform-dependent deployment. In our case study, we examined three frameworks (Ionic, Xamarin and React-Native) in depth. In general, the performance of Ionic was reported to be the slowest of these three frameworks, with our test applications However, we measured Ionic to be the fastest of all three. Compared to other studies [4], we also observed performance enhancements of all evaluated cross-platform frameworks. This holds true for response and processing times as well as sensor access. The latter may be a result of framework improvements within the latest versions as well as improvements of the mobile device platforms. From a user perspective, there are no performance issues recognizable compared to native apps. All evaluated frameworks provide full access to system functionalities and sensors. Although new releases of operating systems provide new functionalities and sometimes also completely different API, the rate and speed of adoption in cross platform frameworks is quite high [25]. All cross-platform frameworks allow generic development for different operating systems, although there still exist limitations. As we discovered in the reference-app, sometimes apps are not completely portable, and still require platform specific adjustments. Furthermore, the re-use of components between different mobile applications is not yet supported. However, Ionic promises to allow the use of other components written in React, Vue and Angular in its next version [16].

Furthermore, long-term support is essential to reach a broad range of users and to enact support of up to date applications. With recent releases of the examined frameworks it could be observed, that backward compatibility to APIs of former releases was not given (e.g., between Angular1 to Angular2). This may be repeated with new releases in the future. Thus, for mobile software engineering, it is questionable if standardized component development (e.g. Web component architecture) will be adopted and foster framework consolidation or if the proliferation of new programming languages and techniques will continue to split the market. Similar reasoning applies to backward compatibility of API. Consequently, further research questions regarding API issues arise. A major question will be, if it is possible to transfer code from current framework applications to new releases and preserve its functionality (even if the API changes). Therefore, as a next step, we intend to define a model-driven approach that tackles this issue. In this context, we plan to compare parser-based methods to transformation-based cross-platform approaches and derive API mappings.

## REFERENCES

[1] Statista, “Internet of Things, Forecast, Number of networked devices worldwide by 2020,” Statista. [Online]. Available: https://de.statista.com/statistik/daten/studie/537093/umfrage/anzahl-der-vernetzten-geraete-im-internet-der-dinge-iot-weltweit/. [Accessed: 07-Mar-2018].

[2] Statista, “Number of smartphone users worldwide 2014-2020.” [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 15-May-2018].

[3] L. Corral, A. Janes, and T. Remencius, “Potential advantages and disadvantages of multiplatform development frameworks–a vision on mobile environments,” Procedia Comput. Sci., vol. 10, pp. 1202–1207, 2012.

[4] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, “Taxonomy of Cross-Platform Mobile Applications Development Approaches,” Ain Shams Eng. J., vol. 8, no. 2, pp. 163–190, Jun. 2017.

[5] Statista, “Android - Proportion of versions, Statistics,” Statista, Feb-2018. [Online]. Available: https://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/. [Accessed: 07-Mar-2018].

[6] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, “Survey, comparison and evaluation of cross platform mobile application development tools,” in Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, 2013, pp. 323–328.

[7] M. Lachgar and A. Abdali, “Decision Framework for Mobile Development Methods,” Int. J. Adv. Comput. Sci. Appl. IJACSA, vol. 8, no. 2, 2017.

[8] V. Ahti, S. Hyrynsalmi, and O. Nevalainen, “An Evaluation Framework for Cross-Platform Mobile App Development Tools: A case analysis of Adobe PhoneGap framework,” 2016.

[9] P. Smutný, “Mobile development tools and cross-platform solutions,” in Proceedings of the 2012 13th International Carpathian Control Conference, ICCC 2012, 2012.

[10] J. Perchat, M. Desertot, and S. Lecomte, “Component based framework to create mobile cross-platform applications,” Procedia Comput. Sci., vol. 19, pp. 1004–1011, 2013.

[11] M. E. Joorabchi, M. Ali, and A. Mesbah, “Detecting inconsistencies in multi-platform mobile apps,” in Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, 2015, pp. 450–460.

[12] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, “Model-Driven Development Based on OMG’s IFML with WebRatio Web and Mobile Platform,” in Engineering the Web in the Big Data Era, 2015, pp. 605–608.

[13] A. Holzinger, P. Treitler, and W. Slany, “Making Apps Useable on Multiple Different Mobile Platforms: On Interoperability for Business Application Development on Smartphones,” in Multidisciplinary Research and Practice for Information Systems, 2012, pp. 176–189.

[14] “Ionic vs React Native vs Xamarin 2018 Comparison | StackShare.” [Online]. Available: https://stackshare.io/stackups/ionic-vs-react-native-vs-xamarin. [Accessed: 07-Mar-2018].

[15] M. Ramos, M. T. Valente, R. Terra, and G. Santos, “AngularJS in the wild: A survey with 460 developers,” in Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools, 2016, pp. 9–16.

[16] Ionic, “Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular,” Ionic Framework. [Online]. Available: https://ionicframework.com/. [Accessed: 07-Mar-2018].

[17] Apache Cordova, “Architectural overview of Cordova platform.” [Online]. Available: https://cordova.apache.org/docs/en/7.x/guide/overview/index.html. [Accessed: 09-Mar-2018].

[18] “Developer Center - Xamarin.” [Online]. Available: https://developer.xamarin.com/. [Accessed: 07-Mar-2018].

[19] Facebook, “React Native · A framework for building native apps using React.” [Online]. Available: https://facebook.github.io/react-native/index.html. [Accessed: 09-Mar-2018].

[20] A. Hevner and S. Chatterjee, “Design science research in information systems,” in Design research in information systems, Springer, 2010, pp. 9–22.

[21] M. E. Joorabchi, A. Mesbah, and P. Kruchten, “Real challenges in mobile app development,” in Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on, 2013, pp. 15–24.

[22] M. Willocx, J. Vossaert, and V. Naessens, “A Quantitative Assessment of Performance in Mobile App Development Tools,” in 2015 IEEE International Conference on Mobile Services, 2015, pp. 454–461.

[23] C. Rieger and T. A. Majchrzak, “Weighted Evaluation Framework for Cross-Platform App Development Approaches,” 2016, pp. 18–39.

[24] Microsoft, “TypeScript is a superset of JavaScript that compiles to clean JavaScript output,” 07-Jun-2018. [Online]. Available: https://github.com/Microsoft/TypeScript. [Accessed: 07-Mar-2018].

[25] M. Martinez and S. Lecomte, “Towards the quality improvement of cross-platform mobile applications,” ArXiv170106767 Cs, Jan. 2017.