

Mobile Apps - Verschiedene Ansätze im Vergleich

von Bernd Hort, 27.03.2019

EntwicklerCamp





Bernd Hort

- Diplom-Informatiker
- Lotus Notes Entwickler seit 3.3
- Java Entwickler
- Web Entwickler

Ich habe diese
geniale Idee für
eine App!



Wie bekomme ich meine App auf das Smartphone?



Klassische Webapp

PWA



Komplexität

Progressive Web Apps



Progressive Web Apps

- Progressive Web Apps erweitern Web Anwendungen um Fähigkeiten, die sie nativen Anwendungen ähnlicher werden lassen
 - Offline-Fähigkeit
 - Starten über den Homescreen
 - Kamerazugriff
 - Lokaler Speicher



Anforderungen an PWA

- **Reliable** - Sollen unabhängig von der Konnektivität funktionieren
- **Fast** - Sollen eine flüssige User Experience erlauben
- **Engaging** - Sollen sich wie eine native Anwendung anfühlen.
- **Safe** - HTTPS ist Pflicht



Web App Manifest

- Im „Web App Manifest“ werden grundlegende Eigenschaften festgelegt
- JSON-Datei wird eingebunden via

```
<link rel="manifest" href="/manifest.json">
```

- Definition auf der Google Developers Seite - [The Web App Manifest](#)
- Folien von der Session „Progressive Web Apps“ von Lars Migula



Web App Manifest

```
{
  "short_name": "Maps",
  "name": "Google Maps",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/images/icons-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/maps/?source=pwa",
  "background_color": "#3367D6",
  "display": "standalone",
  "scope": "/maps/",
  "theme_color": "#3367D6"
}
```



Service Worker

- Der Service Worker ist ein JavaScript Script, welches pro PWA im Hintergrund läuft und stellt die notwendigen Funktionalitäten für Progressive Web Apps zur Verfügung stellt.
- Initialisiert wird der Service Worker durch eine JavaScript-Datei, in der Hooks implementiert sind, die auf Ereignisse der Web App reagieren.



PWA in Angular

- Ab der Version 5.0 unterstützt Angular Progressive Web Apps direkt.
- Im bestehenden Angular-Projekten muss das Package @angular/pwa hinzugefügt werden

```
$ng add @angular/pwa --project <projectname>
```

- Legt manifest.json und ngsw-config.json an
- Erstellt Icons in unterschiedlichen Auflösungen



PWA in Angular

- manifest.json wird erstellt
- ngsw-config.json wird erstellt
- In angular.json werden Referenzen hinterlegt
- In index.html wird das Manifest eingebunden
- In app.module.js wird das ServiceWorkerModule eingebunden
- Erstellt Icons in unterschiedlichen Auflösungen



PWA in Angular

- Build der Angular-Anwendung mit

```
$ng build --prod
```



- Der integrierte Angular-Entwicklungsserver ist zum Testen nicht ausreichend
- Über node.js kann http-server installiert werden
- Starten mit

```
$http-server -p 8080 -c-1 dist/<project-name>
```



Anpassungen ngsw-config.json

- Per Default werden alle Anwendungs-Dateien gecacht
 - Eintrag „assetGroups“ in ngsw-config.json
- REST-API Aufrufe müssen manuell konfiguriert werden
 - Eintrag „dataGroups“
 - Definition von URL-Mustern
 - Unterscheidung „api-performance“ und „api-freshness“
 - Jeweils Konfiguration Cache-Verhalten



Strategie „Performance“ vs. „Freshness“

- Die Cache-Strategie „Performance“ holt die Daten so lange **zuerst aus dem Cache** bis das Ablaufdatum erreicht ist.
- Die Cache-Strategie „Freshness“ versucht immer erst **zuerst** die „frisch“ **vom Server** zu holen. Erst wenn der Timeout überschritten ist, werden Daten aus dem Cache geladen.



