Apps: Native vs. Hybrid vs. Mobile

Autor: Ronny Hartenstein, Axilaris GmbH, Stand: 17.9.2016 (start am 28.8.2016)

TL;DR

Sobald man eine App in den Stores veröffentlich, geht die breite Masse der Endkunden davon aus, dass er eine echte native App bekommt - und wird die Usability, UI-Snappyness und Gesamtperformance daran messen. Aufruckelnde Burger-Menüs sind da ein Killer.

Time-to-market kollidiert hier mit Nutzeranspruch und noch fehlenden Know-How.

Würden wir Stand jetzt eine App aufsetzen müssen, die wesentliche Teile von Obelisk nutzen soll und einige App-Features wie nahtlose Kameraunterstützung haben soll, kann die Empfehlung nur klar Hybride Web App lauten. Mit plain Cordova-Vorbau oder evtl. Meteor.

Wollen wir eine App bauen, die einige wenige Features sehr sehr gut umsetzt und dabei sich wie eine echte App anfühlt (BÄM-mäßig), müsste es Javascript Native sein (Definition s.u.). Dabei wird der Einarbeitungsaufwand (das noch fehlende Know-How) mit weniger Mühen im Cordova-Webview-Zusammenspiel belohnt. Updates müssen teilweise durch den Store - der JS-Kern kann wohl ad-hoc aktualisert werden.

Sollen wir eine unternehmensinterne App aufsetzen, ist eine Hybrid App super, da der Endnutzer nicht zwischen 7 anderen Anwendungen wählen kann.

Inhaltsverzeichnis

Was haben wir?

Cardster

Zielgruppe: Endanwender
Natur: App mit Fokus auf Konsum, wenig Interaktion, eine “Frankenapp”, s.u.
Installation via Appstore
Erwartungshaltung: wie eine “echte” App, müssen es aber nicht verwenden, Schlüsselreiz ist Gewinnspiel-Teilnahme via Sterne-Sammeln
Prinzipiell gibt es keine 7 Alternativen, aber für einen Alltagsbegleiter fehlt ein Mehrwert.

Technik

- Native Apps für iOS und Android, jeweils handgeschrieben
- Apps haben Cordova eingebunden mit unter iOS veralteten UIWebKit statt WkWebView
- Ember MVC Framework für Single Page Apps, die mit einer JSON-REST-Schnittstelle sprechen
- REST-Backend ist Obelisk mit PHP
- Frontend UI-Elemente ist Bootstrap + spezielles Layout Look&Feel - keine nativen UI-Elemente
- kann theoretisch per Browser bedient werden, ist aber dem Endkunden nicht bekannt.

Vorteile:

- schnelle Reaktion durch Ember JS+HTML +
- Backend ist bekanntes Terrain mit Obelisk: Banklogik, Fachlichkeit, Cronjobs

Probleme:

- Verzögerungen bei Login / Abfragen werden als Performance-Probleme angesehen
- Kunde hat die Erwartungshaltung einer echten Nativen App
- Anpassungen an der Rahmen-App müssen durch den App-Freigabeprozess
- Rahmen-App liegt nicht in unserer Hand,
- diese könnten wegen Unkenntnis der Sprachen (Objective-C, Java) und der jeweiligen nativen APIs ohnehin nicht gut von uns gepflegt werden

Idee:

- kontinuierlich Ping gegen Backend, wenn länger als x dauert, dann "Verbindungsprobleme" anzeigen
- aber damit sind keine Geschwindigkeitsmessungen möglich
- und es müsste in den nativen App Teil eingebaut werden

S'peak

Zielgruppe: Sparkassen-Mitarbeiter während Konferenzen
Natur: Mobile Webseite mit App-Features wie Chat, Notizen
Installation via App-Store? derzeit ausschließlich über Browser genutzt
Erwartungshaltung: wie eine "echte" App, haben aber keine Alternative, müssen es verwenden - es gibt keine Apps die ähnliches leisten.

