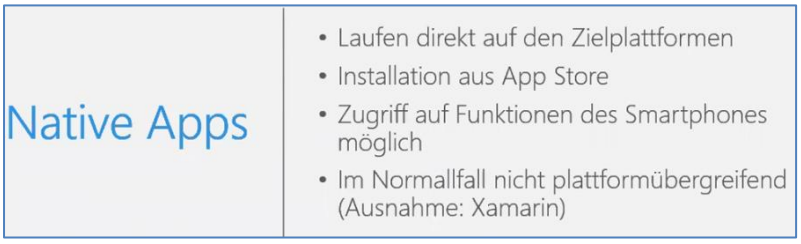
Mobile Web-Apps – Teil 2

Im Grunde unterscheidet man zwischen vier Typen von Apps

a) Web-Apps

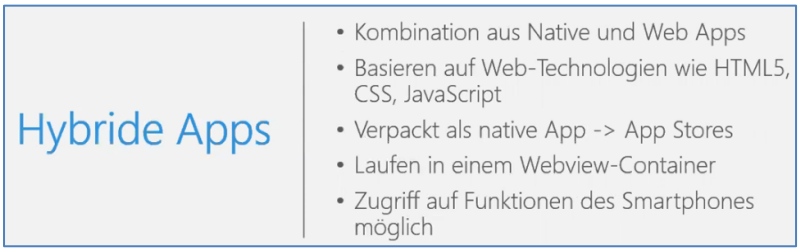
Beispiele: Angular, React, Vue

b) Native Apps



Beispiele: Java (Google), Kotlin, Swift (ioS)

c) Hybrid Apps



Beispiele: Ionic5

d) Cross-kompilierte mobile App

- ReactNative
- NativeScript
- Flutter
- Xamarin

1. Web-Apps (Progressive Web Apps) =/= Mobile Web

Eine Web-App ist eine Web-Applikation, die vom Aussehen her wie eine native Applikation aus den Stores daherkommt und vollständig auf dem mobilen Gerät bzw. Device läuft. Die Hauptverarbeitung läuft auf dem Client und nicht auf dem Server! Mit HTML5, CSS und JS und dessen APIs hat der Entwickler viele Möglichkeiten, auf das Device zuzugreifen – aber nicht dieselben wie bei einer nativen App. So fehlt z.B. der Zugriff auf das Adressbuch. Der Vorteil ist, dass mithilfe einer Web-App viele Plattformen erreicht werden können. Für viele Unternehmen ist auch die vereinfachte Distribution ein Argument.

Sie werden NICHT in App Stores verbreitet, sondern direkt mittels URL. Natürlich sind sie responsive und passen sich an die Smartphones perfekt an.

Man spricht auch von „progressiven Web Apps“, wie z.B.

- **Angular** (Entwickler = Google) – ist eine Plattform („platform“), sehr vieles bereits eingebaut
- **React** (Entwickler = Facebook) – ist eher eine „library“, eher minimalistisch
- **Vue.js =** ein „framework“, fokussiert auf Code

https://www.youtube.com/watch?v=lYWYWyX04JI

Aus diesen Web-Apps kann man mit zusätzlichen Programmen sogar **NATIVE APPS erstellen**. Dafür kann man folgende Entwicklungen verwenden:

- Angular + NativeScript
- React + ReactNative
- Vue + NativeScript

2. Native Apps

Dabei wird eine mobile App exakt auf ein konkretes Betriebssystem wie z.B. iOS oder Android zugeschnitten und entwickelt. Das bedeutet, dass der Programmcode **nicht 1:1 für eine andere Plattform** verwendet werden kann.

Will man als App-Entwickler in den **Stores wie bei Android und Apple** die App anbieten ist dies schon aufwendig. Apple z.B. verlangt eine Jahresgebühr für das Entwicklerprogramm usw. Dazu kommt, dass z.B. Apple selbst entscheidet, ob die App zugelassen wird.