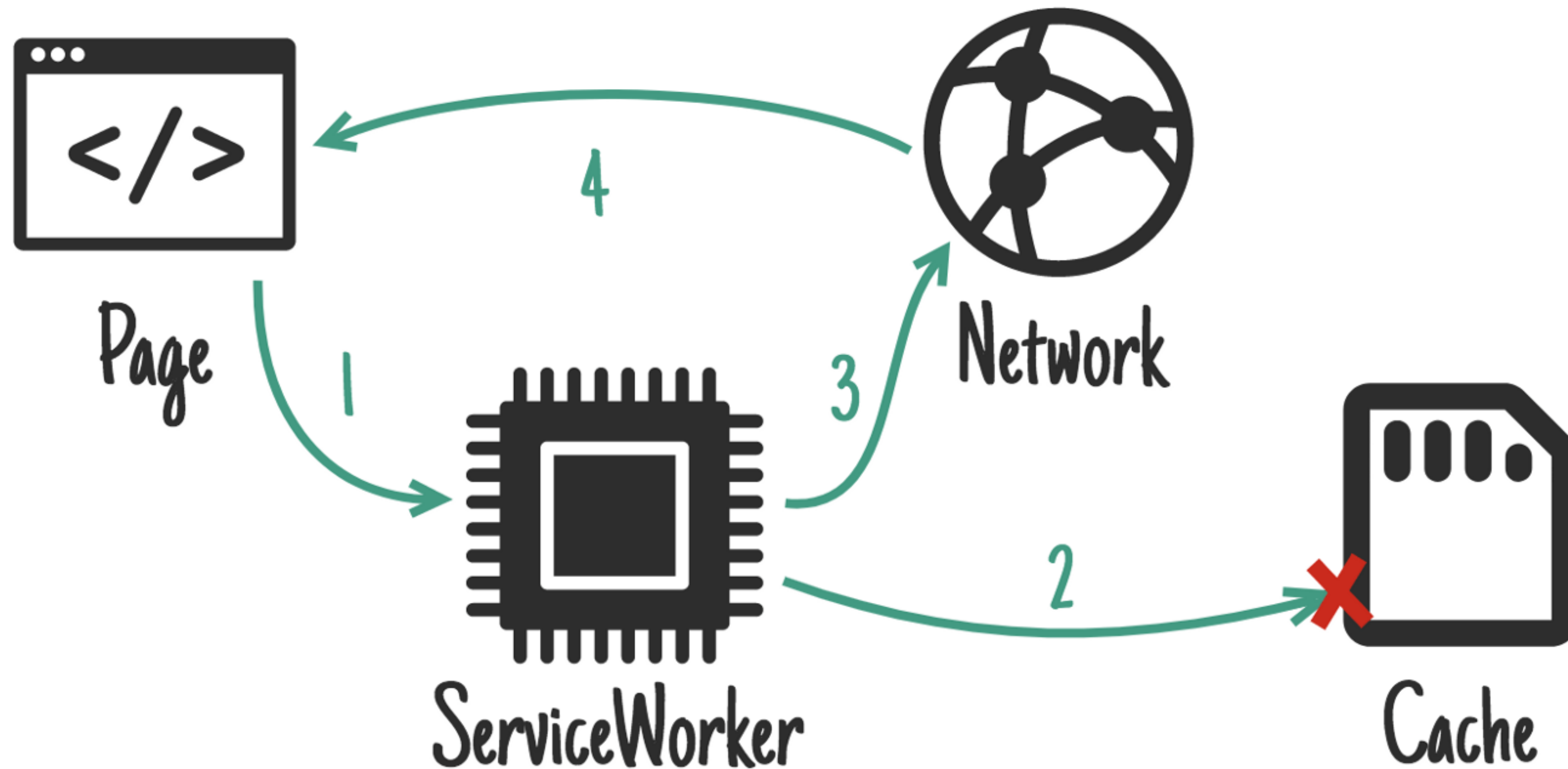
Strategie „Performance“ vs. „Freshness“

```
{
  "dataGroups": [
    {
      "name": "api-performance",
      "urls": [
        "/assets/i18n/**",
        "/api/**"
      ],
      "cacheConfig": {
        "strategy": "performance",
        "maxSize": 100,
        "maxAge": "3d"
      }
    },
    {
      "name": "api-freshness",
      "urls": [
        "/api/fresh-todo-list"
      ],
      "cacheConfig": {
        "strategy": "freshness",
        "maxSize": 100,
        "maxAge": "3d",
        "timeout": "10s"
      }
    }
  ]
}
```



