Mobile App Development in HTML5

Spyros Xanthopoulos[1, a] and Stelios Xinogalos[2, b]

[1] *IT Center, Aristotle University of Thessaloniki, Thessaloniki, Greece*
[2]*Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece*
[a]*xant@auth.gr*
[b]*stelios@uom.edu.gr*

Abstract. Mobile apps have been around for only a few years, but have gained immense consumer attention. Developers creating native apps are faced with the daunting task of maintaining different sources for each target platform, while web and hybrid apps are faced with a series of technical and non technical issues. Native apps function smoothly in a standard way that the user is already familiar with and provide the richest user experience having full and direct access to the onboard hardware and software. On the other hand web apps offer cross-platform compatibility, saving development time, effort and money for developers. Moreover, HTML5 is expected to enhance user experience offered by web apps, which is one of their major shortcomings. The aim of this paper is to investigate various aspects of HTML5 in mobile app development.

Keywords: HTML5, JavaScript, mobile app development, web app, browser.
PACS: 68M11, 68N15, 68N19, 68U35

INTRODUCTION

Companies in their effort to extend their applications to the mobile audience have various solutions with advantages and drawbacks. On the one hand, native applications are platform specific and need dedicated software development kits and frameworks that are tied to the target platform. In case that multiple mobile platforms are targeted the apps must be developed separately creating and managing different versions for each platform spending time, effort and money. Here lies the attraction of developing apps written once and have it work on multiple devices no matter what technologies are in the hands of the end users.

In 2012 Facebook CEO Mark Zuckerberg admitted that betting too much on HTML5 as opposed to native was a big mistake that they made as a company because "HTML5 it just wasn't there". Although initially Facebook embraced HTML5 for its apps for iOS and Android, finally the company turned to native mobile experience on these two major mobile platforms. On the other hand Gartner recommends that by 2015 hybrid apps will be the majority of enterprise mobile apps and web technologies like HTML5 will become the most commonly used language for building mobile apps [4]. Zuckerberg's turnaround on HTML5 for mobile is one example among others, where a company lost time and effort focusing on the wrong direction. Considering cross-platform solutions for building mobile apps there are a number of things that need to be taken into account in order to assess the adequacy of each solution. For instance, each cross-platform tool has its own policy about the set of target-platform features that are supported and when new features that come out are going to be incorporated leading to a certain delay in upgrading the apps compared to native development. There are also other factors to consider when it might be suitable to use a cross-platform tool like technical limitations, app store restrictions and more [3].

In this paper we investigate technical and non technical considerations that can help to identify whether HTML5 can deliver what it promises in the domain of mobile apps. Because there is a plethora of different issues concerning mobile app development, we focused on the most common issues through literature research and other sources like developer communities, experienced programmers and personal views.

DEVELOPMENT APPROACHES

Building *native apps* today is the complete opposite of the cross-platform development. Native applications are typically downloaded and installed from the official platform's app store, and have a platform specific functionally and look and feel. At the other extreme, mobile web applications known as *web apps* are common web sites optimized for mobile devices. Specific JavaScript libraries are used to achieve a platform specific look and feel and functionality, while accessing and manipulating (with limitations) the underlying hardware, sensors and user data is based on standardized APIs imposed by consortiums like W3C. Web-enabled mobile apps will play a critical role in the mobile device area, while there is an ongoing battle between native apps and HTML5-based Open Web applications [3]. In the Web-based application development model developers use standard development tools and there is no need for compilation or linking.

There is one approach that tries to balance the trade-off between native and web technologies, the *hybrid apps* [5]. Developers building hybrid apps still develop their applications using standard web technologies in the same way like web apps with the additional capability of specific APIs provided by the development platform to access the underlying hardware and software components. In the general case, the applications act like a thin web browser that renders the source code providing also a bridge to the underlying native functionality. Gartner in a press release made in 2013 recommended that hybrid apps would be used in more than 50 percent of mobile apps by 2016 [4]. However, different customer needs (40% native, 40% hybrid, 20% web apps) and enterprise needs (10% native, 60% hybrid, 30% web apps) result in different development approaches.

In some cases though there are cross-platform development tools like Titanium where the user interface is native and the UI source code is *interpreted* at run time creating the necessary native user interface components. Finally, there is a last approach, where the necessary native source code is *generated* by a cross compiler for each target platform, but currently available solutions in this category are not mature enough [5] [6].