Vorteile nativer Apps:

- Hohe Performance
- Zugriffsmöglichkeit auf die Hardware eines Gerätes (z.B. Kamera, Mikrofon)

Der **Nachteil** ist, dass diese Apps für jede Plattform fast komplett neu entwickelt werden müssen. Auf die unterschiedlichen Bedienungskonzepte muss unbedingt eingegangen werden, damit die Benutzer die Software annehmen bzw. kaufen.

Beispiele für die **Entwicklungsumgebungen** sind

- Xcode und
- Android Studio

Native Sprache: Java (Android) und Swift (ioS). Jedoch zunehmend Kotlin statt Java

Kotlin löst Java als Programmiersprache für Android-Apps ab. Sie glänzt mit kurzen und verständlichen Sprach-Konstrukten, die Arbeit sparen und die Übersicht verbessern. Erfinder ist JetBrains. Google empfiehlt Kotlin für alle neuen Android-Apps. Code Beispiele findet man unter ct.de/yumk.

3. Hybrid-Apps oder Container-App

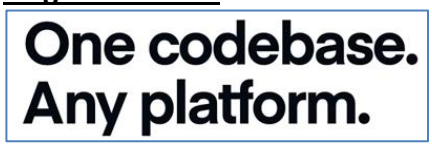
Um die Vorteile der Web-Apps mit den Vorteilen der nativen Apps zu kombinieren, kann eine Web-App in einem **Container (WebView)** einer nativen App ablaufen und können über Plugins auf native Funktionen und Komponenten des mobilen Endgerätes zugreifen. Das bedeutet, dass die App wie eine native App daherkommt, im Inneren aber weiterhin eine HTML5-App ist. Mit den erwähnten Plugins (= zusätzlichen JavaScript-APIs) hat der Entwickler trotzdem den kompletten Zugriff auf die nativen Möglichkeiten der Devices – wobei gewisse proprietäre Aspekte in Kauf genommen werden.

Dabei wird die App zunächst **betriebssystemunabhängig** entwickelt. Anschließend wird sie mit nativen Funktionen kombiniert und erweitert, um gerätespezifische Eigenschaften wie z.B. Sensoren und Kamera oder die Kontakte nutzen zu können. Sie können über einen App-Store veröffentlicht und installiert werden, ohne dass mehrere parallele Entwicklungsstränge existieren müssen.

Typische Anwendung:

- **Ionic** – (aktuell Sept. 2021 = ionic5) ist ein Open-Source-Framework, mit dessen Hilfe grafische Benutzeroberflächen unter dem Einsatz von HTML5, CSS3 und JavaScript entwickelt werden. Es basiert auf Angular von Google, einem JavaScript-Framework. Mobile Ionic-Apps sind auf mobilen Endgeräten und mobilen Betriebssystemen mit Webbrowser bzw. WebView lauffähig.

Slogan von Ionic:



alleine oder + Zusammenarbeit mit 

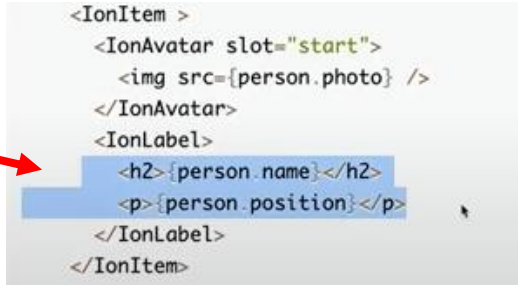
Ionic kann selbständig WebApps erzeugen. Aber es kann mit Angular, React und Vue zusammenarbeiten, um umfangreichere realistische Apps zu gestalten.

Addiert man „**Capacitor**“ dann kann man **native mobile Apps erstellen**. Capacitor „wrappt“ den WebApp-Code und macht daraus ein „Wrapped WebApp“.

