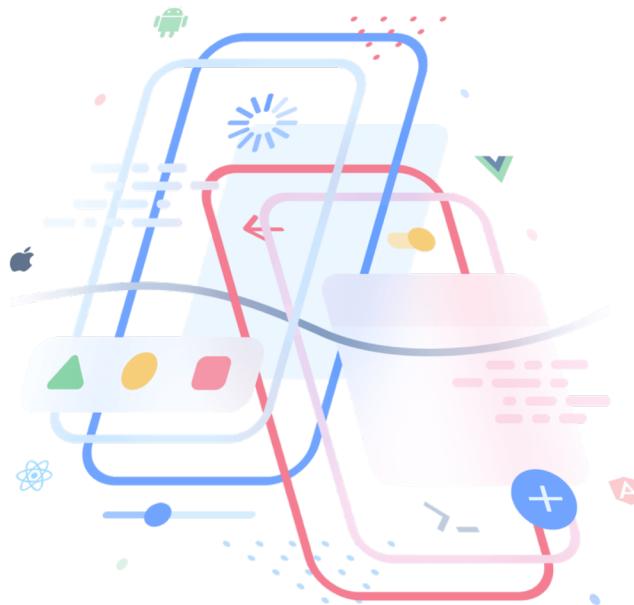


# Moderne Web-Apps mit Ionic

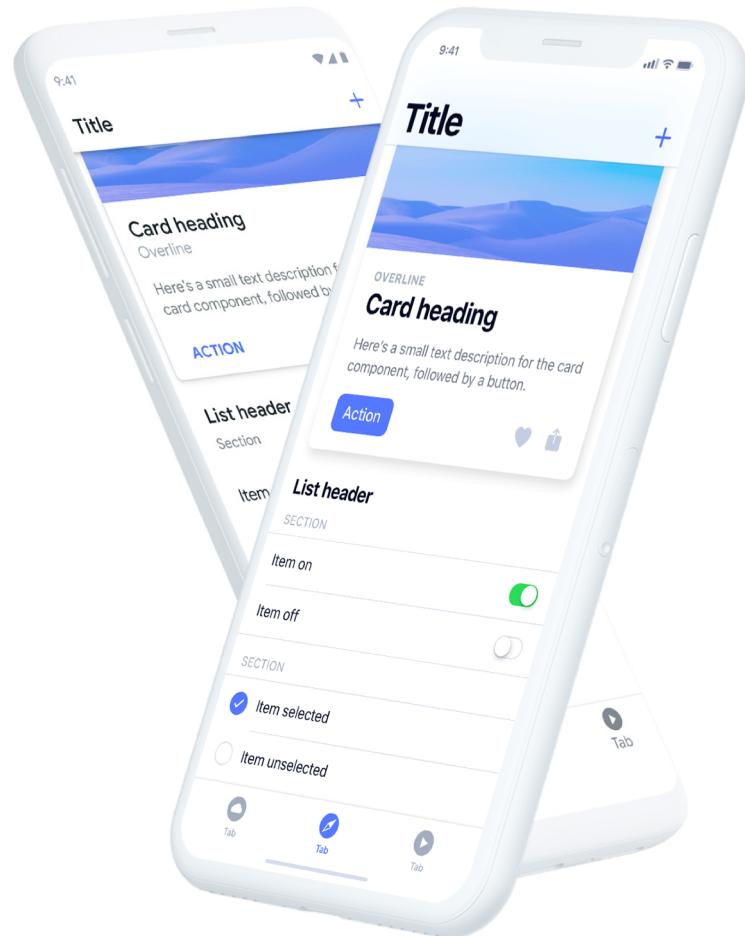
Norbert D. Frank



# Was ist Ionic

- Mobile UI-Toolkit
- Framework-agnostische UI-Komponenten
- Integration in moderne JS-Frameworks (Angular, React, Vue.js)
- iOS, Android, Desktop & Web → eine Code-Basis mit Web-Technologien (HTML, CSS, JS)
- „Wordpress for mobile apps“