Technik:

- Meteor Framework 1.4
- läuft in Webbrowser,
- kompiliert zu Android und iOS mit Cordova 4 und WkWebView
- Frontend ist HTML+JS, welches innerhalb der App mit ausgeliefert wird
- Updates werden beim Start nachgeladen und in der App gecacht (Update-on-the-wire)
- Frontend UI-Elemente in App sind Bootstrap - keine nativen UI-Elemente
- Frontend kommuniziert zum Backend via Websocket
- Backend ist NodeJS, Javascript
- Backend wiederum kommuniziert zu Obelisk via REST-Schnittstelle (JSON)
- in Obelisk wird Login abgewickelt
- Inhalte werden von Obelisk ausgeliefert und in Frontend in IFrame dargestellt
- App-nahe Funktionen: Chat, Notizen (in JS implementiert)

Vorteile:

- App-nahe Funktionen sind sehr schnell realisierbar

Probleme:

- Login funktioniert ab und an nicht mehr - Meteor muss dann neu gestartet werden - Websocket liefert noch richtige Antworten, wird aber vom FE nicht mehr verarbeitet

- Scrolling in Iframe von Außen ist tricky

Update 17.9. Single Page oder nicht?

Cardster nutzt Ember, ein SPA-JS-Framework. Theoretisch ist es auch als normale PHP-Anwendung mit normalen Seitenwecheln und Formularen denkbar.

aus [11]
Single page webapps, I believe, come from a very, very strange reaction to deathly slow Rails/Django/PHP backends. Perhaps it's the single pager that's the antipattern….

AgroSecure

Zielgruppe: eigene Mitarbeiter des AG -
Natur: Enterprise-App
Erwartungshaltung: klassische Enterprise-App, muss man als Nutzer nehmen wie sie ist :)

Technik:
- Meteor 1.2 (veraltet) mit langsamen UIWebKit Container
- von der Natur her eine echte App, keine Mobile Webseite
- Frontend UI-Elemente sind Standard Bootstrap 3 Komponenten + leichte Designanpassungen (Theme)

Haushaltsbuch

Zielgruppe: Onlinebanking-Nutzer
Natur: Webseite, ist aber keine App
Erwartungshaltung: da ins Onlinebanking eingegliedert.. Webseite mit dynamischen Elementen

Technik:
- Frontend sind JQuery-Widgets
- Backend ist Obelisk-Ajax
- Frontend kommuniziert zu Backend via XHR

Problem:
- Widgets kennen nur sich und unterliegende Widgets (z.B. Splitting-Icon mit Splitting-Maske
- Kommunikation zw. Widgets, z.B. Diagramm zu Zettel, Icon zu Reiter-Tab
- zentraler URL-Hash-Hub: wie bekommt man Infos in die Widgets?

Lösung wäre:
- Redux Observer als zentrales Messaging-Backbone
- ggf. Ansätze von React: State (Zustand) oben rein, View (Ausgabe) unten raus. Aktionen ändern State.

Arten von Apps #1

Liste mit Quellen siehe unten..

aus [8]
Creating an amazing mobile experience has become the holy grail for JavaScript developers in recent years.
We spent days optimizing performance and user experience, dreading the most feared enemy of all. The page reflow. We still do.
Safari has become the new Internet Explorer. The only major browser that still only receives major updates yearly.

let's break the mobile web into three categories

1. *Mobile Websites*
2. *Mobile Web App*
3. *Hybrid Mobile App -> Difficulty Insane*

Mobile Website

aus [8]
your friendly informational website
very little interactivity involved
navigate around and read articles
share a page and submit a form like "Request a quote"
Competent developers can create great mobile websites [..] with a medium amount of effort
[but] gazillion analytics/ad scripts these sites use that end up hurting performance
million annoying popups, fixed sharing toolbars and fixed headers that fill the already small mobile view
when we remove all the clutter so many websites inflict on your helpless mobile browser, we end up with a pretty great experience

Difficulty Level: Medium

Mobile Web App

aus [8]
mobile versions of Twitter, GoDaddy, and Udemy
difference between mobile web apps and mobile websites is purely semantic. I'm sure that 5 different people will have 5 different definitions of what exactly the differences is.
A mobile web app's main goal is to enable interaction, user engagement and allowing the user to perform different actions.