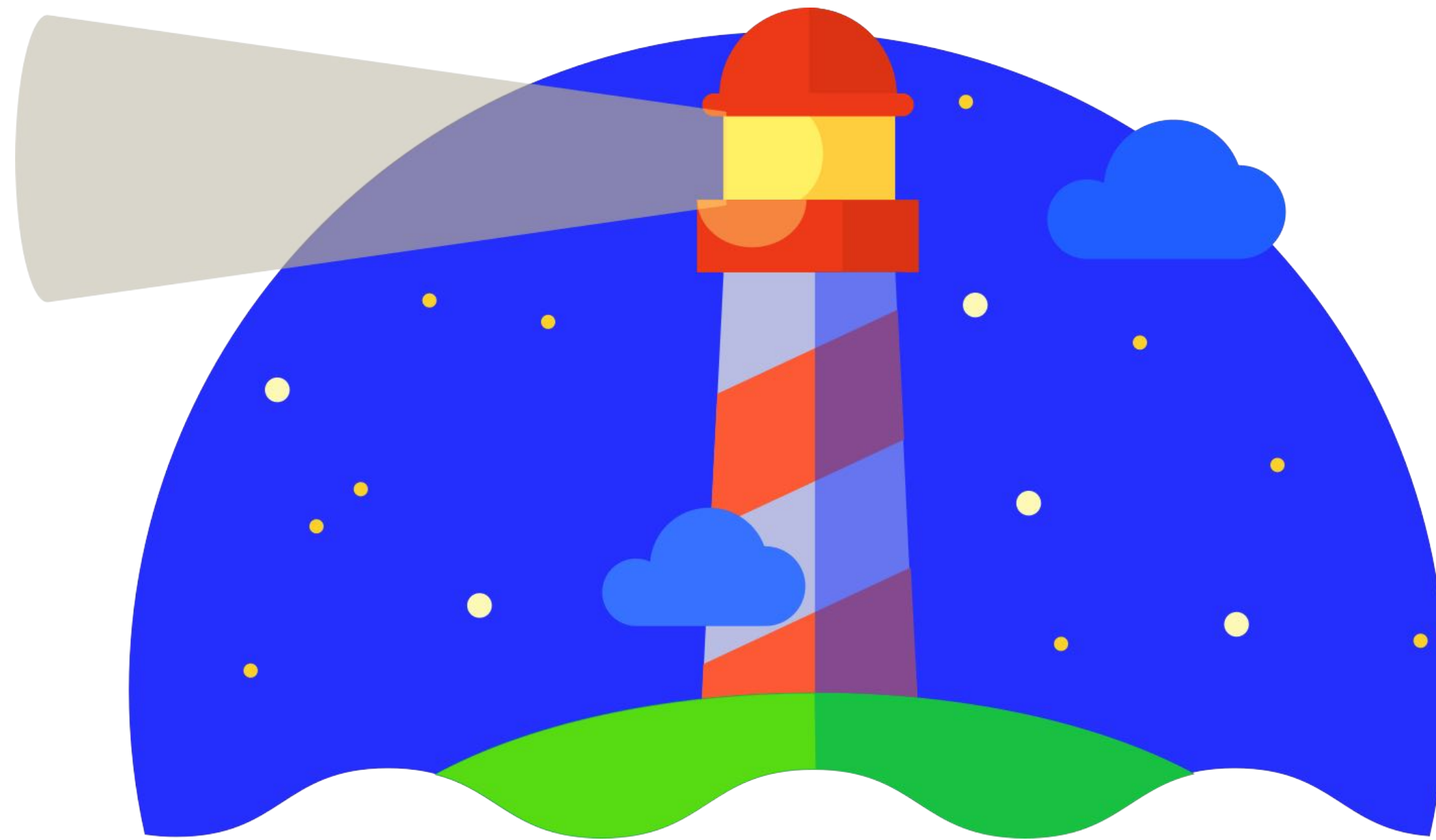
 **assono**

Strategie „Performance“ vs. „Freshness“



Chrome Developer Tools

- Über die Chrome Developer Tools kann ein Audit durchgeführt werden, um die PWA-Eigenschaften untersuchen zu können.