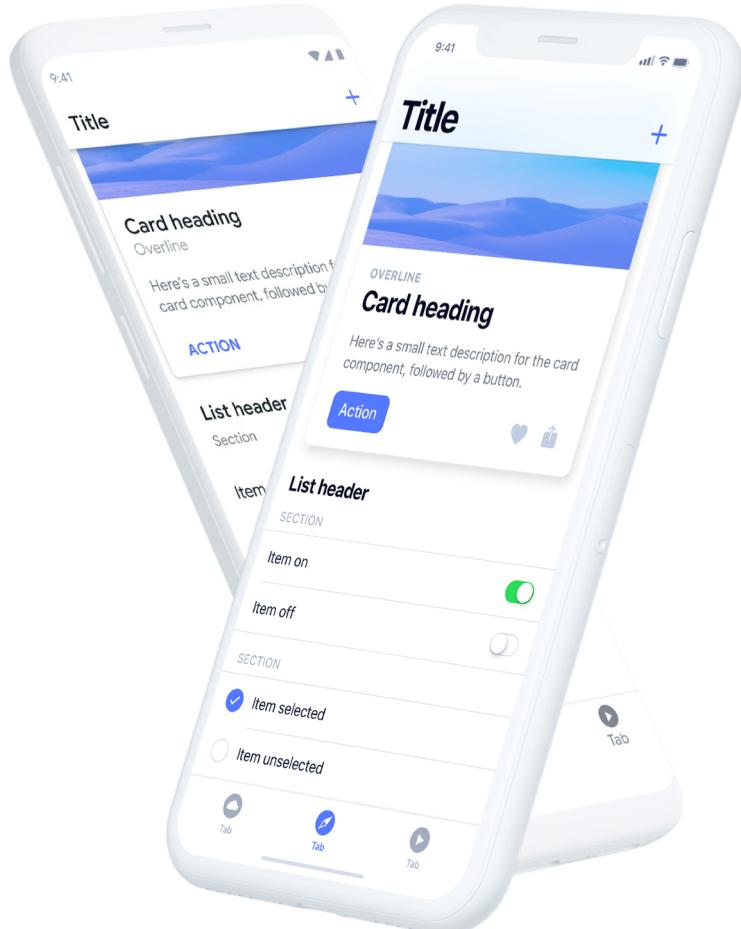
*Ionic bietet ein umfangreiches Ökosystem, um Anwendungen für alle Plattformen auf Basis von Webtechnologien zu entwickeln*



# Wofür ist Ionic geeignet

Ganz einfach gesagt: alle Web-Anwendungen bis auf:

1. Projekte, für die die UI-Komponenten nicht geeignet sind (besondere Anforderungen, exotische Layouts)
2. Sehr hohe Performance-Anforderungen (für die meisten Anwendungsfälle ist Ionic aber vergleichbar schnell zu nativen Anwendungen)



# Hybrid vs. Native

## Einschätzung 2020

- Die Diskussion wird schon seit vielen Jahren z.T. sehr leidenschaftlich geführt.
- Neue Abstraktionsebenen zu schaffen gehört zur Geschichte der IT. Das ist üblicherweise mit vielen Vorteilen verbunden, aber man erkauft sich auch leichte Nachteile.
- Hybrid vs. Native ist im Kern eine Kosten-Nutzenabwägung. Schätzungen zufolge ersparen hybride Anwendungen 75-80% an Aufwand im Vergleich zu nativen Anwendungen.
- Hybrid 2020 ist nicht vergleichbar mit hybrid 2014. Das Web ist weit gekommen.
- Visual Studio Code ist eine Web-App!