DISTRIBUTION CHANNELS

The typical distribution process of mobile apps to end users is through the platform's official app store, such as Google Play for Google and iTunes for Apple. This process is subjected to quality controls and developer program policies that all apps must comply. This is an extra value that guaranties that apps distributed through official channels are secure and qualitative. Developer program policies and app review guidelines include rules that apply to any content the app displays or links to, advertisements it shows or user-generated content it hosts [1]. Violation of these policies may cause app rejection, and additionally, as the policy rules may change they must be checked back regularly. App stores also constitute fully operational market frameworks where users can search, locate, inspect reviews, buy apps in a standard way and get notifications for upgrading onboard installed apps.

On the other hand there are other ways of distributing apps. Smartphones are equipped with a number of web browsers, so accessing a web app using the web through the local web browser is easy. Also, users can bookmark the URL for repeated use and the most up-to-date version of the app is always delivered, while in case of installed apps the users are constrained to reinstall the applications (the user must check e.g. if the access rights of the app have changed). In the case of Internet access the user must be aware of the dangers and security risks, although the browsers provide a number of protection mechanisms like cross-origin control. Although in the general case an Internet connection is required and there is a severe impact on performance in case of unavailable or slow websites, HTML5 can make the web app available in off line mode without the use of Internet or Wi-Fi network connection. Hybrid apps can be distributed through the platform's app store but in any case the app must comply with the app store's policy to be accepted. It is important also to notice that the upgrades of the installed native and hybrid apps cannot be forced. Developers are constricted to support not only the last deployed version but all past versions that are currently installed on user devices, leading to an extra maintenance effort overhead. Web apps, as opposed to the other types, do not need to be upgraded. Upgrades are instantly available to end users.

ACCESS TO HARDWARE SENSORS

W3C specifies several DOM Events [9] to provide access to the various sensors available on a hosting device. A DOM (Document Object Model) application takes advantage of interfaces, methods and other features in order to create dynamic and interactive content exposed to end users in a user agent while an *event* is an instantiation of one specific event type such as a mouse click. According to W3C all user agents (UA) implementing the Sensor API Specification must provide new DOM events for each sensor type. The corresponding event must be fired when there is a change in the sensor data and if the UA (that implements the interface) does not support the sensor type then the event will not be fired. If the DOM Sensor Event is supported then it is fired in accordance with Table 1.

TABLE 1. HTML5 Sensor Events and Related Data Formats.

Category	Sensor type	Data Type	Unit
Devicetemperature	Ambient temperature sensor	Double	Degree Celsius (ºC)
Devicepressure	Pressure sensor	Double	KiloPascal (kP)
Devicehumidity	Relative humidity sensor	Double	Percentage
Devicelight	Light sensor	Double	Lux
Devicenoise	Ambient noise sensor	Double	DbA
Deviceproximity	Proximity sensor.	Double	Centimetres (cm)

W3C provides also specifications [8] for obtaining information about the *physical orientation* and *movement* of the hosting device from sources like gyroscopes, compasses and accelerometers: the `deviceorientation` event supplies the physical orientation of the device; the `devicemotion` event supplies the accelerator and also the rotation rate; and the `compassneedscalibration` event helps identify if a compass used by the above events needs calibration.

RUN OFFLINE

One major advantage that native applications have always had over mobile web apps is the ability to run offline. With the advent of HTML5 Application Cache, Web Storage and Indexed Database API it is possible to create mobile web apps that run fully offline. Web apps can utilize the HTML5 offline application support in order to ensure the availability of the apps even when the device is offline or when the network connection is unavailable, for instance when the device is getting outside the ISP coverage area or by bad signal or other problems [9] [11] [12].

HTML5 Application Cache is achieved when the application provides a manifest file in which there is a listing of all JavaScript, styling or other type of files needed for the web app to work offline. The browser checks the manifest file listing and creates a local copy of all the files in order to be used offline. For example, a simple manifest file (contacts.appcache) can store locally the following three files: contacts.html, contacts.css and contacts.js. HTML5 Application Cache provides a means for offline functionality, but what about the user's data? Web Storage is designed for storing small scale data, while Indexed Database API is adequate for more structured data [12].

