Cross-Platform Development Frameworks

Overview of contemporary technologies and methods for cross-platform application development.

Keith Vassallo (Masters Student)
Department of Computer Information Systems
Faculty of Information and Communication Technology
University of Malta
Msida, Malta

keith.vassallo.11@um.edu.mt

Dr. Lalit Garg
Department of Computer Information Systems
Faculty of Information and Communication Technology
University of Malta
Msida, Malta

lalit.garg@um.edu.mt

***Abstract*— This is paper provides an overview of the technologies currently (2016) available and in development which allow the development of cross-platform applications. Both server-side and client-side applications are considered, as well as applications for web, desktop and mobile devices such as smartphones and tablets. A web-based approach is recommended for the development of truly cross-platform applications across devices and operating system.**

Topics discussed include the contemporary background within which cross-platform technologies are developing, full-stack web development using a MEAN stack, cross-platform mobile development methodologies and web-based desktop application development.

Keywords—cross-platform; full-stack; MEAN; mobile applications; web-based desktop applications.

I. INTRODUCTION

A. Convergance

The development of software in contemporary times takes many different forms, including but not limited to desktop, web (client and server) and mobile applications as well as the development of embedded devices. With the advancement in web technologies in the past decade, nowadays "both desktop and mobile software systems are usually built to leverage resources available on the World Wide Web" [1]. Moreover, it is clear that many devices that were previously the realm of embedded device programmers, such as mobile phones, televisions, car entertainment systems and electronic watches, are now leveraging the capabilities of operating systems initially designed for smartphones. Indeed, both the Android and iOS[1] operating systems are today available on a wide variety of devices. With an emerging convergence between embedded devices and general purpose computers, as well as the already established convergence between desktop and web computing, it is time to start looking at which technologies are available to allow developers to build applications that work across a broad spectrum of devices and operating systems.

B. Structure of this paper

In this paper, literature available about different contemporary technologies available for cross-platform development is reviewed. Starting with web technologies, popular development environments and frameworks available for client-side and server-side development are mentioned, with a focus on frameworks allowing the development of full-stack web applications, i.e. from both the client and server side. Next, this paper provides an overview of the native and cross-platform technologies available for mobile application development, and provides a comparison between the two. Finally, these are merged into a discussion providing recommendations for developers who wish to make their applications as widely-available as possible.

II. FULL-STACK WEB DEVELOPMENT

A. Traditional Web Development

Traditionally, web applications were developed using a LAMP [2] architecture. This provides all of the components needed for the development of the back-end of a web application; Linux as the server operating system, Apache as the web server, MySQL as the DBMS and PHP as the server-side language. Several alternatives are available, such as Ruby on Rails, Java EE, Python, ASP.NET MVC and many more. However, such technologies only cover the back-end, thus for front-end development, a different set of skills is required, using HTML, CSS, JavaScript and newer technologies such as HTML5 Canvas, LocalStorage and WebGL. Hence, a full-stack web developer would need to be proficient in a server-side language, database management system, HTML, CSS and JavaScript as well as other rapidly emerging and evolving client-side technologies.

B. Node.js

Node.js (or simply Node) was developed in 2009 by Ryan Dahl, as a single-threaded, non-blocking, server-side JavaScript environment, written in C and C++ [6]. With the development of Node, the idea was popularized to use the same language for

[1] As tvOS on Apple TV and WatchOS on Apple Watch

both back and front-end development, effectively implementing 'end-to-end' development. "The current environment of web applications demands performance and scalability" [3]. This is especially true with AJAX/Web2.0 applications. The traditional web-sever has been implemented using a single-thread per connection approach, and follows a request (from client) and response (from server) model. Modern web application servers need to leverage multiple-threads per connection, and support events. Node.js "achieves [this] both through server-side JavaScript and event-driven I/O." [3].