# Hybrid vs. Native

- Hybride Anwendungen starten etwas langsamer (Kaltstart), sind anschließend typischerweise von nativen Anwendungen kaum zu unterscheiden.
- Die Geschwindigkeit der meisten Anwendungen hängt eher von der Netzwerkgeschwindigkeit, als von hybrid vs nativ ab.
- Das Web war noch nie so gut wie heute. Den gleichen Technologiestack auf allen Geräten zu nutzen, bringt enorme Vorteile mit sich.
- Alternativen für Cross-Platform-Apps:
  - React Native / NativeScript (JavaScript)
  - Flutter (Dart)
  - Xamarin (C#)

# Ionic UI-Komponenten

*Low Level /  
Base Utilities*

*High Level /  
Ready to use*



# Was bisher geschah

Ionic-Historie und meine eigene Geschichte

- Developer, Consultant, IT-Architekt @ Lucom GmbH
- JS-Enthusiast, Frontend und Web im allgemeinen
- @norbertdfrank

**LUCOM**



[www.lucom.com](http://www.lucom.com)

# Ein Blick zurück



Ionic wird gegründet

11.2013: erstes Alpha-Release

05.2015: Version 1.0 basierend  
auf AngularJS 1

2012

2013

2014

2015

2016

N. Frank

Erste Schritte mobile App-Development

„Mehr vom Geld“-App veröffentlicht

Mehrere Upgrades „Mehr vom Geld“

Erste produktive Ionic-App

# Ein Blick zurück



2016: RC-Phase Angular 2, Ionic 2 Beta

01.2017: Ionic 2 (Angular 2)  
04.2017: Ionic 3 (Angular 4)

01.2019: Ionic 4

05/06.2019: Capacitor 1.0,  
Stencil 1.0

01.2020: Ionic 5 RC



- N. Frank
- Mehrere Ionic-Apps entwickelt
  - Kompletter Rewrite von „Mehr vom Geld“ mit Ionic 2/3
  - Umbenennung zu „Finfluence“ und Veröffentlichung von 1.0
  - Upgrade auf Ionic 4
  - Desktop-App, ggf. PWA



# Neu in V4: Framework-agnostische Web Components

Was ist das eigentlich?

- Alle UI-Komponenten sind als Web Components umgesetzt
- Ionic kann daher ab Version 4 mit jedem JS-Framework oder ganz ohne eingesetzt werden
- Die mitgelieferten Integrationen in Angular, React und Vue.js kapseln die Web Components mit den Mitteln des jeweiligen Frameworks

## Ionic einsetzen

In das gewünschte Framework integrieren

# Eine Ionic-App anlegen

Option 1: ohne Framework

Ionic lässt sich als Library einfach einbinden durch:

```
<link href="https://unpkg.com/@ionic/core@4.4.2/css/ionic.bundle.css"  
      rel="stylesheet">  
src="https://unpkg.com/@ionic/core@4.4.2/dist/ionic.js"></script>
```

Mehr Info: <https://github.com/ionic-team/ionic/blob/master/core/README.md>

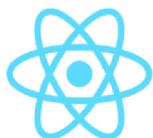
# Eine Ionic-App anlegen

## Option 2: mit Framework-Integration

Ionic bietet derzeit drei Framework-Integrationen:



Produktionsreif, umfangreichste Integration



Produktionsreif



Integration im Beta-Status



# Eine Ionic-App anlegen

Angular

Eine Angular-App mit Ionic lässt sich per Ionic CLI anlegen:



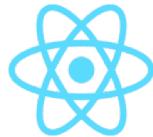
```
$ ionic start <name> <template> --type=angular
```

Auch eine Ionic 3 App wird noch unterstützt:



```
$ ionic start <name> <template> --type=ionic-angular
```

# Eine Ionic-App anlegen



React

Eine React-App mit Ionic lässt sich per Ionic CLI (ab Version 5.0.0) anlegen:



```
$ ionic start <name> <template> --type=react
```

# Eine Ionic-App anlegen



## Vue.js

Eine Vue-App lässt sich nicht per Ionic-CLI anlegen. Stattdessen legt man eine Vue-App per Vue-CLI an und integriert Ionic als Plugin:

```
$ npm install @ionic/vue @ionic/core --save
```

# Eine Ionic-App anlegen



## Vue.js

Initialisierung in *main.js*:

```
import Vue from "vue";
import App from "./App.vue";
import Ionic from "@ionic/vue";
import router from "./router";
import store from "./store";

Vue.use(Ionic);
new Vue({
  router,
  store,
  render: h => h(App)
}).$mount("#app");
```

# Die UI-Components

Building-Blocks der App

# Die UI-Components

Ionic besteht aus ca. 50 UI-Components. Ein grober Überblick:

