Mobile Web Apps: HTML5 or RemoteUI?

Dipl.-Ing. Daniel Thommes | Stuttgart Media University
thommes@hdm-stuttgart.de | December 2012

1 Abstract

In this document we compare HTML5[1] and RemoteUI[2] as solutions for realizing mobile web applications. We analyze the differences between the technologies based on their conformance with the important requirements. We ask, whether it is possible to realize web apps that behave very similar to native applications, but are based on web technologies.

2 Conformance to Requirements

The browser support for many features of the HTML5 specification is subject to frequent change. Where possible, we use references to the web sites http://caniuse.com and http://mobilehtml5.org that give detailed information about browser versions and their support for different technologies. Several extensions to HTML like the Device APIs[3] are working drafts and due to insufficient support not yet listed on the web pages above. As soon as information about browser support is available, we are willing to integrate it into this document.

The RemoteUI technology is under active development. Many features are planned (marked correspondingly) and will be supported in future versions of the client and the server framework. We will update this document as soon as the these features become available. RemoteUI is however no product but a single-person's research project. Purpose of this document is to give a prospect for future directions of the latter and to gather feedback.

This document has been created in December 2012. All given information is to the best of our recent knowledge. If you find mistakes, agree or disagree, please mail to thommes@hdm-stuttgart.de for active discussion.

3 User Experience Requirements

3.1 Provide a rich user interface

Native application frameworks for mobile devices provide user interfaces for various interaction models. A remote application should support the same or nearly the same widgets / views to express user interfaces.

Note: HTML allows additional UI elements by constructing them with HTML elements, stylesheets and additional JavaScript code. This however comes with the cost of larger initial download sizes (conflicts with 7.2) and decreased UI performance (conflicts with 3.2). In this table we refer to the native widgets that are part of the HTML5 specification:

1 http://www.whatwg.org/specs/web-apps/current-work/multipage/
2 http://www.remoteui.org
3 http://www.w3.org/2009/dap/

Approach	HTML5	RemoteUI
Text Input, Buttons, Drop-Down Lists, Radio Buttons, Labels	Full support	Full support
Sliders	Not fully supported by all browsers http://caniuse.com/#feat=input-range	Full support
Spinners	Not fully supported by all browsers http://caniuse.com/#feat=input-number	Full support
Progressbars	Not fully supported by all browsers http://caniuse.com/#feat=progressmeter	Full support
Date Chooser	Not fully supported by all browsers http://caniuse.com/#feat=input-datetime	Full support
Toggle Button	Not natively supported	Full support
Drag and Drop	No - nearly no support http://caniuse.com/#feat=dragndrop	Planned
2D vector drawing	Good support for Canvas and SVG http://caniuse.com/#feat=canvas http://caniuse.com/#search=svg	2D vector support on the client is planned. The RemoteUI JEE framework additionally provides delivery of rendered image content adapted for client resolution and density. This approach can be taken, if the 2D content needs no further modification after delivery.
3D vector drawing	No - nearly no support http://caniuse.com/#feat=webgl	Not planned
Layout	Layout in HTML is (typically) done with CSS. Several CSS features however are not available in mobile browsers. A detailed analysis would exceed the scope of this document. The following link shows critical CSS features like grid layout or viewport units first: http://caniuse.com/#agents=mobile&cats=CSS&sort=rscore	Layout for RemoteUI apps is done with special layout widgets defined in the RemoteUI markup. The concept closely follows the technique used in Android's layout system. Currently supported are LinearLayout, FrameLayout, ListViews and GridViews. Planned are GridLayout and RelativeLayout.

Approach	HTML5	RemoteUI
Styling	Styling like layout in HTML is done with CSS. The basic styling features like color, padding, margin etc. are typically supported by all browsers. Advanced styling features may be critical: http://caniuse.com/#agents=mobile&cats=CSS&sort=rscore	RemoteUI markup at the moment allows the discrete styling of all widgets. Support for abstract style definitions is planned and will closely follow Android's styles and themes[4] approach.
Full Screen	No - nearly no support http://caniuse.com/#feat=fullscreen	Planned
UI Animation	Good support http://caniuse.com/#feat=css-animation Drawback: Several browsers stop animation on user input (e.g. scrolling)	Planned