At the core of the paradigm driving Node.js is the non-blocking asynchronous model. The resources available on the server-side are typically files and databases. Considering the traditional web-server as a FIFO queue, and the resources as shared and concurrently accessed, one can see that a traditional web-server is blocking. It serializes requests into a queue and does not allow faster processes (such as the smaller web requests in an AJAX application) from completing prior to larger requests – it is strictly a first-in, first-out queue. On concurrent systems, blocking algorithms suffer from performance degradation as resources are locked by the first process that acquires a resource – therefore, if that process is delayed for any reason, all other waiting processes will also be delayed [4]. In contrast, non-blocking algorithms "guarantee that if there are one or more active processes trying to perform operations on a shared data structure, an operations [sic] will complete within finite number of time steps" [4].

Using such approaches, Node has achieved very good performance results. In a study conducted in [3], and corroborated in several other studies, Node outperforms both Apache and EventMachine[2] in tests consisting of a large number of small requests to shared resources (see Figure 1). However, more than performance gain, Node has managed to achieve mind-share. In the 2016 developers' survey by Stack Overflow [5], both Node and JavaScript on the server-side (i.e. via Node) feature as the most used and loved technologies by developers; "JavaScript is so pervasive that it's in all top 3-tech combinations used by Back-End Developers. This suggests a lot of these Back-End Developers are probably Full-Stack Developers in disguise. Our internal stats suggest about 60% of professional developers work full-stack" [5].

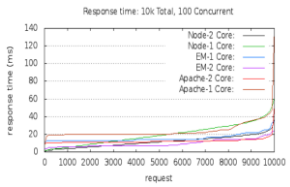

Fig. 1. Comparison of Apache, EventMachine & Node response time [3]

Node uses JavaScript on the server-side. By virtue of the fact that JavaScript is the only standards-compliant language supported by major web browsers, Node effectively makes JavaScript a full-stack language. This development means that developers can now stick to one language when developing both client-side and server-side logic for web applications. However, it leaves some gaps. Whereas JavaScript is functional and event-driven, traditional DBMSs used for hosting web-application content are relational. Moreover, as Node applications tend to be Single-Page Applications (SPAs), every page or resource accessed by the client will have the same URL. This makes link sharing and search-engine optimization difficult. Hence, with Node, developers must still use other paradigms – at least the relational paradigm to interact with databases, and a traditional web server to handle the re-writing of client URLs into Node application function calls.

C. MEAN

The development and rise of Node led to developers looking elsewhere for their web-application data storage requirements. Now that JavaScript was being used for the server-side and client-side code, could it be extended to the persistence layer? The answer came in the form of document-databases, particularly MongoDB as it exposes a JavaScript API to the data [7]. MongoDB stores data as documents in a JSON format known as BSON. It uses functions to perform CRUD operations on this data. Through JavaScript, callback functions can be used to chain functions together for more advanced operations [8]. Besides a continuity of paradigm, research by several teams including Radulescu et. al. [8] show that MongoDB outperforms traditional RDBMSs (Oracle in the study mentioned) by a large factor (see Table 1). This is especially true for a large number of small requests, as would be generated by a modern web-application.

TABLE I. INSERT TIMES (MS) ORACLE VS MONGODB [8]

No. of records	Oracle Database	MongoDB
10	31	800
100	47	4
1000	1563	40
10000	8750	681
100000	83287	4350
1000000	882078	57671

The missing piece of the puzzle is routing. This is needed in modern web applications since they tend to be single-page applications. This means that rather than each user screen being on a different HTML page and requiring a request/response cycle with the web server, the single page uses JavaScript via xmlHttpRequest to request data in the background, and then update the browser's Document Object Model (DOM). However, this means that a web application has only one URL, making link sharing and search engine optimization difficult. A URL router can address these issues. A router maps URLs typed into the browser to function calls in the web application. Several routers exist, but in maintaining the JavaScript theme, routers working at the client side via JavaScript have emerged. These include the popular Express[3] router.

The stack is rounded off with a data-binding layer. New programming paradigms such as promises are used to make it easy for content shown in the browser DOM to be updated by changes in data state, and vice versa. Libraries such as

[2] An event-driven I/O library for the Ruby programming language.

AngularJS[4], ReactJS[5] and KnockoutJS[6] have emerged which use event streams to achieve this automatically. This allows the web application to be updated in real-time, based on changes to state or data; "most libraries construct a dependency graph behind the scenes. Whenever an expression changes, the dependent expressions are recalculated and their values updated." [9]

Together, MongoDB, Express Routing, AngularJS and Node.js, provide the MEAN stack – an alternative web application development stack intended for rich web applications (also known as 'thick clients'). Although the stack is called MEAN, and there is indeed a framework that is itself called MEAN[7], the term is also used to describe other frameworks that use some or all of these technologies, but with different implementing frameworks.