- **Layout:** Listen, Cards, Grid, Image, Text, Avatar, Chips, Badges
- **Seitenstruktur:** Menü, Toolbar, Tabs, Seitenmenü, Segmente, Floating Action Buttons, modale Dialoge, Slides
- **Navigation:** Routing-Outlets, Nav-Outlets
- **Aktivitätsanzeige:** Spinner, Progressbar, Skeleton-Text, Toast
- **Interaktion:** Popups, Alert-Sheets, Pull-To-Refresh, Slide-Menü, Reorder, Infinite Scroll, Searchbar
- **Formulare:** Inputs, Datepicker, Selects, Radio-Groups, Range
- **Buttons, Icons (Ionicons)**

# Die UI-Components

- Die Möglichkeiten der UI-Components sind gut dokumentiert wenn man die grundsätzlichen Konzepte verstanden hat
- Die Optionen der meisten Components unterteilen sich in folgende Bereiche:
  - Properties
  - Slots
  - Methods
  - Events
  - CSS Custom Properties

# Exkurs: Shadow Dom, Slots & CSS Properties

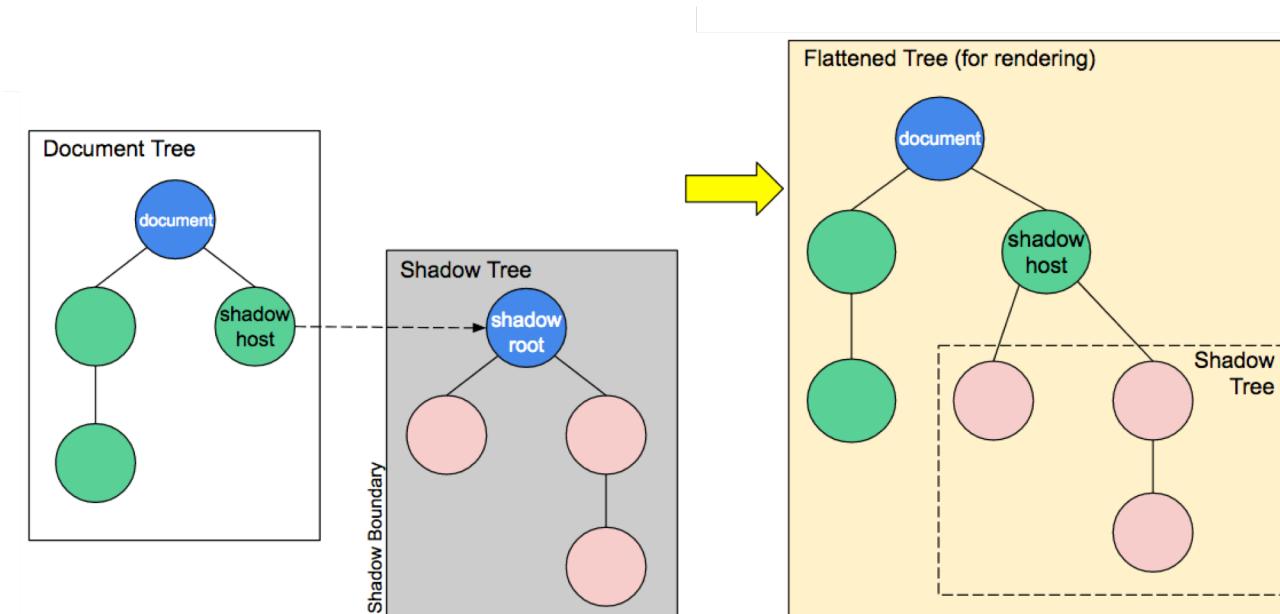
Die Components sind in Ionic 4 als Web Components umgesetzt. Sie lassen sich überwiegend genauso einsetzen, wie in früheren Releases, aber folgende Konzepte sind wichtig:

1. Shadow Dom
2. Slots
3. CSS Custom Properties

# Exkurs: Shadow Dom, Slots & CSS Properties

## Shadow DOM

Die Inhalte der Components werden außerhalb des normalen DOM (gelegentlich auch als „light DOM“ bezeichnet) gerendert und sind in einem sogenannten „Shadow DOM“ gekapselt.