3.2 Maximize UI responsiveness

Approach	HTML5	RemoteUI
Let direct manipulation of UI objects (e.g. pressing a button) result in visual feedback within 0.1s [5].	With special preparation (see "Fast Buttons" below), this goal can be achieved for natively supported UI elements (see 3.1). UI elements that are rendered by JavaScript (styled with CSS and oftentimes images) on some browsers and platforms respond significantly slower (see "Fast runtime" below). This leeds to a loss of the native "feel" of a UI, which is easily recognized by all users.	RemoteUI renders the UI with the native UI-frameworks on each platform. These widgets usually react in adequate time.
Provide adequate visual feedback for actions taking longer than 1s.	Progress bars or spinners are required. They are not natively available on all browsers (see 3.1).	RemoteUI renders special UI widgets to provide visual feedback (e.g. text input with progress indicator). Progressbars and -spinners are built-in UI widgets.
Instant click behavior ("Fast Buttons")	Problematic because typical mobile browsers and WebViews after a	RemoteUI renders native buttons. These are "fast" by default.

4 https://developer.android.com/guide/topics/ui/themes.html

5 see: J.Nielsen,Usability engineering, MorganKaufmann, p.135, 1993.

Approach	HTML5	RemoteUI
	touch-up event wait 300ms for a second touch forming a double-click. Additional JavaScript code is necessary to change this behavior while still distinguishing between scroll and click events[6] (conflicts with 7.2). To support this, the touch event API is mandatory (see 3.4).	
Fast runtime using native code or JIT	**Android:** JIT since 2.2 (V8), However performance problems exist as experienced with Titanium Mobile apps and JQuery-Mobile web apps that use V8 for JavaScript execution. **iOS:** JIT only available in Safari, not in other Apps. JavaScript performance in Safari is very good.	**Android:** JIT for Dalvik byte code since Android 2.2. Image decoding is done in native code. Performance is very good. **iOS:** RemoteUI Client is natively implemented. Its performance is very good.
Parallel processing e.g. for deserialization, layout & image decoding	Partially possible with Web Workers, that are not available in all browsers: http://caniuse.com/#feat=webworkers Additionally debugging background tasks is not supported by several browsers (conflicts with 4.1)	All RemoteUI client implementations make extensive use of multi-threading to shift work into the background.
Support instant input validation	Validating user input on the client side was one of the first use cases for JavaScript. It is fast and can be used to mark input fields accordingly. It is however always required to repeat the validation on the server, because malicious clients can circumvent or just deactivate the JavaScript validation. If no adequate server framework is used, validation code must be implemented twice.	RemoteUI for JEE will support Bean Validation (JSR 303)[7] to provide validation on the server. User input is instantly transferred to the server. Special validation messages allow the display of server-side validation results on the client. Basic client side validation based on regular expressions and data types is planned. They will automatically be extracted from existing Bean Validation annotations.

6 https://developers.google.com/mobile/articles/fast_buttons

7 http://beanvalidation.org/1.0/spec/

3.3 Support content adaptation

Approach	HTML5	RemoteUI
Device dependent resources and layouts	Partially possible with CSS media queries that however only affect the styling. (Good support) http://caniuse.com/#feat=css-mediaqueries If also the DOM shall be device dependent, this can be achieved with special server frameworks or client side DOM manipulation code.	Content adaption will be integral part of the RemoteUI-JEE framework. This includes provision of different resources and layouts for different devices with the ability for transcoding images and using different serialization schemes for UI transfer.
Localization	HTML5 does not define APIs or rules for localization. Localization libraries are available for JavaScript (conflicts with 7.2) Localization of HTML content must be done on the server by applying an appropriate framework.	Localization will be integral part of the RemoteUI-JEE framework's content adaption components.

3.4 Support touch input

Approach	HTML5	RemoteUI
Support touch input	Good support (excluding Opera Mini): http://caniuse.com/#feat=touch	Full support

3.5 Support sensor input

Approach	HTML5	RemoteUI
GeoLocation	Good support http://caniuse.com/#feat=geolocation	Planned
Device Orientation	Not supported by Android 2.3 or lower: http://caniuse.com/#feat=deviceorientation	Full support
Accelerometer, Gyroscope, Magnetometer	Not supported by Android 2.3 or lower: http://mobilehtml5.org/	Planned
NFC	No - might be added as module to hybrid apps or as browser plugin.	Planned
Camera	Only supported by Opera Mobile: http://caniuse.com/#feat=stream	For Barcode scanning, other applications possible