Hybride Apps



Hybride Apps

- Hybride Apps werden in HTML und JavaScript programmiert.
- Die fertige Web-Anwendung wird durch ein Framework in Code für eine native iOS- oder Android-Anwendung umgewandelt.
- Mit Xcode oder Android Studio wird der Code zu einer nativen App kompiliert.
- Apache Cordova ist die Basis für die meisten dieser Frameworks.



Ionic Framework

- Ionic basiert ebenfalls auf Apache Cordova
- Ursprünglich unterstützte Ionic nur Angular (bzw. AngularJS)
- Ab der Version 4.0 kann auch React oder Vue.js verwendet werden.
- Erste Ansätze sich von Cordova zu lösen.
 - Eigenentwicklung Capacitor
- Bezahlter Service für Build-Unterstützung
 - Erstellen von iOS-Apps ohne Mac



Ionic installieren

- Ionic verwendet den npm-Packager-Manager für den Install
- Installieren mit

```
$npm install -g ionic
```



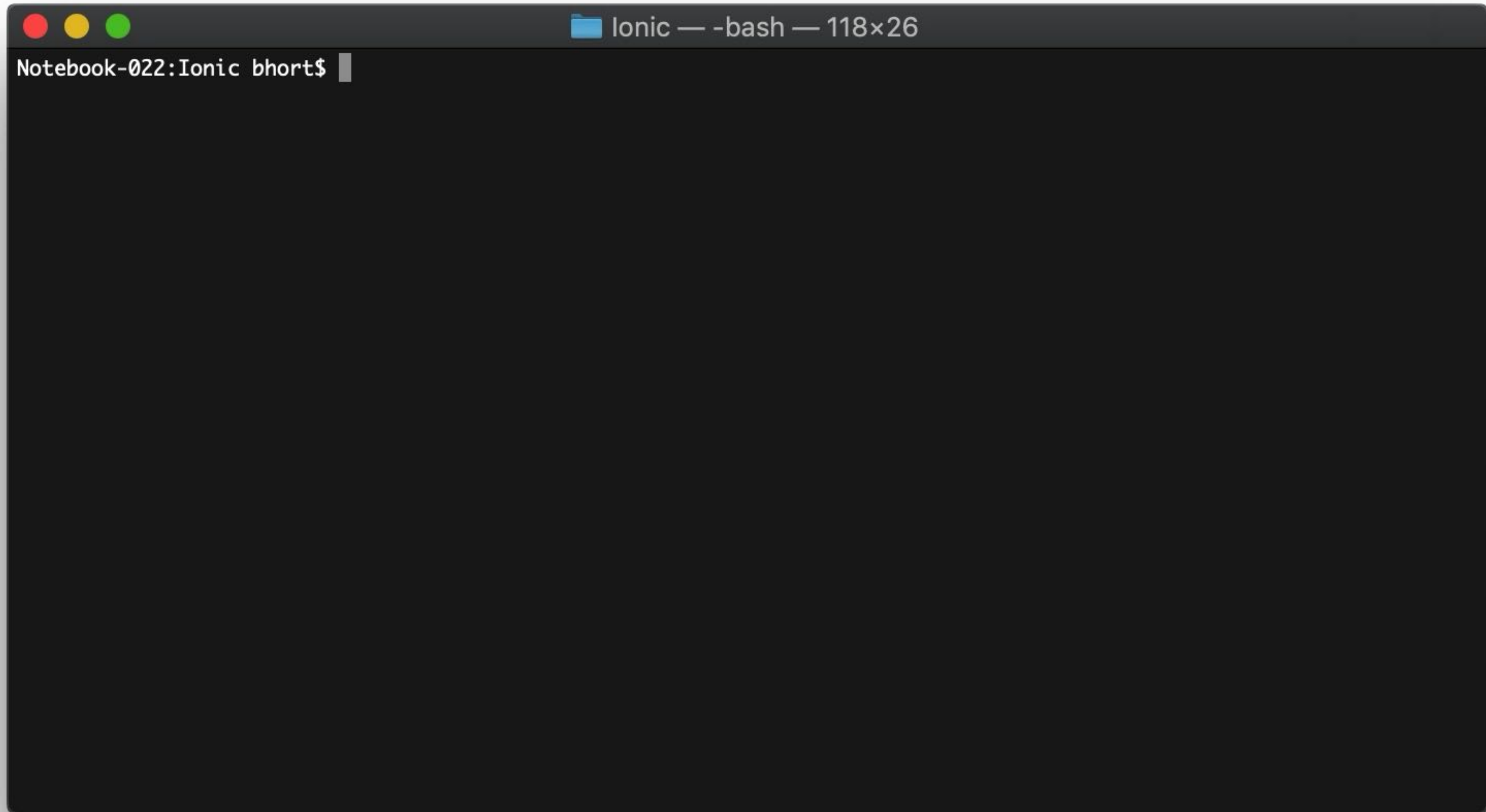
Neues Ionic Projekt



- Neues Projekt anlegen mit

```
$ionic start {Projektname}
```

- Weitere Parameter für die Art der Anwendung und ein Basistemplate



Cordova installieren

- Cordova verwendet ebenfalls den npm-Packager-Maneger für den Install
- Installieren mit

```
$npm install -g cordova
```



Storage einbinden

- Das Storage-Package kapselt die verschiedenen Storage Optionen
- Installieren mit

```
$npm install @ionic/storage
```

- Das Cordova Plugin für SQLite installieren

```
$ionic cordova plugin add cordova-sqlite-storage
```



Storage Module einbinden



- Wie auf der [Ionic](#)-Seite beschrieben
- Datei src/app/app.module.ts anpassen
 - Import hinzufügen

```
import { IonicStorageModule } from '@ionic/storage';
```

```