Difficulty Level: Hard

It's hard to create great mobile web apps. [..] if you ask the team behind the new mobile.twitter.com they'll tell you it was no cakewalk. Same goes for m.facebook.com. when you compare these two great examples to their native app counterparts, they kinda suck.

Mobile Web Apps are Hard Even for Google
visit the mobile web versions of flagship Google products such as Gmail, Google Inbox, Google Photos and Google Calendar.
You'll be unpleasantly surprised to be greeted by something Google calls "App Download Interstitials." This term loosely translates to: "An annoying modal window that covers the entire screen and invites to download the native app version of a website."
Yes, Google is actually violating its own mobile web guidelines in several of its flagship web apps.
Google Inbox flat-out doesn't even have a mobile web version. They didn't even try. Priceless

Hybrid Mobile App

aus [7]
The promised land. The land of the App Store
The way this works is simple: Both iOS and Android have something called a WebView component which allows you to render web pages. Think of it as Chrome minus the tabs, url bar and everything else that isn't part of the browser's viewport. This component is only the part that actually renders the web page.

Difficulty Level: Insane

By creating a hybrid mobile app we've taken all of the complexity of a mobile web app and added a new equally complex layer on top of it. This layer includes the native app's behavior and functionality.
On top of the added complexity, we've also significantly changed our user's expectations. Users will no longer expect a "web experience" but a "native experience" instead.
Every time your hybrid mobile app won't behave exactly like its native counterpart your end user will feel somethings wrong and "behaves weird." This means we're sort of setting ourselves for failure before the first line of code was written.
I'm now going to share with you something now that I've learned over the last 3 years working on a hybrid mobile app. Something PhoneGap, Ionic or any other hybrid app evangelist will fail to mention:

> *Building hybrid mobile apps will require triple the effort and triple the developer proficiency to achieve the same or slightly worse results than their native counterparts.*

There's so much native developers "get for free" and so many workarounds & hacks required to make this whole thing work. After all of that, the end result still isn't what I would call "show off worthy."
That's why I have to say the investment in hybrid mobile apps just isn't worth it. At least not for consumer apps.

There's only one place where going down the hybrid app path makes sense: The Enterprise™.
When you create a custom enterprise app you know the user can't just go to the app store and choose between 7 different apps that do the exact same thing.
Only your app will be fulfilling that specific function. And supporting both iOS and Android from launch is crucial. Having a web version is an added bonus too.
Enterprise users still set a high bar when it comes to mobile user experience. But you don't have to wow or dazzle them. And you'll usually end up enabling functionality that was only available on the desktop up until that point.
For these reasons and a few more I won't get into, hybrid mobile apps found great success in the enterprise.
But little success anywhere else.

Update 17.9.
Aktuelles Beispiel einer Enterprise-App ist mobileX-CrossMIP. Cordova + JS vermutlich in der App + Offline-Funktion via Sqlite-Sync.
https://www.mobilexag.de/de/blog/mobilex-crossmip-ab-ende-august-verfuegbar
Warum Cordova?
https://entwickler.de/online/javascript/javascript-statt-xamarin-interview-254899.html

The Prophecy

aus [7]
But there is hope!
A different breed of JavaScript mobile frameworks has come to maturity. These mobile frameworks use JavaScript runtimes but aren't browser based.
Instead, frameworks such as React Native and NativeScript let you write JavaScript code while rendering native UI components. It sounds amazing, and it is.
But there's a growing debate in the JavaScript community whether these frameworks are generally a good thing.