3.6 Support service discovery mechanisms

Approach	HTML5	RemoteUI
ZeroConf for LAN services	While Safari desktop version has provides this feature, no mobile browser supports it. It might be added as module of a hybrid app or browser plugin (e.g. plugins on the desktop are available for Internet Explorer and Firefox).	ZeroConf discovery for LAN RemoteUI services is built into the client.
Barcode reader integration	No, however every barcode reader can parse and open URLs.	A barcode reader is integrated into the RemoteUI client. The rui:// protocol scheme additionally allows usage of arbitrary barcode readers delegating these URIs to the RemoteUI client.
Bluetooth discovery	To our knowledge, no mobile browser has built-in bluetooth functionality. It is however possible, to route network traffic over Bluetooth connections, if the operating system offers this functionality and the required UI to configure it.	RemoteUI can use the Bluetooth SPP profile as transport layer. The client has built-in Bluetooth discovery and connection functionality.

3.7 Support full keyboard input

Approach	HTML5	RemoteUI
Support all key events of a connected keyboard	Theoretically yes. Key codes partially differ from browser to browser. A library is recommended to realize browser-independent code (conflicts with 7.2). Early versions of (desktop) IE, Chrome and Opera did not bind certain keys (F5) or executed the default browser behavior in addition to possibly bound JavaScript functions. Current state of this feature in all mobile browsers is unknown.	Full support. Configurable shortcut for ending a RemoteUI session (default: long press the back button). **Android & iOS:** Home-Button cannot be overridden

Approach	HTML5	RemoteUI
	Android & iOS: Home-Button cannot be overridden	
Different virtual key-boards for different input types.	Good support Android browser only since 4.0 http://mobilehtml5.org/	Full support

3.8 Support full cursor / mouse input

Approach	HTML5	RemoteUI
Context menu (right mouse click)	Not natively supported on most browsers - can be added as own functionality (conflicts with 7.2) http://caniuse.com/#feat=menu	Context menus are supported as native widgets.

3.9 Support the majority of mobile devices

Actual market share estimates[8] show, that Android today is, and in future will be the most important platform for smart phones (50-60%). Followed by iOS (about 20%) and Windows Phone (5 – 19% in year 2016). Supporting these three platform results in support for 87 – 91% of the overall smart phone market. We also know, that as of today more than 60% of the Android devices are running Android 2.3 or lower[9], meaning that these devices have a market share of about 30%. Because updates are not available for many Android devices, the migration to newer versions of the Android OS is significantly slower than the migration to newer version of iOS. If the discussed technology shall be applied in consumer markets or in enterprise scenarios with BYOD[10] policy, the client software must support at least the most important platforms and versions.

Approach	HTML5	RemoteUI
Provide a client with good requirements conformance that runs on the major platforms and versions	**iOS:** Apple continuously updates its operating system even for older devices. E.g the iPhone 3GS is still supported by iOS 6. With a system update, the latest version of Safari is installed. Mobile Safari is for performance reasons (3.2) and tool support (4.1) a good choice for web app development. However the	**iOS:** The RemoteUI iOS client is still under active development. Probably RemoteUI will only support iOS 6 enabled devices. With this update came auto-layout features for supporting different screen sizes. **Android:** The current version of the Android RemoteUI client runs on devices with Android 2.2 and higher,

8 http://www.idc.com/getdoc.jsp?containerId=prUS23523812#.UL8WyR9eBOs (June 2012)

9 https://developer.android.com/about/dashboards/index.html (October 2012)

10 Bring You Own Device (https://en.wikipedia.org/wiki/Bring_your_own_device)

Approach	HTML5	RemoteUI
	browser cannot be extended with plugins or user scripts. Google Chrome as alternative is slower, not using JIT (3.2) and does not support plugins or extensions as well. **Android:** Android's built-in browser lacks several features in version 2.3 and lower. Google Chrome for Android is much better regarding features and tooling (4.1) , is however only available on Android devices running version 4.0 and above. The best browser seems to be Firefox for Android, that supports devices from Android 2.2 upwards. **Windows Phone:** As current information about Internet Explorer on Windows Phone is sparse and the market share is still low, we did not examine it in this document. Installing dedicated browser software always conflicts with 3.10.	supporting about 97% of all Android devices on the market. **Windows Phone:** A RemoteUI client for Windows Phone is not under development, yet.