Mit Ionic nutzt man auch normales HTML:
Beispiel siehe hier:

https://www.youtube.com/watch?v=I6mCTKst9II

https://www.youtube.com/watch?v=_03VKmdrxV8



4. Cross-kompilierte mobile App

Die Grundidee ist es, **native mobile Apps** für die unterschiedlichen mobilen Betriebssysteme auf Grundlage **EINER Quellcodebasis** zu generieren. Dazu werden Cross-Compiler eingesetzt, die den Quellcode ohne weitere Zwischenschicht für unterschiedliche Betriebssysteme cross-kompilieren, sodass eine native mobile App entsteht.

Entwicklungswerkzeuge :

- **Xamarin -** ist eine Softwareanwendung, die in C# programmiert ist. Mithilfe der IDE vom Xamarin Studio, die auch in Microsoft Visual Studio integriert ist, können native mobile Apps entwickelt werden.
- **React Native** - wurde 2015 von Facebook veröffentlicht und nutzt das JavaScript-Framework React zur Entwicklung. Code hat nichts mit HTML zu tun.
- **NativeScript -** welches alleine (reines JavaScript) oder zur Unterstützung mit Angular oder Vue arbeiten kann
- **Flutter** - von Google 2017 vorgestellt, im Mai 2020 erste stabile Form, Programmiersprache: Dart

Der Code von JavaScript, dieser Programme, wird nicht direkt zu native „swift“ oder „java“ - Code kompilliert (compiled), sondern wird in einer virtuelle Maschine des Apps gehostet und nicht direkt kompilliert.

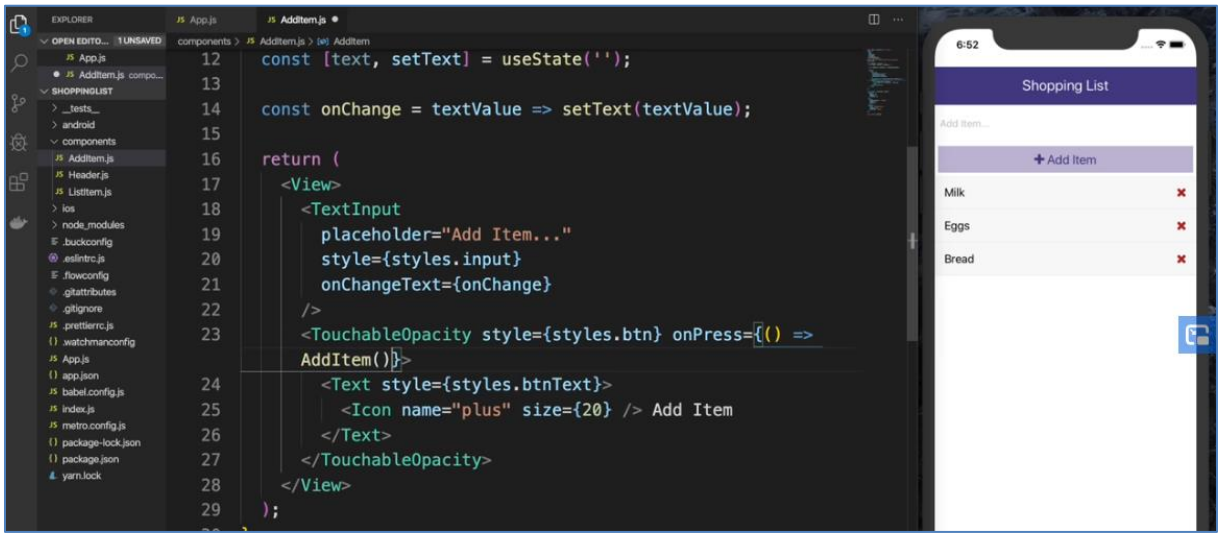
https://www.youtube.com/watch?v=hC-7tRy7GQs

https://www.youtube.com/watch?v=4m7msadL5iA

NativeScript is...

an open source framework for building truly native mobile apps with JavaScript. Use web skills, like Angular, Vue.js, and CSS, and get native UI and performance on iOS and Android.