# Exkurs: Shadow Dom, Slots & CSS Properties

## Slots

Slots sind Platzhalter in Web Components, die mit Inhalten aus dem Light DOM gefüllt werden können. Technisch bleiben die Nodes im Light DOM, werden aber an die vorgesehene Stelle im Shadow DOM „projiziert“.

Ionic macht häufigen Gebrauch von Slots.

**Zu beachten ist:** die interne Darstellung der Web Components lässt sich nicht per CSS-Selektoren stylen (außer dem Shadow-Host). Dies gilt aber nicht für den Inhalt von Slots, da diese nicht zum Shadow DOM gehören.

# Exkurs: Shadow Dom, Slots & CSS Properties

## CSS Custom Properties

Die Nodes im Shadow DOM lassen sich von außen nicht per CSS stylen. Um trotzdem Einfluss auf das Design nehmen zu können, werden CSS Custom Properties eingesetzt.

CSS Custom Properties werden auch als „CSS Variablen“ bezeichnet und sind grob vergleichbar mit Variablen in Pre-Compilern wie SASS mit dem Unterschied, dass sie zur Laufzeit existieren.

Sie werden definiert wie normale CSS-Properties, aber mit vorangestelltem doppelten Minus:

```
--my-background: #dddddd;
```

Zugriff über die `var()`-Funktion:

```
background-color: var(--my-background);
```

Ionic-Components erlauben es über diverse CSS Custom Properties, Einfluss auf das Design der Komponente zu nehmen.

# Demo

# Native Funktionen nutzen

Kamera, Dateisystem & mehr

# Native Features

Typische Beispiele für native Funktionen:

- Dateisystem
- Location
- Kamera
- Sensoren
- Native Dienste: Notifications, Kontakte, Datenbanken..

# Native Features

Folgende vier Ablaufumgebungen sind für Ionic relevant:

- Web/PWA
- Desktop (Electron)
- iOS
- Android

# Option 1: Cordova/Phonegap



- Cordova gibt es seit ca. 10 Jahren und wird seit einigen Jahren von der Apache Software Foundation entwickelt
- Cordova hat ein umfangreiches Plugin-Ökosystem
- Unterstütze Plattformen:
  - iOS
  - Android
  - Windows 8.1, Windows 8.1 Phone, Windows 10
  - OS X
  - Electron