3.10 Avoid installation of special software if possible

Approach	HTML5	RemoteUI
Use pre-installed software to realize remote access	The idea of HTML is to use an arbitrary browser to access every web app in the WWW. However in reality, there are limitations to this idea. As this document lays out, especially mobile browsers lack a variety of features, often-	Not possible, due to the RemoteUI client being required.

Approach	HTML5	RemoteUI
	times precluding there application for newer web apps. The solution is to prescribe certain browser software to be used with certain web apps. In effect, HTML can only fulfill this requirement, if the required browser is already pre-installed on the users's device (e.g. Safari on iOS). At least on Android this is a problem, because the built-in browser lacks many features and is updated only with OS updates. Because older devices oftentimes are excluded from updates, there will be a need for installation of prescribed browsers.	

3.11 Allow Session Continuation

Session continuation describes the process of continuing a running session on a different device. As an example, a user leaves her office and wants to continue a session started on her desktop PC on a mobile computer, tablet or phone.

Approach	HTML5	RemoteUI
Reconnect to a running session with another device	Many rich HTML5 applications use mechanisms like cookies, session storage, local storage or index DB that make session continuation difficult or impossible. The HTTP protocol itself is however stateless. If qualified synchronization mechanisms are applied or local storage is avoided, session continuation is possible.	The RemoteUI client is stateless. All state is saved in the server session. It is possible to continue from a saved state, if the server-session can be recreated (e.g. by user-id).

3.12 Support server-initiated updates (server-push)

Approach	HTML5	RemoteUI
Use a persistent connection with the server	Different technologies have emerged to realize this feature in HTML. While polling is an inefficient technology, long polling, HTTP streaming, Comet Services etc. have been applied. Recently the WebSockets standard has been introduced, that solves this problem in a standardized way. However, this feature is not yet available in all browsers: http://caniuse.com/#feat=websockets	RemoteUI uses WebSockets to connect with the web server. On server-side a growing number of JEE web servers support WebSockets, enabling the usage of the RemoteUI JEE framework. All RemoteUI clients support this technology. Other technologies, e.g. long polling, can also be added in case of need. RemoteUI can alternatively use a persistent TCP connection to a special RemoteUI server or a Bluetooth connection (RFCOMM).

4 Development Requirements

Usually it is desirable to minimize the development effort for realizing web apps. We suppose the availability of different tools to be an indicator for prospective development effort.

4.1 Tool-Support

Approach	HTML5	RemoteUI
UI Editor	Comprehensive amount of available HTML-Editors available - WYSIWYG and source code based.	At the moment descriptive UI language (XML) with preview in RemoteUI client. Special editor is planned.
Debugging client code on the target device	Debugging is possible by adding additional debug libraries or using browser-specific debug tools.[11] [12] [13] [14] JavaScript source level debugging with some tools requires changing productive JavaScript code. Debugging with	Debugging on the target device is not required when developing a web application with RemoteUI, because the client does not execute any script code. Should debugging the RemoteUI client itself be necessary, this can be

11 https://people.apache.org/~pmuellr/weinre/docs/latest/Home.html
http://lexandera.com/aardwolf/

12 https://developer.apple.com/technologies/safari/developer-tools.html

13 https://hacks.mozilla.org/2012/08/remote-debugging-on-firefox-for-android/

Approach	HTML5	RemoteUI
	one of the browser-specific tools seems to be the best approach.	done with: **Android:** Eclipse Debugging **iOS:** XCode Debugging
Profiling client code on the target device	Several JS-libraries are available for profiling in arbitrary browsers. Profiling with the latter requires changing the productive code. Some browser-specific tools support remote profiling.[12] [14]	If the client is to be profiled (should not be necessary): **Android:** Profiler built into Eclipse **iOS:** XCode Instruments
Code completion for client code	Difficult due to untyped script language, however different tools provide satisfying code completion.	A RemoteUI web application is implemented on the server, e.g. with Eclipse WTP. The code completion capabilities of Eclipse are excellent.
Refactoring support for client code	Difficult due to untyped script language, however different tools have basic refactoring support.	Excellent (Eclipse, see above)