@NgModule({  
  ...  
  imports: [..., IonicStorageModule.forRoot()],  
  ...  
})
```

HTTP einbinden



- Das HTTP-Package kapselt den HTTP-Zugriff
- Installieren mit

```
$npm install @ionic-native/http
```

- Das zugehörige Cordova Plugin installieren

```
$ionic cordova plugin add cordova-plugin-advanced-http
```

HTTP einbinden



- Wie auf der [Ionic](#)-Seite beschrieben
- Datei src/app/app.module.ts anpassen
 - Import hinzufügen

```
import { HTTP } from '@ionic-native/http/ngx';
```

```
@NgModule({  
  ...  
  imports: [..., HTTP],  
  ...  
})
```

Verwendung von Storage und HTTP

- Der Entwickler ist selber dafür verantwortlich, die Strategie für das Zusammenspiel zwischen Storage und HTTP-Anfragen zu programmieren
 - Beim Ablegen von Daten im Storage den Zeitpunkt speichern
 - Für die Daten ein Ablaufzeitraum festlegen
 - Entscheiden, ob zuerst der Cache angefragt wird oder Daten via REST holen

Für iOS oder Android bilden



- iOS-Build

```
$ionic cordova prepare ios
```

- anschließend in Xcode Project öffnen

- Android-Build

```
$ionic cordova prepare android
```

- anschließend in Android Studio öffnen



Vergleich

Vorteile Progressive Web Apps

- Schnellere Entwicklung, weil direkt im Browser getestet werden kann.
- Kein aufwendiges Deployen auf Testgeräten
- Kein Approval im Apple iOS App Store bzw. Android Play Store
- Kein Apple Entwickler Lizenz notwendig
- Kein Organisations-App-Store notwendig
- Kein Mac notwendig

Vorteile Hybride Apps

- Zugriff auf native Ressourcen
 - Barcode-Scanner
 - Live-Bild
 - Push-Notifications unter iOS
- Native Look & Feel
- Größerer Ruhm!