# Option 1: Cordova/Phonegap

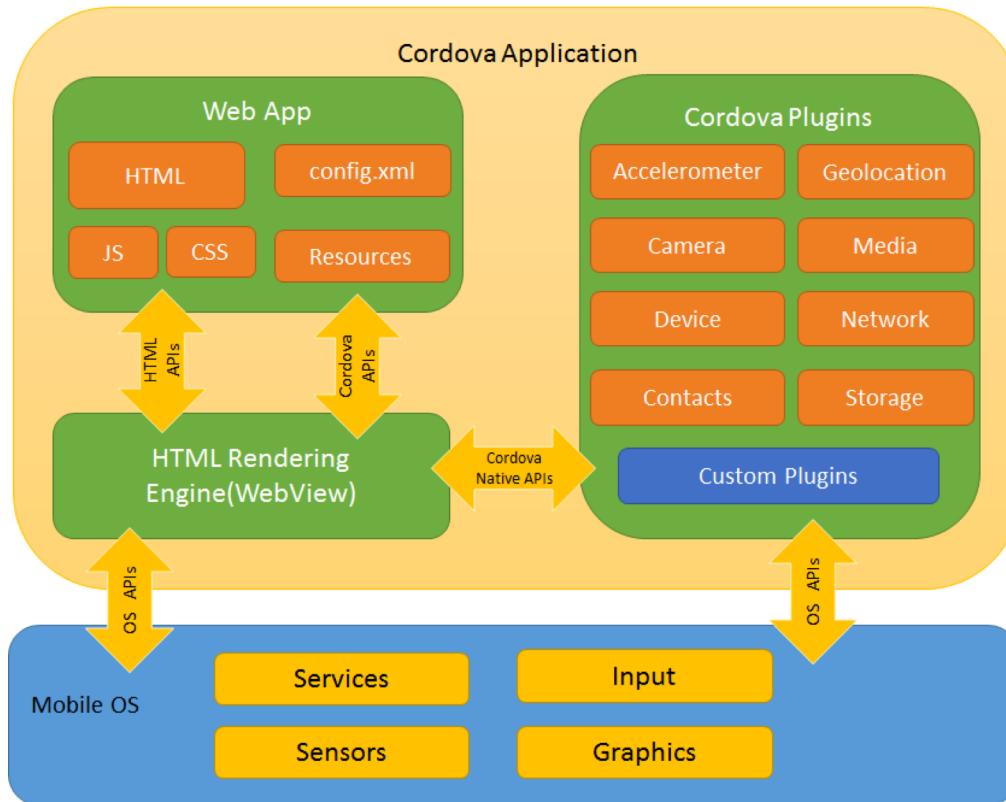


Technische Informationen:

- Cordova-Projekte werden über eine zentrale Konfigurationsdatei (config.xml) gesteuert
- Die nativen Projekte werden aus der Konfiguration bei jeder Änderung neu erzeugt
- Die API vieler Plugins ist Callback-basiert
- Updates von Plattform und Plugins werden durch Löschen und neu Hinzufügen vorgenommen
- Die Plattformen und CLI haben unabhängige Release-Zyklen

# Option 1: Cordova/Phonegap

Architektur:



# Ionic Native

Ionic Native erleichtert die Verwendung von Cordova-Plugins deutlich:

- Ionic Native bietet Wrapper um viele Cordova-Plugins und überführt diese in eine einheitlichere Promise-basierte API
- TypeScript-Unterstützung
- Ionic Native kann auch ohne Ionic eingesetzt werden
- Es muss jeweils das Cordova Plugin und der Ionic Native Wrapper installiert werden
- Ionic Native Wrapper können Mocks für Plattformen (Browser z.B.) enthalten, wenn Features in der Umgebung nicht zur Verfügung stehen und zur Entwicklungszeit simuliert werden