Spoiler: React Native ist gemeint

Arten von Apps #2

aus [2]

- *Native Development*
- *Cordova based Hybrid app development -*
- *Mobile app development in React Native*
- *App development using Xamarin*
- *Last but not the least DIY (Do It Yourself) tools*

Selection of the right tool/platform upfront is a very crucial factor in the long term mobile app development strategy.

most prominent factors in choosing a specific app development platform or approach depends on the below key factors –

- *Skill set of the individual or development team*
- *Unique requirements of the app*
- *Targeted user segment*
- *Long term strategic vision of the individual or an organization.*

hybrid frameworks like Ionic, Onsen UI, Intel XDK and Sencha Touch to name a few. All these frameworks deal mostly with JavaScript, CSS and HTML only, and support almost all mobile platforms

aus [6]
different options for mobile app development — native, hybrid and what we're calling "JavaScript Native"

aus [7]
A day rarely goes by without someone lamenting the "problems" of web technologies, often specifically on mobile. They're either too slow, lack native look-and-feel, or just don't have a smooth enough UX. Their death has long been prophesied in the annals of technology.

Native

Apps auf iOS: Objective-C od. Swift?
aus [6]

building applications using each platform's native language and development tools typically associated with the very best performance, as you're coding directly against the native platform APIs

higher development costs, as developers must learn the how to build for each platform individually
very little skill reuse across platforms (aka mastering Swift won't really help you build an Android app)
for cases where every ounce of performance is critical, native may be the only option

Wann Performance wirklich wichtig ist

> *Many developers might be saying to themselves, "What do you mean? Performance is always critical!" Well, that is true — to a degree. Think about it this way, the engine in a Formula One car theoretically performs better than the engine in a consumer car (at least from a speed perspective anyway), but this does not mean that consumer cars should start having Formula One engines. The performance of an application is as much about expectations and perceptions as it is about actual performance — and those expectations are often heavily dependent on the nature of the application you are creating.*

need to maintain multiple codebases for a single app that targets multiple platforms

aus [9]
[its] weird and inefficient that most companies nowadays have two completely separate mobile development teams, one for Android and one for iOS. They each write apps which are 90% similar and yet their codebase is 90% different.
How weird is it to be on one of those teams? Always playing catch with another team who's developing the exact same thing you are, but using a completely different set of technologies.

Frankenapp (Hybrid ohne Framework)

aus [7]
When you set out to build a fast, high quality mobile app with web technologies, it's not nearly as simple as just throwing in some "mobile-enabled" widgets, writing some jQuery, and calling it a day. Throwing together a responsive framework with a slider widget and a hamburger menu widget isn't going to make your app feel like, well, an app. Instead, you get what often feels like a "Frankenapp."

Your homemade UI just does not look nor feel like a native app, period. This is the #1 reason Cordova has earned a reputation for being slow and non-native. It has nothing to do with what Cordova does, but everything with what it doesn't do

Hybrid

aus [6]

creating a native app that launches a WebView, or an embeddable browser instance provided by mobile OSes (i.e. iOS, Android, and so forth), and uses that WebView to drive your native application.
using the same technologies you would use to build a web app: HTML, JavaScript, and CSS
Today, most hybrid apps are built on top of Apache Cordova
Cordova provides a consistent set of JavaScript APIs to access device capabilities through plugins, which are built with native code.
the Cordova camera plugin API lets you access your device's camera using a call to navigator.camera.getPicture().
offers the most opportunities for code reuse from your existing web assets.
has suffered from perception of its performance, a view that wasn't helped when hardware vendors chose to leave important performance enhancements out of the WebView available to hybrid apps(though, this situation has improved dramatically in recent releases).
→ seit WkWebView auf iOS ein kleineres Problem

aus [7]
Very few developers should be using the browser primitives as-is. Instead, they should use frameworks. They should modify UI through an MV framework like Angular, Ember, or React, which will properly batch and update the UI to avoid costly reflows and subsequent flickers. They should use mobile web frameworks like Kendo, Sencha, or Ionic to handle the actual mobile components (buttons, tabs, navigation, etc.) instead of writing their own, which are guaranteed to have performance and UX issues (for context: we've been agonizing over how to build the perfect mobile button for over two years!).*

Um den Unterschied von Nativer App Entwicklung zu Hybrid noch besser zu verstehen, sollte der gesamt Artikel [7] gelesen werden.