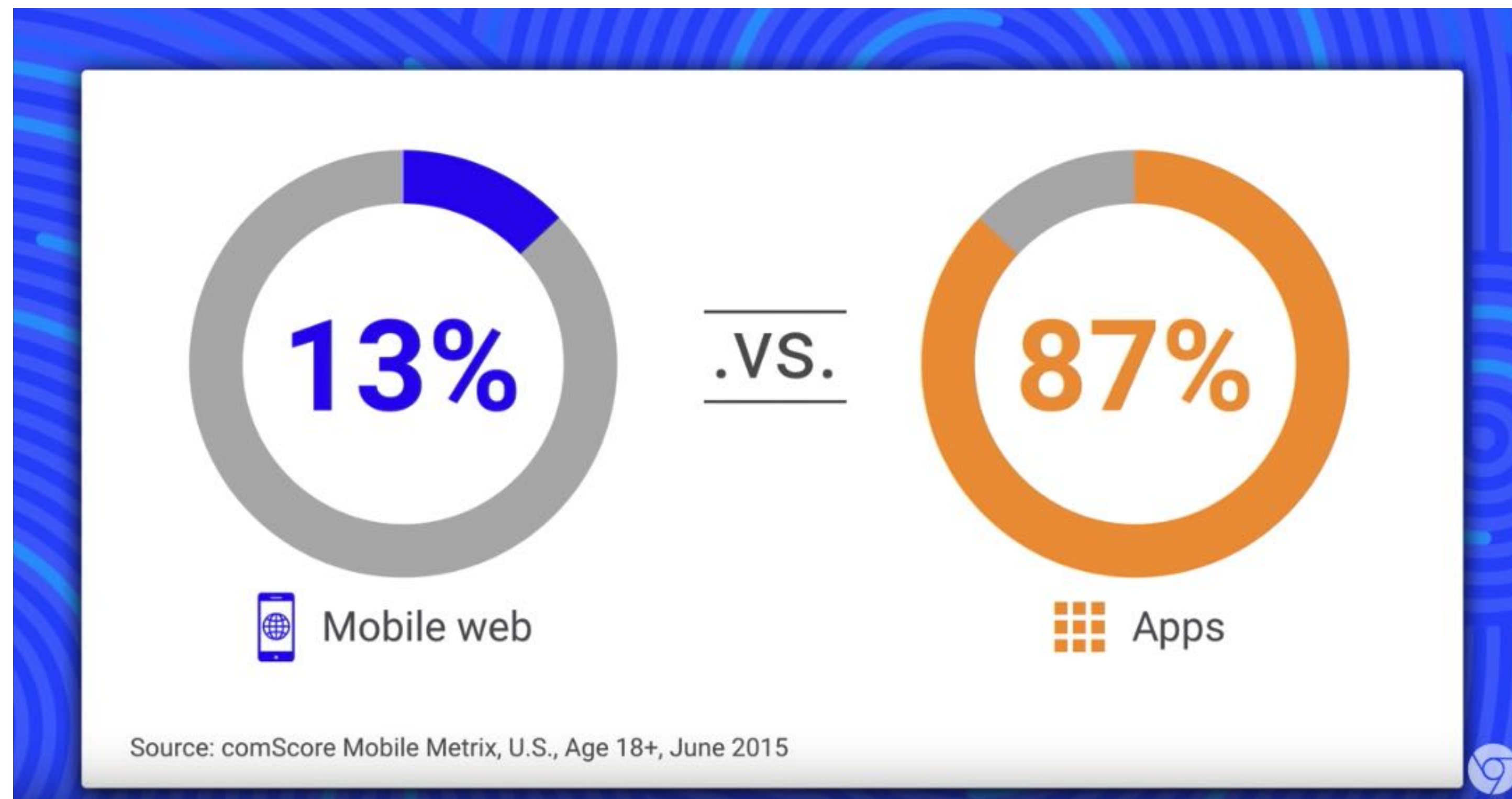


Fragen?