# Option 2: Capacitor

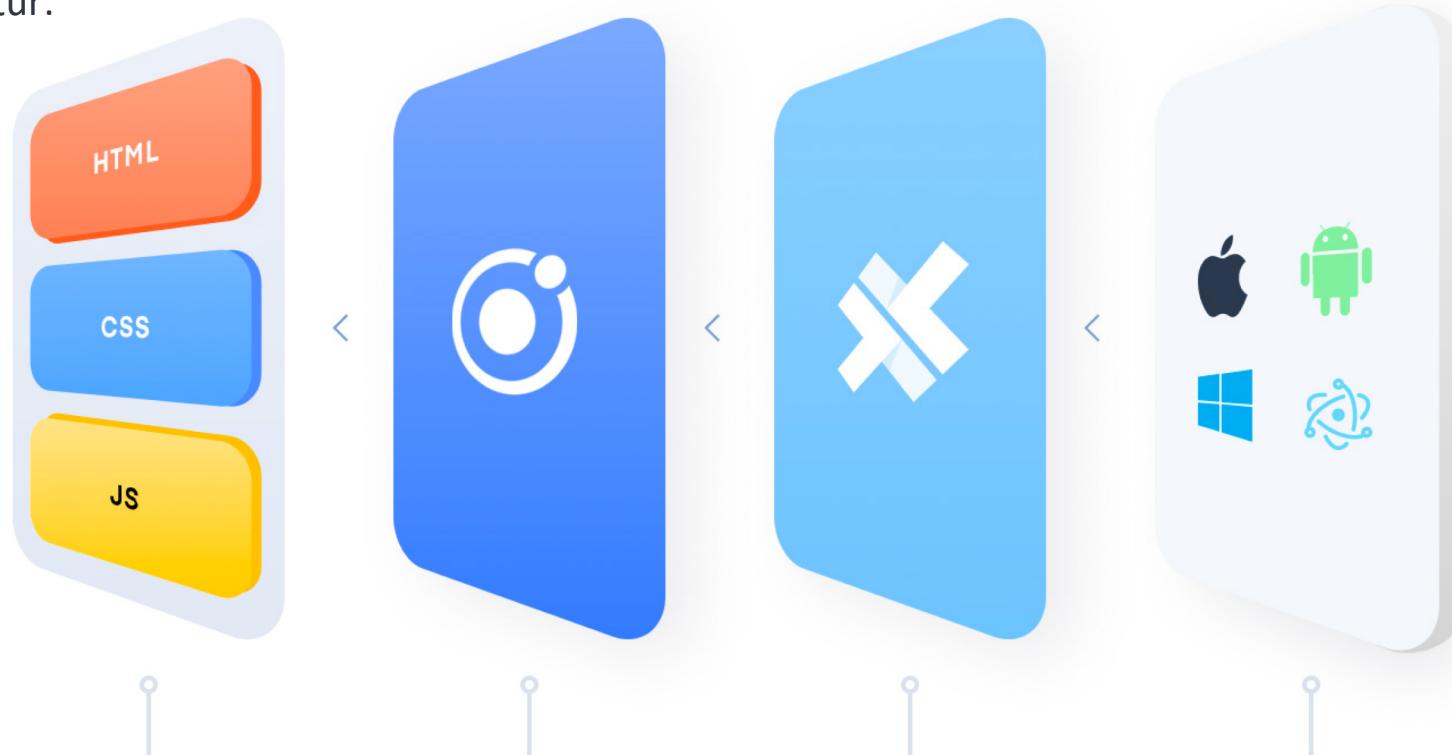


- Das Ionic-Team hat Capacitor als moderne Alternative zu Cordova entwickelt
- Konzeptionell sind die beiden Projekte sehr ähnlich, aber Capacitor konnte frisch mit dem heutigen Stand der Technik starten
- Unterstützte Plattformen:
  - Web / PWA
  - iOS
  - Android
  - Electron

# Option 2: Capacitor



Architektur:



YOUR APP (ANGULAR, REACT, VUE...)

UI CONTROLS (IONIC)

NATIVE ACCESS (CAPACITOR)

DISTRIBUTION PLATFORMS

# Cordova vs. Capacitor



Wesentliche Unterschiede von Capacitor zu Cordova:

- Die nativen Projekte (Android Studio, Xcode) werden nur einmalig angelegt und die weitere Konfiguration findet über die native IDE statt
- Kein *deviceReady* mehr! Alle Plugins stehen sofort zur Verfügung
- Eine umfangreiche API steht als Teil von @capacitor/core zur Verfügung und es müssen keine separaten Plugins installiert werden
- Plugins werden per NPM verwaltet
- Die meisten Cordova-Plugins funktionieren auch mit Capacitor – auch mit Ionic Native!

# Cordova vs. Capacitor



## Einschätzung:

Capacitor ist die moderne Interpretation des hybriden Web-Containers für native Apps. Die Core-API ist umfangreich und zusätzlich laufen die meisten Cordova-Plugins auch unter Capacitor.

Einen Performance-Vorteil bietet Capacitor nicht, da die Basisarchitektur im Kern gleich ist.

Der größte sichtbare Unterschied ist die Verwendung der nativen IDE als Teil des Entwicklungsprozesses. Dies ist für Einsteiger eine zusätzliche Hürde, aber für komplexe Projekte ein deutlicher Vorteil.

Meine Empfehlung: [ausprobieren!](#)

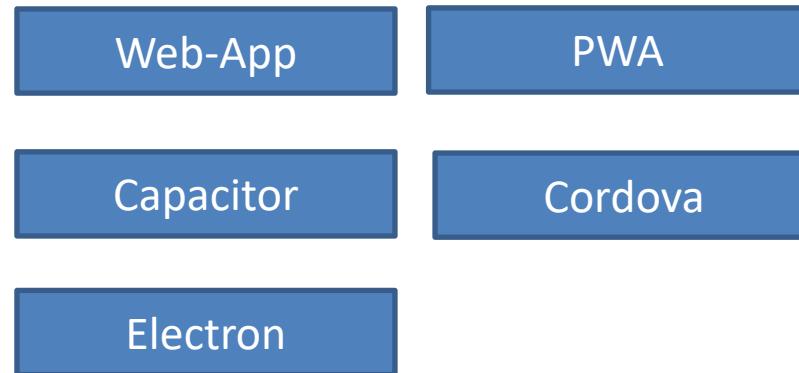
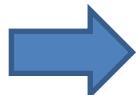
# Die App veröffentlichen