JavaScript Native

aus [6]
a series of JavaScript-based solutions have been garnering a great deal of attention. These solutions have native UIs but aren't purely native, as the underlying code is still deployed as JavaScript. They are built, at least in large part, with web technologies like JavaScript and, in some cases, CSS, but aren't truly hybrid, as there is no HTML or WebView

popularized by Appcelerator Titanium
has come back into the spotlight with the recent release of both React Native and NativeScript
they do not use a WebView; there is no HTML, and there also is no DOM.
their user interfaces using the platform's native UI components
NativeScript and Appcelerator Titanium use XML, and React Native uses JSX [..] for defining UI components
example, a <button> element in NativeScript is actually implemented as UIButton on iOS and an android.widget.Button on Android, but the native implementation details are abstracted away from the developer

interpret JavaScript code at runtime. here is no build step that takes your JavaScript code and converts it into Objective–C/Swift/Java/C#.
they utilize a JavaScript virtual machine to interpret JavaScript code, and to translate that code into the native APIs that build the app's user interface

Tabris.js https://tabrisjs.com/examples
Appcelerator Titanium https://www.appcelerator.com/mobile-app-development-products/

Dank Javascript ist es immernoch Single-Thread mit Callbacks (Callback-Hölle, Promises..). D.h. wenn JS rechnet, blockiert die App. Mit Safari 10 und Chrome auf Android funktionieren aber ServiceWorker!
→ spielt für uns eine untergeordnete Rolle

aus [6]
the performance implications are often overstated and are unlikely to impact a large number of applications — particularly things like internal business applications, where the fast iteration and skill reuse may offer the most benefit.

aus [6]
JS Native challenge the conventional hybrid vs. native logic.
Need the very best performance and native user interfaces? Go native.
Need to reuse web development skills? Go hybrid.
JavaScript Native frameworks complicate these guidelines, as JavaScript Native frameworks can offer performance that's comparable with truly native apps, let you build interfaces with native UI components, and also let you reuse web development skills.
can offer one of the web's best features — a fast development cycle
Video über das Styling eines Buttons und Live-Update

[is not a] panacea for all mobile development problems
does not mean that [it is] are appropriate for all development teams.
a few hurdles you may hit

- *it's highly likely that you'll need to learn a bit about how the native platforms work in order to build complex and performant UIs*
- *Memory management is far more important*
- *CSS-like implementations aren't the same as a browser's CSS implementation. [they] take CSS properties and convert them into their native equivalents.*

aus [9]
It's React
It's Flexbox
It's Javascript
It's Native

But I'm not quite sure that React + JavaScriptCore + Flexbox + XHR + WebSockets qualify as "The Web."

By choosing React Native and the like, Web developers are no longer developing for the web. Even though it certainly still feels like it for the most part.
Web Platform evangelists may say React Native is not the hero JavaScript deserves. But I say React Native the hero JavaScript needs right now.
React Native is the framework to which mobile JavaScript apps will be compared to from now on. Web or otherwise.

Cross-compiled Native Apps

aus [6]
the language you write your code in is different than the platform's native development language.
Xamarin lets you write iOS and Android apps in C#,
RubyMotion lets you write iOS and Android apps using Ruby,
RoboVM lets you write iOS and Android apps using Java
unlike JavaScript Native apps, cross-compiled apps statically compile your application code from the language it's authored in to the language of the target platform
→ kein Hot-Code-Update, es ist immer ein App-Store Freigabe-Prozess nötig

JS Native oder Hybrid?

aus [6]

- *Hybrid offers the quickest way to an app with the most code reuse, but has performance implications and a non-native UI.*
- *Native offers the fastest performance and access to the full breadth of OS features, but has the complexity of maintaining multiple codebases and the need for developers with platform-specific development skills.*
- *JavaScript Native frameworks offer some of the code and much of the skill reuse in addition to a web-like, fast development workflow. But, JavaScript Native apps don't completely negate the need for platform-specific coding knowledge, and by targeting multiple platforms with a single codebase, some platform-specific features or styling may be unavailable or difficult to access.*
- *Cross-compiled native apps offer developers the ability to use a language they're already familiar with, as well as existing frameworks and tools from their ecosystem of choice. Cross-compiled native apps can also offer native-like performance, as true native bytecode is used at runtime.*