5 Security Requirements

5.1 Allow to enhance security while preserving functionality

Approach	HTML5	RemoteUI
Allow functionality-preserving security patches	Security patches usually come as browser updates. The latter can also contain unwanted changes to JavaScript APIs, rendering behavior or other functionality. Additionally, HTML5 is a "living standard". For this reason, browser updates sometimes break existing applications.	Due to the full control over server and client source code, security patches are possible, while preserving all functionality and protocol features.
Allow extension with new security features	New security features, e.g. authentication mechanisms, most of the time are only possible by developing browser-specific plugins. By choosing a plugin-based solution, the project is automatic-	All RemoteUI client and server source code is available to partners and allows modification. This includes the extension with new security features. Because RemoteUI clients are native applications, nearly

14 https://developers.google.com/chrome/mobile/docs/debugging

	ally bound to the update cycles of the targeted browser. Oftentimes at least major browser updates render existing plugins non-functional.	everything can be implemented.

5.2 Avoid plain sensitive information on the mobile device

Approach	HTML5	RemoteUI
Secure all local information (cache)	Unless the browser cache is encrypted, a root user can get access to all data that have been cached, stored in cookies or local storage etc. The default browsers do not encrypt their cache. A possible solution is to disable caching, which however conflicts with different performance requirements (3.2, 7.2). One solution is to develop a custom browser with cache encryption, as done by some software providers (conflicts with 3.10).	Client side cache encryption is planned as optional feature.

5.3 Provide security by design for typical vulnerabilities

The OWASP[15] gives useful hints how to implement secure web applications. In their top ten[16] of security risks many typical failures are described. Some of them are highly related to client behavior and therefore relevant for technology decisions. The following table only names some of them:

Approach	HTML5	RemoteUI
Avoid Cross-Site Scripting (XSS)[17] and Cross-Site Request Forgery[18]	Problem: The browsers sends information to another server, not intended by the user, because injected code in an HTML page makes the browser request "foreign“ resources. The problem is solvable by using	The RemoteUI client by design does not send data to other than the currently connected server. The RemoteUI server can however send URIs to let the RemoteUI client open them in other applications. These URIs must not contain sensit-

15 https://www.owasp.org
16 https://www.owasp.org/index.php/Top_10_2010-Main
17 https://www.owasp.org/index.php/Top_10_2010-A2
18 https://www.owasp.org/index.php/Top_10_2010-A5

Approach	HTML5	RemoteUI
	appropriate server code filtering user input, however many developers just forget this.	ive information (e.g. originating from input fields in the RemoteUI view).
Avoid Failure to Restrict URL Access[19]	Access to web resources is controlled by assigning required rules to URL patterns. The OWASP recommends to check every URL (page / resource) of the web application for its security. HTML by design works with linked content. Although a WebSocket-based HTML5 app could minimize the effort to control every page, this app architecture is unusual.	A RemoteUI application only has a single URL (the WebSocket address). It is sufficient to check access rules for this URL. (If different roles shall result in different functionality, this can be achieved with RemoteUI's server side code level protection and UI-annotations.)

6 Accessibility Requirements

6.1 Allow screen reader access to user interfaces

Approach	HTML5	RemoteUI
Let the OS's built-in screen reader read UI content	**Android:** Android's screen reader is available from version 4.0 upwards. It cannot read HTML content displayed in Android's browser. Firefox however supports the screen reader. **iOS:** The built-in screen reader reads content displayed by the browser.	**Android:** Android's screen reader is available from version 4.0 upwards. It reads native user interfaces, including RemoteUI generated UIs. **iOS:** The built-in screen reader reads native UI elements including RemoteUI generated UIs.

7 Common Requirements

7.1 Minimize the amount of client-side business logic

Reasons for this requirement are:

- Developing and maintaining business logic in a single location and in a single programming language is less effort.

19 https://www.owasp.org/index.php/Top_10_2010-A8

- Client-side logic is more vulnerable for attacks and modification than server-side logic.
- If client-side logic is deployed as interpreted script or byte code, it is possible to copy and reproduce it. This is e.g. the case for JavaScript or Java byte code (Java applications or Android applications).

HTML5	RemoteUI
The amount of client side business logic depends on the implementation. Oftentimes the use of JavaScript-based rich clients with a significant amount of client side business logic are propagated when talking about HTML5. E.g. local storage or index db are common techniques to realize rich clients that have the advantage being executable without an online connection. Such a solution usually benefits from lower overall traffic (see 7.2). It is also possible to realize HTML5 clients with minimal business logic. They are not able to work offline, however fulfill the above mentioned requirements.	The RemoteUI client is thin. It does only provide UI logic. All business logic is located at the server by design. As a result a RemoteUI application requires a network connection. If desired, it is however possible to embed the RemoteUI client components into a native application. Parts of this application could be executed in offline mode.