With *HTML5 Web Storage* the data is stored in name-value pairs reaching a storage limit of at least 5MB and can only be accessed by the page that stored the data. There are two objects available for storing data on the client: `localStorage` that stores data with no expiration date and `sessionStorage` that stores data for one session and the data are destroyed when the browser tab is closed. The following example shows the necessary JavaScript code for creating a `localStorage` name/value pair with the name = "languagesetting" and value = "english". The use of `localStorage` ensures that the data will be available after the browser is closed.

```
localStorage.languagesetting = "english";
document.getElementById("languagesetting").innerHTML = localStorage.languagesetting;
```

HTML5 Indexed Database specification comes as a continuation of the Web SQL Database, a specification inactivated in 2010 after reaching an impasse to proceed along a standardization path. The stored data consists of records holding simple values indexed using keys. A query language can be used for locating the records either by the key or by using the index. JavaScript can be used for creating a contacts database (for example) that holds contacts stored as records with "phone" and "name" attributes. Additionally, an index can be created on the "name" attribute of the records for quickly accessing contacts by name, while this index can be used also to implement a uniqueness constraint.

HTML5 client-side storage APIs provide a powerful technique bringing web apps into parity with native applications. Application data can be used immediately when the app is started reducing start-up latency, while web apps can operate even when the network signal is unreliable.

CONCLUSIONS

Developers creating native apps are faced with the daunting task of maintaining different sources for each target platform and supporting the last and all past deployed versions that are currently installed on user devices, while web and hybrid apps are faced with a series of technical and non technical issues. Native apps achieve maximum performance since they are compiled and the onboard installed code is executed directly by the underline operating system without any need of additional layers or web-downloaded source code. On the other hand, one of the major advantages of web apps is cross-platform compatibility with no need for installation, version upgrading or app store policy rules compliance.

The right choice of the development strategy depends primarily on the application requirements. In this paper we presented the techniques involved for implementing HTML5 web apps having native app advantages like fully offline capabilities and accessing device sensors via JavaScript. The challenges for making HTML5 a more powerful tool lies primarily on standardization. Mobile web browsers are fairly standardized and improvements are on-going leading to a common functionality at least for the greatest platforms (Android, iOS, RIM, Windows), while for other platforms there may be differences in the supported functionality requiring extra development across browsers. Sites have emerged performing compliance tests for open specifications like W3C and WebGL using arbitrary scoring systems ([2], [7]).

In alignment with this challenge further research should be carried out in order to implement an extended HTML5 compliance and performance testing framework for mobile web browsers with the aim of helping developers assess the adequacy of each development strategy. As stated in a previous work of ours attempting a comparative analysis of *native*, *web*, *hybrid*, *interpreted* and *generated apps* each development approach - including web apps in HTM5 - has specific advantages and disadvantages. So, the proposed framework must allow adjustable scoring systems based on user requirements depending on how important specific features are for a given application, like device's hardware and information access (orientation, geolocation etc), access time restrictions, data storage needs, processing load, look and feel, target platforms and others. The system must run specific compliance, qualitative and performance tests and finally report the best possible solution (and alternatives) and also the necessary development technologies (e.g. cross platforms tools).

REFERENCES

1. Google, Google Play Developer Program Policies, available at http://play.google.com/about/developer-content-policy.html.
2. Leenheer, N. (2013). The HTML5 test – how well does your browser support HTML? Available online at: http://html5test.com.
3. Mikkonen, T. and Taivalsaari, A. (2011). Apps vs. Open Web: The Battle of the Decade. In *Proc. of the 2nd Workshop on Software Engineering for Mobile Application Development*, 22-26.
4. Muelen, R., Rivera, J. (2013). Gartner recommends a hybrid approach for business-to-employee mobile apps, press release, Gartner site. Available at: http://www.gartner.com/newsroom/id/2429815.
5. Raj, R., and Tolety, S., (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *India Conference (INDICON)*, 2012 Annual IEEE, 625 – 629.
6. Xanthopoulos, S. and Xinogalos, S. (2013). A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In *Proc. of the 6th Balkan Conference in Informatics (BCI '13)*, 213-220.
7. Xinogalos S., Psannis E. K. and Sifaleras A. (2012). Recent advances delivered by HTML 5 in mobile cloud computing applications: a survey. In *Proc. of the 5th Balkan Conference in Informatics (BCI '12)*, 199-204.
8. W3C Working Group, (2014), DeviceOrientation Event Specification, available at: http://w3c.github.io/deviceorientation/spec-source-orientation.html.
9. W3C Working Group, (2014), HTML5: A vocabulary and associated APIs for HTML and XHTML, available at http://www.w3.org/html/wg/drafts/html/master/.
10. W3C Working Group, (2011), Sensor API Specification, available at https://dvcs.w3.org/hg/dap/raw-file/default/sensor-api/Overview.html.
11. W3C Working Group, (2010), Web SQL Database, available at http://www.w3.org/TR/webdatabase/.
12. W3C Working Group, (2013), Web Storage, available at http://www.w3.org/TR/webstorage/.