III. Cross-Platform Mobile Development

A. Native Applications

The mobile smartphone market is fragmented between major operating systems such as Android, iOS, Windows Phone amongst others [10]. This makes the development of cross-platform mobile applications a challenge, since each operating system exposes different APIs, uses different programming languages and supports different features based on the model and device being used by the customer. In response to this, several cross-platform mobile application development suites have emerged, to simplify and quicken the pace of development.

Development using the tools made for the platform (such as Android Studio for Android and Xcode for iOS) is referred to as *native* development. "From the end user perspective, native apps provide the richest user experience. Source code is efficient, with fast performance, consistent look and feel and full access to the underlying platform hardware and data." [11]. However, developing native applications requires a different set of skills for each platform.

B. Web Apps

Cross-platform mobile development tools aim to address this issue, whilst still maintaining as much of the user experience and functionality of the native mobile device as possible.

One approach is the creation of web apps. Rather than building a native application, a web application optimized for mobile devices is constructed. This can then be accessed via a mobile browser, without the need for installation. The main problem with such applications is that they have limited access to the device's hardware and data, and they cannot be used (in many cases) without an active Internet connection. They also tend to be slower than native applications, and do not maintain the consistent look and feel of the mobile operating system [11].

Web apps use HTML5 technologies, which include a set of APIs for added functionality and reduced power consumption. These include Web Storage, Indexed Database API, File API, Web SQL Database, Offline Web and Geolocation API [12].

C. Hybrid Applications

Hybrid applications attempt to combine the advantages of a native application with the speed of development and cross-platform nature of a web application. Hybrid applications embed a web application into a thin device-native container, used to display the web content. Using specialized APIs, the web application can then make calls to the container, which will query and relay device-native system calls to the web app. Some of the most popular implementations include PhoneGap[8] and Cordova[9] [11]. Besides allowing more access to device data and functionality, hybrid applications download the web application's data to the local device, meaning the application can be used even without an active Internet connection.

D. Interpreted Applications

With interpreted applications the user interface that the user interacts with is generated automatically and uses native components to provide a consistent look and feel. However, the application logic can be built using a wide variety of languages, depending on the skillset of developers. Supported languages include, but are not limited to, Java, Ruby and XML.

Whilst such applications provide a native look and feel, the developer is reliant on the interpreted framework for support of any new user-interface elements introduced by the mobile operating system. For example, when Google introduced Material UI[10] in Android, users of interpreted development environments had to wait for their environment to support it before they could use it. Popular interpreted development environments include Appcelerator Titanium Mobile[11] [11].

E. Generated Applications

Generated applications are compiled for a specific device, just like their native counterparts. Hence, a different app is created for each targeted device. Popular examples of application generators include Applause[12] for CRUD applications, and the Unity game engine for game development.

"In theory it is also possible to exploit the produced native code, in order to meet specific needs… However, in practice utilization of the generated native code is difficult because of its automated structure." [11].

F. Comparing Approaches

A comparison of the different approaches towards mobile application development is beyond the scope of this paper, and has already been covered by many including [11, 13, 14, 15].

[4] https://angularjs.org/
[5] https://facebook.github.io/react/
[6] http://knockoutjs.com/
[7] http://mean.io/
[8] http://phonegap.com/
[9] https://cordova.apache.org/
[10] https://www.google.com/design/spec/material-design/introduction.html
[11] http://www.appcelerator.com/
[12] http://www.applause.com/

The purpose of this paper, however, is to find technologies whereby an application can be built across platforms, be it mobile, web or desktop. By virtue of the fact that the web only supports HTML, CSS and JavaScript, it is trivial to deduce that the technology which would entail the less re-writing of code to develop mobile applications from an existing source is the web.