# Build-Step

Überblick über Deployment-Optionen:

Ionic hat keine eigenen Build-Scripte mehr (App-Scripts)! Stattdessen wird das Bundle mit den Tools des eingesetzten Frameworks erstellt. Für manche Befehle stellt das Ionic CLI einen Wrapper zur Verfügung.

So gibt es zwar noch den Befehl [ionic build](#), aber dies nutzt das Angular-CLI intern.



# Migration auf Ionic 4

Vieles unverändert, vieles verändert

# Migration

## Migration Ionic 3 nach Ionic 4:

- Nicht vergleichbar mit dem Upgrade von Ionic 1 nach 2, aber bei komplexen Apps nicht zu unterschätzen
- Wesentliche Änderungen:
  - Viele Markup-Änderungen in den Componenten
  - CSS-Variables anstatt SASS
  - Angular-Router
  - RxJS 6
  - Die API vieler Overlay-Components ist nun asynchron

# Migration

Tipps:

- Der offizielle Migrations-Guide ist sehr hilfreich:  
<https://ionicframework.com/docs/building/migration>
- TSLint-Migrationstool nutzen (mit Option --fix): <https://github.com/ionic-team/v4-migration-tslint>
- Entscheiden, ob `rxjs-compat` eingesetzt werden soll oder vollständig auf RxJS-6-Features umgestellt wird
- Doku zum Angular Router in Ionic lesen:  
<https://ionicframework.com/docs/angular/navigation>
- Genügend Zeit reservieren! Nachdem die Konzepte und Vorgehensweisen verinnerlicht sind, ist es reine Fleißarbeit.

# Stencil

One last thing...

# Was ist Stencil?



Ein mächtiger Compiler für Web Components (Build-Time-Abstraction)

- Von Ionic für die eigenen Komponenten entwickelt, aber unabhängig davon als Open Source verfügbar
- Besonders geeignet für die Erstellung von framework-unabhängigen Design Systems
- Stencil kann genutzt werden, um einzelne Komponenten zu entwickeln, aber auch, um ganze Applikationen zu entwickeln
- TypeScript, JSX, Virtual DOM, Routing, State Management, Doku-Generierung, unglaublich kleine Bundles (TodoMvc < 3kb!)
- Wichtig: wenn man eine Ionic-App mit einem Framework seiner Wahl entwickelt, hat man mit Stencil nicht direkt zu tun – es sei denn, man nutzt Stencil explizit für die eigene App

# Zu guter Letzt

## Fazit & Zusammenfassung

# Zusammenfassung



- Ionic bietet umfangreiche framework-unabhängig UI-Components
- Integrationen in Angular, React und Vue vorhanden
- Basierend auf modernen Webtechnologien
- Capacitor ist eine moderne Alternative zu Cordova und unterstützt die meisten Cordova-Plugins
- Ionic Native vereinfacht den Einsatz von Cordova-Plugins
- Der Web-Components-Compiler Stencil kann auch für eigene Komponenten oder Anwendungen verwendet werden
- Ionic 5 im RC-Status mit neuer Animations- & Gesture-API

# Zusammenfassung



- Artikel mit gleichem Schwerpunkt:
  - <https://www.heise.de/developer/artikel/Cross-Plattform-Entwicklung-mit-Ionic-4-Stencil-und-Capacitor-Teil-1-4572461.html>
  - <https://www.heise.de/developer/artikel/Cross-Plattform-Entwicklung-mit-Ionic-4-Stencil-und-Capacitor-Teil-2-4572495.html>

Vielen Dank – Das war's!

@norbertdfrank