7.2 Minimize data amount, maximize performance

Approach	HTML5	RemoteUI
Minimize initial download size	All app logic and resources for displaying non-builtin UI elements (see 3.1) must be downloaded (includes images, styles, JavaScript code). Because of the same origin policy[20] once loaded content cannot be reused (e.g. from cache) when connecting to a different server. Some (desktop) browsers provide a user script mechanism to install scripts that are not subject to the same origin policy. An alternative are plugins or extensions. These solutions however are not stand-	On initial load RemoteUI transfers a handshake and the UI description of the first view. No business or UI logic is transferred. The client has all UI logic built-in and business logic is kept on server side by design.

20 https://en.wikipedia.org/wiki/Same_origin_policy

Approach	HTML5	RemoteUI
	ardized and require browser-specific programming. Pre-installed mobile browsers typically do not support extensions or user scripts (conflict with 3.10).	
Minimize traffic during application runtime	The "richer“ an HTML5 application is (see 7.1), the less traffic is usually generated after the initial download has been completed. Oftentimes data are transferred as business entities, that need less data amount than their UI counterparts.	The RemoteUI system does not transfer business entities but only UI descriptions, events and manipulation commands. The resulting traffic during runtime is probably slightly higher than with business entities. This depends on the specific application and might be compensated by RemoteUIs overall data amount minimization techniques.
Use an efficient image codec for pixel image transfer. The codec must support alpha channel encoding, because many UI elements rely on it.	Typical image formats with alpha channel support are PNG and GIF. While full color PNG (PNG-24) provides full quality encoding but low compression, PNG-8 and GIF limit the number of colors to 256 introducing significant quality loss for certain image types. WebP is a solution that however is not supported by mobile browsers[21].	RemoteUI uses WebP as its default image format. Alternatively a fallback to PNG-24 is possible.
Support binary encoding for message transfer	JSON and XML do not allow the efficient encoding of binary data (Base64 introduces overhead of about 33%). It is desirable to transfer at least images in binary encoding. If additionally server push (3.12) shall be combined with this requirement, the browser must be able to handle binary data within JavaScript (and binary WebSockets). Support for this fea-	By default, RemoteUI uses binary encoding for all messaging, object transfer and images. Switching to an XML variant of the RemoteUI protocol is possible, using Base64 encoding for images. Additionally the serialization functionality in RemoteUI is modular and can be replaced by alternative

21 Only Chrome on Android seems to have support for WebP but at the time of this writing has a bug regarding colors.

Approach	HTML5	RemoteUI
	ture has only recently been added to mobile browsers (excluding e.g. 2.3. the Android browser): http://caniuse.com/#feat=blobbuilder http://caniuse.com/#feat=typedarrays Alternatively images can be loaded via traditional HTTP GET. The server can only push the URL of the image and the browser loads the latter. This however introduces an additional network roundtrip, negatively influencing the responsiveness of the system (conflicts with 3.2).	libraries. This way, also a JSON variant is possible.
Keep the communication necessary to establish a (secure) connection to a minimum	Several technologies have been introduced to reduce the overhead of establishing connections for the download of different resources building an HTML web page (styles, scripts, images...). One of them is HTTP 1.1 persistent connections to allow several requests and responses over a single TCP connection. This avoids the TCP handshake and is well supported by most browsers. When handling SSL-connections, it is desirable to have a browser supporting SSL session resuming. This avoids recalculation of a new secret for every connection. There is no overview about which browser supports this feature. The Browser of Android 2.3.4 does not[22].	RemoteUI only creates one connection, requiring an initial TCP-, an optional SSL- and a WebSocket-handshake. After the connection is established, it is used throughout the complete session. For reconnection cases (e.g. due to connection loss), SSL resume will be supported.