IV. Cross-Platform Desktop Development

A. Cross-Platform GUI Toolkits

Compared to mobile application development and rich web application development, desktop application development is a mature space. For decades, technologies have been available to develop desktop applications, and these include cross-platform applications. The storage and business logic components of an application can be written in almost any language of the developer's choice, as compilers for most programming languages are available for the major operating systems (Windows, Mac OS, Linux). The stumbling block has traditionally been the user interface, where the three operating systems use different technologies, look and feel.

One solution to this problem has been to develop cross-platform GUI libraries. Libraries such as QT[13], GTK[14] and wxWidgets[15] provide APIs for a wide variety of programming languages. Developers can choose a language of their choice and then integrate with the toolkit to create the interface they need. Although successive versions of such libraries keep improving, it is still easy to tell that applications built with such libraries are not native, even if they differ only slightly from the native application toolkit [16].

Another solution is to provide a GUI toolkit as part of a programming language, which is the approach used by languages such as Java, which uses AWT/Swing and more recently JavaFX to address this issue. Here too, however, the applications will approach, but not perfectly imitate, the native look and feel of applications on the target platform. Moreover, languages such as Java require a runtime environment to be installed on the client machine.

B. A Web-Based Approach

Following the success of hybrid applications on mobile devices, which use web technologies to develop mobile applications, the same approach is now being used to develop desktop applications. Electron[16], NW.js[17] (formerly known as node-webkit), Chromium Embedded Framework[18] and AppJS[19] are all frameworks that allow developers to use web technologies to develop desktop applications using HTML5, CSS and JavaScript. Although several frameworks and technologies have been developed over the years for cross-platform application development (including Java and the .NET framework), web/desktop frameworks do not require a different skillset; the existing skills used for web application development can be used for desktop development.

There are a number of challenges to this approach. In essence, web applications and desktop applications have traditionally been based on different paradigms, thus creating an impedance mismatch that "reflects the fact that the World Wide Web was originally designed to be a document distribution environment – not a software platform" [1]. Moreover, these desktop frameworks are, in essence, containers around web rendering frameworks, normally either Chrome (via Chromium) or WebKit. There are subtle differences in the way these renderers display content, which has to be accounted for if, for example, the rendering engine used by the cross-platform mobile tool is different.

This being said, web applications are becoming richer and the features traditionally associated with desktop development (multi-threading, graphics, instant response, etc.) are now becoming cross-domain features, easing the transition from desktop to web development, but also, as is this case, vice versa.

V. A Unified Approach

A. The Case for Web

From the research carried out in this paper, it is suggested that applications based on web technologies are a viable way forward for true cross-platform applications, regardless of device and operating system. By building applications for the web first, and then integrating them into mobile devices as hybrid applications and as desktop applications via available frameworks, one can preserve the vast majority of not just the application persistence and logic, but also user interface.

Modern web applications tend to be *responsive*. This means that the user interface of such applications adapts to the size of the user's device. Hence, the developer can focus on first creating a responsive web application using modern technologies and then, with considerably little effort, such applications can then be ported to the mobile space using a hybrid framework. Similarly, the same base code can then be ported to the desktop. There are a number of different technologies that can be used to achieve this. In the next section, a selection of technologies will be recommended based on the aforementioned research.

B. Recommended Technologies

It is recommended that development begins by creating a web application based on Node. This application will be based on a MEAN stack, i.e. using a JavaScript document-oriented database, a client-side router, a JavaScript data-binding framework and the Node server.

One efficient approach is to use a framework that can automatically generate hybrid mobile applications from web application code. Such frameworks include Meteor[20]. Meteor uses a MEAN stack approach to the development of mobile

[13] http://www.qt.io/
[14] http://www.gtk.org/
[15] https://www.wxwidgets.org/
[16] http://electron.atom.io/
[17] http://nwjs.io/
[18] https://bitbucket.org/chromiumembedded/cef
[19] http://appjs.com/
[20] https://www.meteor.com/

applications. However, it adds on top of Node by providing additional features such as templating engine support and in-browser document database. This means that the client maintains a restricted local copy of the database in cache, to speed up and reduce requests that would normally go directly to the server. Meteor supports a number of data-binding frameworks including AngularJS, ReactJS and Blaze. Additional features for rapid development include hot-code push (the ability so see changes in the app without refreshing) and quick deployment.

