# PWA

## Progressive Web Apps

WJAX 2019

# Karsten Sitterberg

- Entwickler, Trainer, Berater
  - @kakulty
  - https://sitterberg.com
- Java User Group Münster
- Frontend Freunde Münster

JAVA USER GROUP MÜNSTER

# Vorteile: Native App

- Offlinefähig
- Installierbar
- Zugriff auf reichhaltige APIs

# Web

- Web ist offener Standard
- Deployment einfach
- Nicht installierbar, nicht offlinefähig
- Kein Zugriff auf System-APIs
- **Abhilfe: Progressive Web Apps**

Installierbar

Offline fähig

HTML5 APIs

# FIRE

**F**ast

App reagiert auf Nutzer-Eingaben **direkt** (Button-Toggle, Input, ...)

**I**ntegrated

Auf Device **installierbar** (wie native App)

**R**eliable

App lädt und öffnet **direkt**, auch wenn **offline**

**E**ngaging

**Interaktiv** durch Ausnutzung des Devices (Push, Sensors, Kamera, ...)

# Aktueller Plattform Support (Android 9, iOS 13)

| | Android | iOS |
|---|---|---|
| Service Worker | ✓ | ✓ |
| SW-Caching | ✓ | ✓ |
| Push | ✓ | x |
| Sync | ✓ | x |
| Manifest | ✓ | ✓ |

Progressive

# Einsatzbereich

Google-Suche umgestellt: PWA

Statt Native-Apps: Web Apps (Pizza bestellen, …)

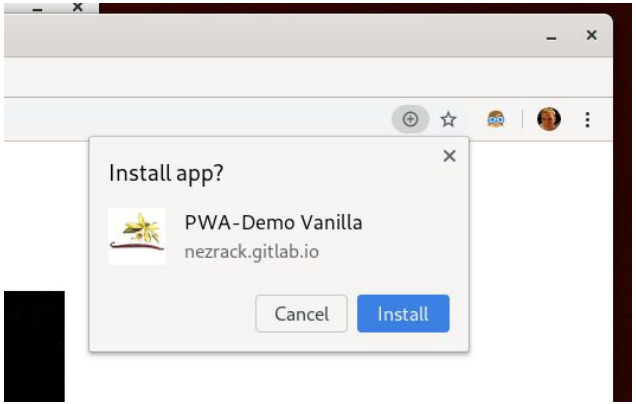
Ausgangsbasis für Native-App

# Einsatzbereich

## PWA ist für **jeden**!

Aber: Unterschiedliche “Nutzungstiefe” der Features

- Google-Suche umgestellt: PWA
- Statt Native-Apps: Web Apps (Pizza bestellen, …)
- Ausgangsbasis für Native-App