Approach	HTML5	RemoteUI
	An HTML5 app using WebSockets can lower the number of required connections. Another alternative is the use of SPDY[23], support for it is however incomplete: http://caniuse.com/#feat=spdy	
Support content compression	HTTP gzip and deflate are supported by all browsers. The so called CRIME[24] attack has lead to SSL compression being disabled in recent browsers for HTTP and SPDY as well. However SSL compression combined with HTTP compression only compresses the HTTP headers, only slightly reducing data amount. It is recommended to turn HTTP compression on, which is controllable via the server.[25]	Zlib support is planned, either via WebSocket per frame deflate[26] or message compression.
Follow common performance recommendations	Google describes web speed best practices[27]. Additionally they offer technologies like the mod_pagespead Apache module, the SPDY protocol or the WebP image format. Many of them can only be used, if the client and the server have corresponding support.	RemoteUI adopts many of the recommendations already known from existing web applications. Additionally it uses new technologies like e.g. the WebP image format. There are no compatibility problems, because client and server are tightly coupled software projects.

22 http://vincent.bernat.im/en/blog/2011-mobile-browsing-content-optimization.html
23 http://googlecode.blogspot.de/2012/01/making-web-speedier-and-safer-with-spdy.html
24 http://isecpartners.com/blog/2012/9/14/details-on-the-crime-attack.html
25 https://developers.google.com/speed/articles/use-compression
26 https://tools.ietf.org/html/draft-tyoshino-hybi-websocket-perframe-deflate-00
27 https://developers.google.com/speed/

8 Summary

The arguments for using HTML5 to realize mobile web applications at first sight are plausible. With the approach "Write once, run anywhere“[28] the technology promises minimal implementation effort with maximum market reach. A detailed look at the features and performance of mobile browser implementations however reveals, that it is extremely difficult to support all implementations while still providing the rich and fast user interfaces, the users expect. Oftentimes different browsers interpret markup or script in different ways. This makes testing and debugging time-consuming and costly. "Write Once, Debug Everywhere" once was the title of an article by William Wong[29] where he complained about Java being not as portable as promised. With JavaScript we see the same problems reappear today, however with a significant difference: Debugging everywhere is not even possible (see 4.1).

The situation of HTML5 on mobile devices improves with each browser update. A major problem however is the bad update policy of most Android device manufacturers resulting in Android 2.3 at the time of this writing being the most used Android version. The built-in browser of this OS lacks many features.

One solution is to abandon the idea of "Write once, run anywhere“ and just prescribe a certain "target browser", minimizing test and debugging effort. There is however no browser implementation running on all major mobile platforms. Google Chrome is available for iOS and Android. On iOS it is however a kind of "slow Safari“, not supporting the features of its Android counterpart (see 3.9). On Android Chrome is only available since version 4.0 of the OS. At the moment there is no implementation for Windows Phone. It seems that customizations, tests and debugging for all platforms (e.g. Safari on iOS, Firefox on Android, Internet Explorer on Windows Phone) are inevitable when using HTML5 for mobile web applications.

The RemoteUI system is based on a thin client idea, that offers many advantages but also requires a persistent online connection. The installation of the RemoteUI client on the target platform is required. RemoteUI web applications are implemented on the server resulting in centralized and easy to maintain business logic for all supported client platforms. All components of the system are optimized for mobile usage scenarios. RemoteUI provides:

- A state of the art server side web framework with data binding, annotation-driven security configuration & validation. Based on the Spring Framework[30], the RemoteUI framework can be integrated in nearly every Java-based software architecture.
- Server-controlled content adaption for a diversity of different mobile devices
- Binary serialization and messaging over a persistent connection, minimizing data amount and latency

28 Originally a Java slogan: https://en.wikipedia.org/wiki/Write_once,_run_anywhere

29 http://electronicdesign.com/article/embedded-software/write-once-debug-everywhere2255

30 http://www.springsource.org

- Support for UI prioritization that has no counter part in HTML
- A considerable amount of UI widgets that render fast and have native look & feel
- Features and extensibility for secure web applications, e.g. the possibility for client side cache encryption and simple security configuration on the server
- Full access to client and server source code for dedicated partners allowing extensions, customizations and optimal maintenance

In addition, it is possible to exchange different modules of the RemoteUI system to allow the realization of an HTML5-based RemoteUI client. The serialization module can be configured to use XML, as an extension even JSON is possible. Images can be encoded with PNG, if the browser does not support WebP. The abstract UI descriptions can be transformed to HTML and CSS either with JavaScript or XSLT.

It can however be expected, that natively implemented RemoteUI clients perform better than an HTML5 counterpart. As soon as mobile browsers better conform to the requirements discussed in this document, it might make sense to reconsider the idea of "Write once, run anywhere".