Info: React Native verwendet keine HTML Codes:



https://www.youtube.com/watch?v=PKRXbLnfXXk

https://www.youtube.com/watch?v=lYWYWyX04JI

Allgemeines:

Web-Apps

Das sind mobile Webanwendungen, die in der Regel auf Basis von HTML5, CSS und JavaScript entwickelt werden. Diese Web-Apps laufen vollständig in einem Webbrowser wie z.B. Safari für iOS oder Google Chrome für Android bzw. in einer bildschirmfüllenden Web-View und somit auf beliebigen mobilen Endgeräten. Dadurch ist eine Ausrichtung auf ein bestimmtes Endgerät nicht zwingend notwendig. Man schreibt den Code ein einziges Mal und er funktioniert auf sämtlichen Endgeräten. Der gemeinsame Nenner ist stets HTML5.

Größter Vorteil ist die Plattform übergreifende Nutzbarkeit.

- Vorteile: reine Web-Apps werden in der Regel nicht über einen App Store vertrieben. Zudem können sie ohne Installation genutzt werden.
- Nachteile: Es sollte eine möglichst breitbandige Netzverbindung bestehen. Meist sind sie nicht in der Lage auf sämtliche Hardwarekomponenten eines mobilen Endgerätes zuzugreifen. Da sie nicht über einen App Store angeboten werden, unterliegen sie auch nicht der hierbei durchgeführten Qualitätskontrolle.

Mobile Web-Apps beruhen wie mobile Webseiten auf Webtechniken. Optisch erinnern sie aber an native Apps und sie zeichnen sich außerdem durch den intensiven Gebrauch von JavaScript und JavaScript-APIs aus.

Zur Abgrenzung von mobiler Web-App gegenüber mobiler Webseite werden verschiedene Kriterien herangezogen:

1)Optik – Look & Feel

Eine mobile Web-App vermittelt eine App-like Erfahrung. Das heißt, dass sie vom Benutzer ähnlich wie eine native App wahrgenommen wird. Typisch ist also für Web-Apps ein bestimmtes Aussehen. Üblicherweise orientieren sich Web-Apps optisch mehr an den nativen Apps als an der Desktop-Seite.
Typisch sind folgende Komponenten (im Unterschied zu mobilen Webseiten):

- Fixierte Toolbar oben, eine Reihe von Buttons unten.
- Animierte Seitenübergänge bei der Navigation zwischen den verschiedenen Bereichen.
- Typisch sind auch Elemente wie ActionSheet, das sind Meldungsfenster, die mit zusätzlichen Optionen von unten auf dem Bildschirm erscheinen, oder Ladeanzeiger.

2)Funktionalität

Webseiten besucht man, Web-Apps nutzt man. Die Interaktion bei Webseiten ist Lesen, bei Web-Apps ist es mehr. Es gibt natürlich Ausnahmen von diesem Prinzip und im Allgemeinen

geht die Tendenz im Web immer mehr in Richtung „Anwendungen“ und weg von den „reinen Webseiten“ (nicht zufällig dient HTML5 der Erstellung von Anwendungen). Deswegen hilft das Kriterium der besonderen Funktionalität bei Web-Apps, kann aber nicht zur eindeutigen Abgrenzung dienen.

3)Architektur

Beim klassischen Web ist der Browser ein Thin Client, und der Webserver erledigt die Hauptarbeit. Bei Web-Apps haben wir es hingegen mit einem Fat Client zu tun, Hier macht der Browser wesentlich mehr; involviert sind Techniken wie AppCache, mit der die Webanwendung offline-fähig wird, WebStorage zur Datenspeicherung und vieles mehr – es wird also das daraus, was man treffend als Rich Internet Application bezeichnet.

4)Reichweite

Apps laufen auf einer begrenzten Anzahl an Geräten, weil sie bestimmte Techniken voraussetzen. Eine Webseite kann nicht so viel leisten, weil sie für möglichst viele funktionieren muss.