Of added interest, mobile platforms can be added to a Meteor project, which will cause Meteor to automatically package an application using Cordova, to build a hybrid iOS or Android application. Once bundled for deployment, Meteor applications can be run in a Node container, without the need for the Meteor framework to be installed. Hence, with one development sprint, the developer would have created a web-application and a mobile-friendly web app, as well as a cross-platform mobile application supporting Android and iOS.

The web content created could then be transferred to one of the desktop web app frameworks such as Electron. With relatively little effort, the application can hence become a desktop application.

When any updates are required to the application code, it is only the web content that needs to be updated, and then be also copied to the new version of the desktop application. Despite being written using web technologies, native features will still be available, as Meteor exposes the Cordova APIs to the web app, and desktop applications such as Electron can expose natively functionality.

C. *Known limitations*

As mentioned in other sections of this paper, there are limitations to the web technology approach. Firstly, the application will not have access to all of the functionality provided by the native platform, but rather will be limited to the subset of functionality offered by the framework in use.

Also, despite very close approximation, it is still the case that web applications are not a perfect facsimile of their native counterparts, especially when the native operating system introduces new UI widgets.

Finally, although a web-application itself is cross-platform, the containers used for the mobile and desktop aspects of the applications are not. Hence, implementation differences may be present and need to be accounted for.

VI. Conclusion

It is suggested that web technologies are the future of development not just for rich web applications, but also for mobile and desktop applications. Developers should strive to support as many platforms as possible, to allow their application to reach a wider audience regardless of device and operating system.

References

[1] T. Mikkonen and A. Taivalsaari, "Apps vs. Open Web: The Battle of the Decade In Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development", in *2nd Workshop on Software Engineering for Mobile Application Development*, MSE'11, 2011, pp. 22-26.

[2] J. Lee and B. Ware, *Open source Web development with LAMP*. Boston: Addison-Wesley, 2003.

[3] R. McCune, "Node.js Paradigms and Benchmarks", Undergraduate, University of Notre Dame, 2011.

[4] M. Michael and M. Scott, "Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms", University of Rochester, Rochester, NY, USA, 1995.

[5] "Stack Overflow Developer Survey 2016 Results", *Stack Overflow*, 2016. [Online]. Available: http://stackoverflow.com/research/developer-survey-2016. [Accessed: 24- Mar- 2016].

[6] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs", *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, 2010.

[7] J. Dickey, *Write modern web apps with the MEAN stack*. San Francisco, CA: Peachpit Press, 2015.

[8] A. Boicea, F. Radulescu and L. Agapin, "MongoDB vs Oracle - database comparison".

[9] K. Kambona, E. Gonzalez Boix and W. De Meuter, "An Evaluation of Reactive Programming and Promises for Structuring Collaborative Web Applications", in *European Conference on Object-Oriented Programming*, Lancaster, UK, 2011, p. Article 3.

[10] StatCounter, "Top 8 Mobile Operating Systems from Feb 2015 to Feb 2016", GlobalStats, 2016.

[11] S. Xanthopoulos and S. Xinogalos, "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications", in *6th Balkan Conference in Informatics (BCI'13)*, Thessaloniki, Greece, 2013, pp. 213-220.

[12] "HTML5", *W3.org*, 2016. [Online]. Available: https://www.w3.org/TR/html5/. [Accessed: 25- Mar- 2016].

H. HeitkÖtter, S. Hanschke and T. Majchrzak, "Proceedings of 8th International Conference on Web Information Systems and Technologies", in *WEBIST '12*, Porto, Portugal, 2012, pp. 299-311.

[13] M. Palmieri, I. Singh and A. Cicchetti, "Proceedings of 16th International Conference on Intelligence in Next Generation Networks", in *ICIN '12*, Berlin, Germany, 2012, pp. 179-186.

[14] W. Jobe, "Native Apps Vs. Mobile Web Apps", *Int. J. Interact. Mob. Technol.*, vol. 7, no. 4, p. 27, 2013.

[16] M. Wojtczyk and A. Knoll, "Proceedings of The Third International Conference on Software Engineering Advances", in *ICSEA '08*, Sliema, Malta, 2008, pp. 224-229