The trick is to understand the differences between these approaches, and choose the option where you can take the most advantage of the merits while being least affected by the complications. This will change depending on the type of application you are building, and the skillsets you and your team have

Progressive Web Apps

aus [10]

TL;DR: *sind mobil nutzbare Webangebote, die sich progressiv an die Fähigkeiten des sie nutzenden Gerätes und Browsers anpassen. Je leistungsfähiger das Gerät und der Browser, desto leistungsfähiger die App. Zur Erstellung werden ausschließlich offene Webstandards verwendet.*

Application Shell. *App im Kern erstmal eine mobile Ansicht einer URL zeigt und dann auf die Fähigkeiten des aufrufenden Gerätes und Browsers reagiert. Auf diese Weise bleibt sichergestellt, dass eine „Progressive Web App" ganz grundsätzlich auf jedem Gerät und Browser funktioniert, allerdings in unterschiedlichem Umfang, eben progressiv angepasst an die aufrufende Umgebung.*

Service Workers. *Das Herzstück einer „Progressive Web App" sind die Service Workers, die fortgeschrittene Funktionalitäten erst möglich machen. Ein Service Worker ist - verkürzt gesagt - ein JavaScript, das, anders als bislang möglich, im Hintergrund einer Webanwendung arbeiten kann. Das funktioniert sogar dann, wenn die Website gar nicht geöffnet ist.*
Support [für ServiceWorker] in Chrome, Firefox und Opera, IE ist angekündigt, Safari ab iOS 10
Push-API werden wir wohl eher nicht auf Apple-Geräten sehen

ServiceWorker für Offline-first Anwendungen
Installation auf Smartphone aus Browser heraus (mit App-Banner) via Web App Manifest (JSON-Datei)
Mit entsprechend konfiguriertem Service Worker legst du bei der Gelegenheit gleich mindestens die App Shell auf dem Gerät ab

Frameworks

aus [3]

First of all, congratulations! We are in 2015 and this means you are way better of then 5 years ago, when hybrid development began to raise as a very promising area. The problem is, it was so shitty at that time, all the crappy apps have been burned into the minds of people which lead to preconceptions all around the globe.

Cordova

aus [4]
you have the ability to make calls to native APIs, but the bulk of your app will be HTML and JavaScript inside a WebView
While you can approximate native components – and it is certainly possible to build a great UI with HTML and JS – no Cordova app will match the look and feel of a real native app
The little things – like scrolling acceleration, keyboard behavior, and navigation – all add up and can create frustrating experiences for your customers when they don't behave as expected

Meteor

aus [1]
it does is provide a real-time communication channel to a pre-built backend stack (Node, Mongo/SQL).
build your app in Ionic / React Native, and use Meteor as your server/database/real-time. There are plugins to ease Meteor integration with Ionic / RN, so you won't have trouble

Live Updates

Ember

Ähnlich wie Ionic, aber ohne UI-Komponenten
Ich habe jetzt nicht speziell recherchiert, da wir das ja kennen (Möglichkeiten und Grenzen).

Ionic-Komponenten in Ember nutzen
https://www.npmjs.com/package/ember-ionic

Ionic

aus [1]
based on Angular
Angular itself is easier to learn & work with for smaller projects than React
is a breeze to learn & work with, and cross-compiles to iOS and Android (since it's just Webview) using per-device differentiator classes to theme accordingly - write once, run anywhere.
uses Cordova, which simply spins up your system browser to render you app (called Webview). In the past, working with a hybrid mobile app on Webview was prohibitively slow - you were guaranteed low app ratings just on performance. Nowadays it's less an issue, as

newer phones ship with better system browsers (ie Chrome instead of "Internet"); plus better phone specs
it still is a noticeable performance difference compared to native. Not terrible, but noticeable
Fazit: *use Ionic for smaller projects, for clients with lower budgets, and for apps whose performance ratings aren't dire.*