Warum JavaScript?

JavaScript erlebt gerade einen regelrechten Aufschwung. In der Vergangenheit wurde die Sprache etwas belächelt, weil sie im Wesentlichen zum „Aufhübschen“ von Webseiten verwendet, aber nicht als „echte“ Sprache für professionelle Anwendungsentwicklung gesehen wurde. Mit der enormen Ausbreitungsgeschwindigkeit von HTML5 dreht sich dieser Trend: JavaScript ist auf dem Weg, sich als Sprache zur Anwendungsentwicklung und Oberflächenprogrammierung zu etablieren.

Single Page Web-Apps

Eine spezielle Ausprägung sind „Single Page Apps“.
Eine Single Page App (SPA) ist eine Webanwendung, die keinen Seitenwechsel (Refresh) durchführt, sondern die Oberfläche über dynamischen Austausch der HTML-Elemente mit JavaScript ändert.
Das Konzept verdankt seinen Namen der Tatsache, dass eine solche Anwendung nur eine einzige HTML-Seite benötigt, nämlich die, über die der JavaScript-Code in den Browser geladen wird. Die Anwendung nutzt Ajax-Mechanismen, um mit dem Server zu kommunizieren.
Der Ansatz ist aktuell sehr populär, weil sich damit Anwendungen realisieren lassen, die zum einen für den Benutzer auf den ersten Blick sehr ergonomisch sind, zum anderen bewährte und bekannte Muster für die Entwicklung unterstützen. Bekannte Vertreter des Ansatzes sind unter anderem die JavaScript-Frameworks „AngularJS“ und „Ember“.

(Quelle: Tilkov in: Rest und http, 2015, dpunkt-Verlag, Heidelberg, S. 264)

Bei Web-Apps wird der Client intelligenter, da man nicht alles auf den Server verlagert.

Bei **Single Page Web-Apps wird beim Laden einer HTML-Seite gleich die ganze Web-App geladen. Das ist eine einzige HTML-Seite, von der aber nur ein Teil angezeigt wird**.

Im Ergebnis handelt es sich bei Single Page Web-Apps also mehr um Clientapplikationen im Sinne der Client-Server-Programmierung als um Webapplikationen im Sinne von serverseitig generierten Webseiten. Mobile Webapplikationen, die auf diese Weise entwickelt wurden, integrieren sich hervorragend in Gesamtarchitekturen, bei denen das Backend über ein RESTful API angesprochen werden kann. Optimal für das Zusammenspiel ist eine Kommunikation auf Basis der JavaScript Object Notation (JSON). Weitere Vorteile dieser Applikationsarchitektur sind die potenziellen Offlinefähigkeit der Clients und die Möglichkeit, diese Applikationen mit Werkzeugen wie beispielsweise PhoneGap in native Applikationen zu „verpacken“, die dann einerseits vollen Zugriff auf die Hardwarefunktionalitäten der Ausführungsumgebung haben, als auch wie native Apps durch die entsprechenden Distributionskanäle der Plattformen („App Stores“) verteilt werden können.

Interessante Links:
http://grochtdreis.de/

http://grochtdreis.de/weblog/2012/03/20/am-ende-ist-doch-alles-html-2/

http://www.responsive-webdesign-praxis.de/

Quellen:

Florence Maurice, in: Mobile Webseiten, Strategien, Techniken, DOS und DON´TS für Webentwickler, Hanser Verlag München, 2012

Peter Gasston, in: Moderne Webentwicklung, Geräteunabhängige Entwicklung, Techniken und Trends in HTML5, CSS3 und JavaScript, 2013, dpunkt Verlag Heidelberg

Andrea Ertel und Kai Laborenz, in: Responsive Webdesign, Anpassungsfähige Websites programmieren und gestalten, 2015, Galileo Press, Bonn

Hussein Morsy, in: Adobe Dreamweaver CS6, 2012, Galileo Design, S. 330-352