Bildnachweis

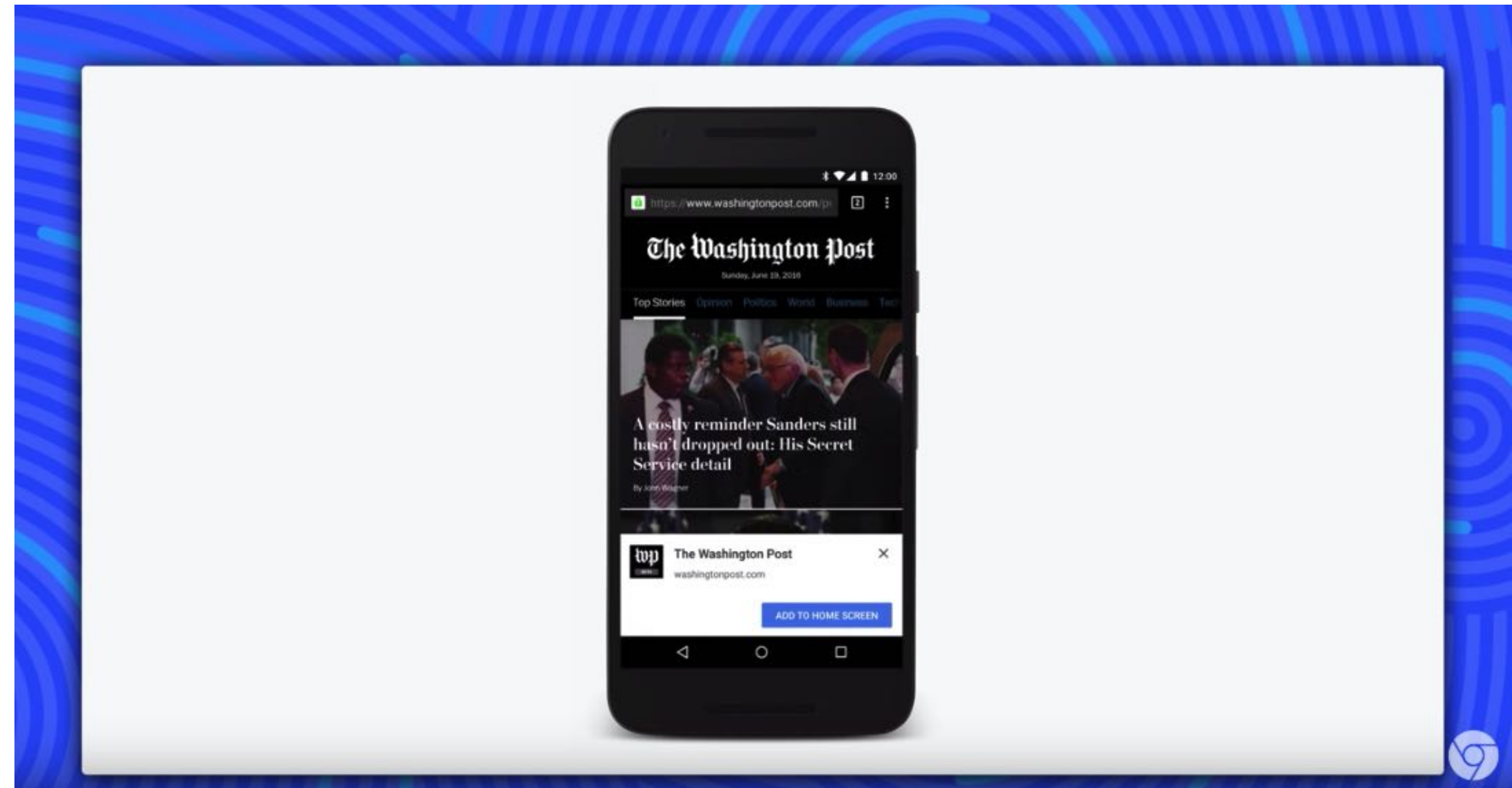
- Das PWA-Logo stammt von <https://github.com/webmaxru/progressive-web-apps-logo>
- Das Ionic-Logo stammt von https://de.wikipedia.org/wiki/Datei:Ionic_Logo.svg
-

Verteilung Mobile Web vs. Apps



„Add-to-Home-Screen“

- Erscheint automatisch nach „häufiger“ Benutzung der Webseite
- Wird vom Browser gesteuert!
- Kein Einfluss wann!



„Splash-Screen“

- Sobald auf dem Home-Screen, wird beim Starten ein Splash-Screen angezeigt.