- bringt Standard-UI-Komponenten mit - ahmt natives Aussehen je Plattform nach

aus [3]
offers great possibilities to build hybrid apps which not only look awesome, but which also behave as natural as a native app and rely on one shared code base

schaut trotzdem auf jeder Plattform gleich (hübsch) aus

*Prototype apps with **Creator***
A simple drag-and-drop prototyping tool for creating real Ionic apps.
Export clean, ready-to-use Ionic code or even iOS and Android native binaries.

aus [7]
it's working. In just a few years Ionic developers have built top apps for the app stores, gaining millions of installs, five star ratings, and even features from Apple and Google. Clearly, for nearly one million Ionic apps and their users, the web platform is fast and good enough.

JQuery Mobile

Sammlung an UI-Komponenten um Native Apps nachzuempfinden
Schaut trotzdem auf jeden Device gleich aus - keine Plattform-spezifischen UI-Komponenten

React Native

aus [1]
based on React (View)
React is easier to maintain for larger projects and teams, as it follows stricter paradigms and design patterns.
their motto is "learn once, write anywhere" - meaning you're using the same framework, and maybe even shared components, but maintaining separate code for your iOS & Android apps
This means more work, on top of the steeper learning curve (in vgl. zu Ionic)
is a different breed than hybrid. You write your code in React components, but those are rendered as native UI components in your mobile app. Your app logic is still JavaScript (which is processed via the system's JS engine); but app logic was never the performance bottleneck of hybrid frameworks - twas UI (due to DOM performance).
will take much more learning, effort, and maintenance - it will give you a much higher quality app.

Fazit: *use React Native for more "hard core" apps: performance is key, budget is higher, or it's my million-dollar idea*

Styling erfolgt via Javascript, mit ähnlichen Präprozessor-Möglichkeiten wie Sass

aus [2]
You need to understand React
you need to understand JSX, JSX is the XML like syntax extension of ECMAScript
you also need to work with Xcode, iOS simulator and command line tools for building Android version of the app
Knowledge of application architecture framework like Flux and related packages like Redux and Reflux etc. help organize and speed up app development
each React native UI component corresponds to the native UI component of iOS and Android
you write components in JavaScript but those are converted to native components

aus [4]
React's best feature when it originally launched for the web was making your app's view layer a pure output of state
Under the hood, your JavaScript code is run on its own thread, separate from the main UI thread
Many software frameworks promise that they will let you make a great app for Android and iOS, but the product frequently ends up somewhere in the middle without feeling truly native to either
***Live Updates** it is possible to do live updates to your app without going through the App Store – much like for a web app. Since the bulk of your app will be JavaScript, you can fetch updates on the fly over the network*
*support for **flexbox** means you can use the exact same layout code for Android, iOS and web, instead of learning three different engines.*

React Native Web
Geteilte Codebasis für iOS und Android, aber derzeit noch nicht fürs Web. Daran arbeitet react-native-web.
https://github.com/grabcode/react-native-web-starter

Xamarin

aus [2]
is among the most feature rich, established and easy to use platforms for rapid mobile app development.
you can build native android, iOS, windows and Mac apps in C# language.
Development philosophy is more or less the same as React native but the skill set requirements change drastically since you need to work with C# and Microsoft tools instead of JavaScript

RubyMotion

aus [6]
there's much more today than simply traditional native or hybrid. For instance, solutions like RubyMotion or Xamarin, that compile to native from Ruby or C# respectively, have gained popularity

NativeScript

aus [9]
There are other libraries who are also trying to "Bring the web into native." The most notable one is NativeScript, which is geared more towards the Angular 2/Enterprise crowd.

von https://www.nativescript.org/
Create apps with 100% native performance and power. Access all native iOS, Android and Windows APIs directly with JavaScript - no messy bridging code required smooth, even on Android.
Beautiful, accessible, platform-native UI - no webviews. Define once and let NativeScript adapt to run everywhere, or tailor the UI to specific devices and screens.
Easily reuse existing plugins from npm, CocoaPods (iOS) and Gradle (Android) directly in NativeScript projects, plus hundreds of NativeScript specific plugins on npm
Write and deploy native mobile apps for iOS, Android and (soon) Windows from a single code base
Architect your applications your way. NativeScript is not tied to any specific JS framework; Angular 2 integration optionally available
Angular and TypeScript. Deeply embedded, first-class support for Angular 2 and TypeScript in the NativeScript CLI
It sounds like magic, and it feels like it, too. Style native UI widgets using the CSS you already know. You can even use SASS and LESS.

weex

aus [9]
Alibaba recently released its own open source framework in this space as well.
http://alibaba.github.io/weex/

erinnert latent an Meteor mit <template> Tag
http://alibaba.github.io/weex/doc/tutorial.html
Kommentare und Ökosystem ist hauptsächlich chinesisch
http://alibaba.github.io/weex/doc/

Analyse

aus [1]
Ionic is easy but slow. React Native is hard but fast. Judge it on your relative time and skill/budget.

Native Apps: 90% unterschiedlicher Code für 90% gleiche Funktionalität

Stellt man eine App in den Store, erwartet der Nutzer eine App - Usability, Oberfläche, Snappyness.
Der Nutzer erkennt (unbewusst) ziemlich schnell wenn “nur” eine Webseite in eine App gerahmt wurde.

Will man eine App bauen die mit den Top-10 in ihren Bereich mithalten will, muss sie nativ sein.

aus [8]
On top of the added complexity, we’ve also significantly changed our user’s expectations. Users will no longer expect a “web experience” but a “native experience” instead.

Quellen

[1] Which Hybrid-Framewok has more future? Ionic, React, or Meteor? (Feb 2016)
https://www.quora.com/Which-Hybrid-Framewok-has-more-future-Ionic-React-or-Meteor/answer/Tyler-Renelle

[2] App Development: Cordova vs React Native vs Xamarin vs DIY (Jun 2016)
http://noeticforce.com/mobile-app-development-cordova-vs-react-native-vs-xamarin

[3] Switching from native iOS to Ionic: Why Hybrid doesn't suck (anymore) (2015)
https://www.airpair.com/javascript/posts/switching-from-ios-to-ionic

[4] Why You Should Consider React Native For Your Mobile App – Smashing Magazine (Apr. 2016)
https://www.smashingmagazine.com/2016/04/consider-react-native-mobile-app/

[5] Native vs Ionic vs Nativescript vs React Native (Jan 2016)
https://www.trustedhousesitters.com/engineering/news/hackathon-jan-2016-native-vs-ionic-vs-nativescript-vs-react-native/

Hackathon um Frameworks für eine Messenger-ähnliche App zu verproben. Native hat gewonnen.

[6] Defining a New Breed of Cross-Platform Mobile Apps -Telerik Developer Network (Okt. 2015)
http://developer.telerik.com/featured/defining-a-new-breed-of-cross-platform-mobile-apps/

[7] The Web Platform is too Low-Level – Ionic and the Mobile Web – Medium (Jul. 2015)
https://medium.com/ionic-and-the-mobile-web/the-web-is-too-low-level-7a4ea4933366#.hy96a7eco

[8] Mobile JavaScript Apps: The Problem with the Mobile Web (Jun 2016)
http://thefullstack.xyz/mobile-javascript-apps/

[9] Mobile JavaScript Apps: The Dawn of React Native (Jul 2016)
http://thefullstack.xyz/react-native/

[10] Kein Buzzword-Bingo: Was sind Progressive Web Apps?
http://t3n.de/news/progressive-web-apps-739224/

[11] Elbowing JavaScript out (Apr. 2016)
http://blog.ikura.co/posts/elbowing-js-out.html

[12] Native Apps are Doomed
https://medium.com/javascript-scene/native-apps-are-doomed-ac397148a2c0#.uqnjvrqpv

[13] Why Native Apps Really are Doomed: Native Apps are Doomed pt 2
https://medium.com/javascript-scene/why-native-apps-really-are-doomed-native-apps-are-doomed-pt-2-e035b43170e9#.m41erm41a