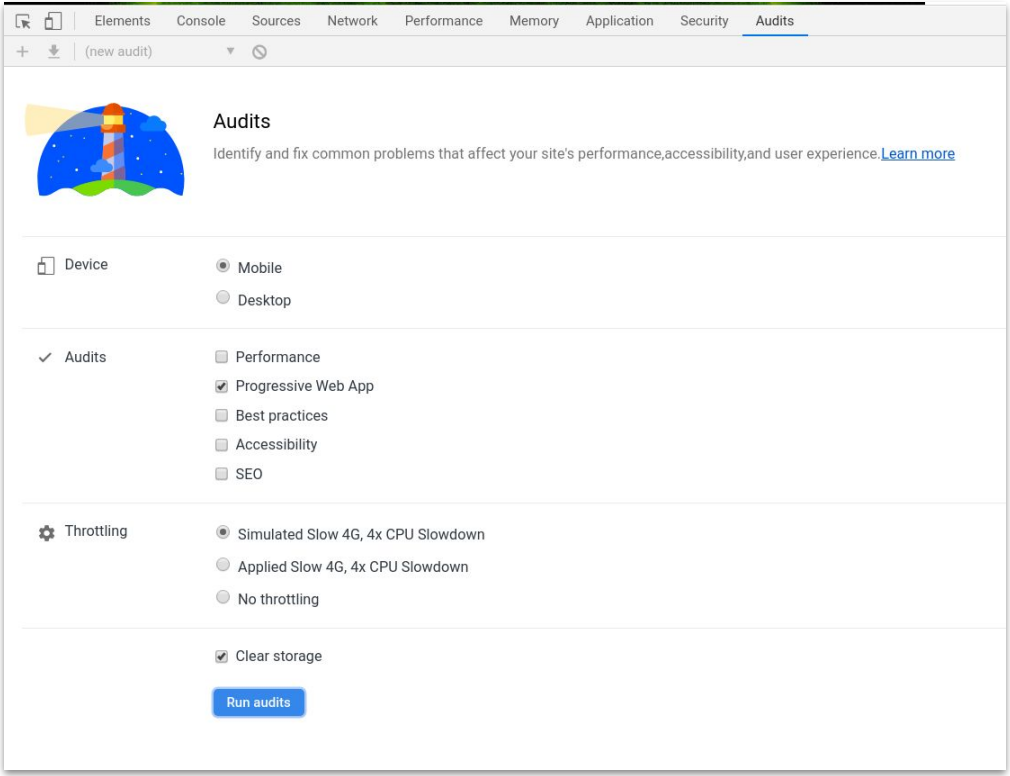


# Verwendete Software

# Angular

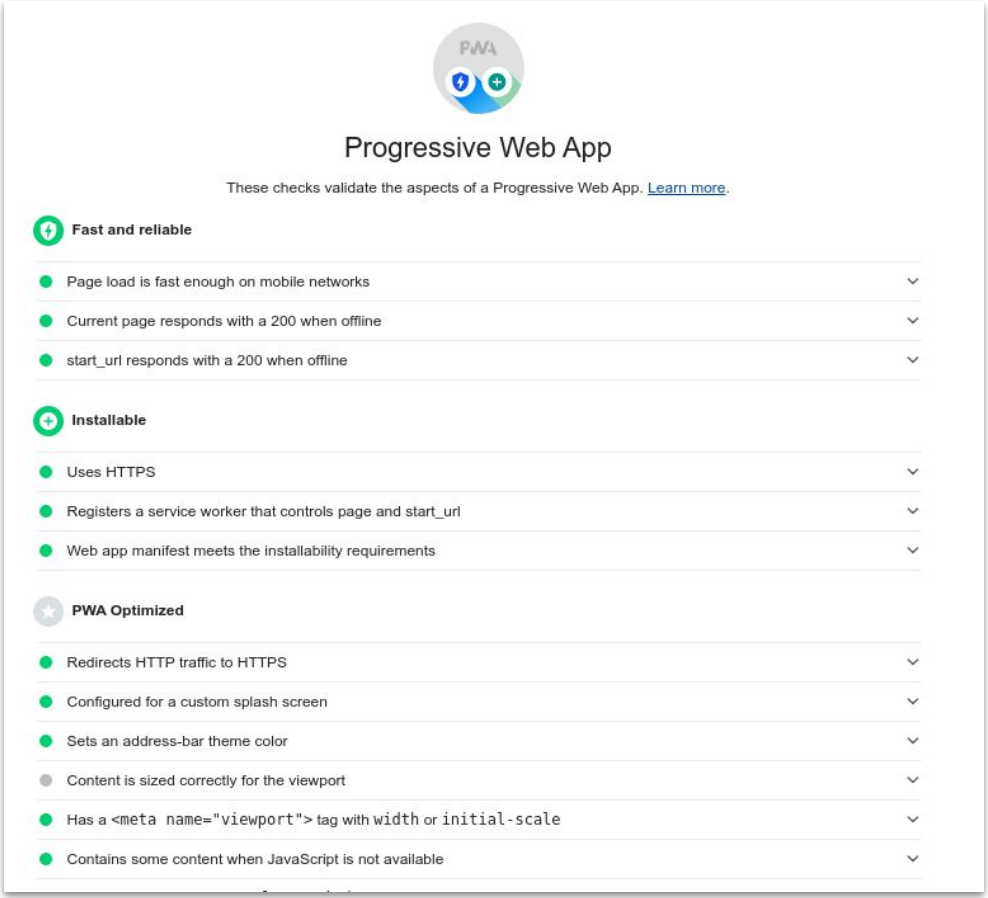
# Chrome Audits

- Ehemals “Lighthouse”



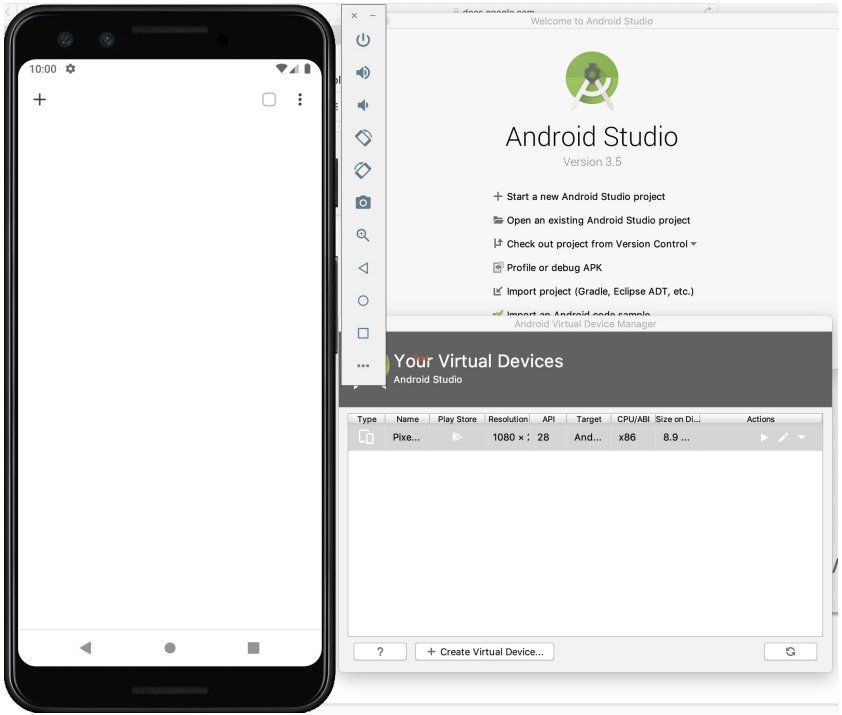
# Chrome Audits

- Ehemals “Lighthouse”



# Emulators

## Android Studio AVD



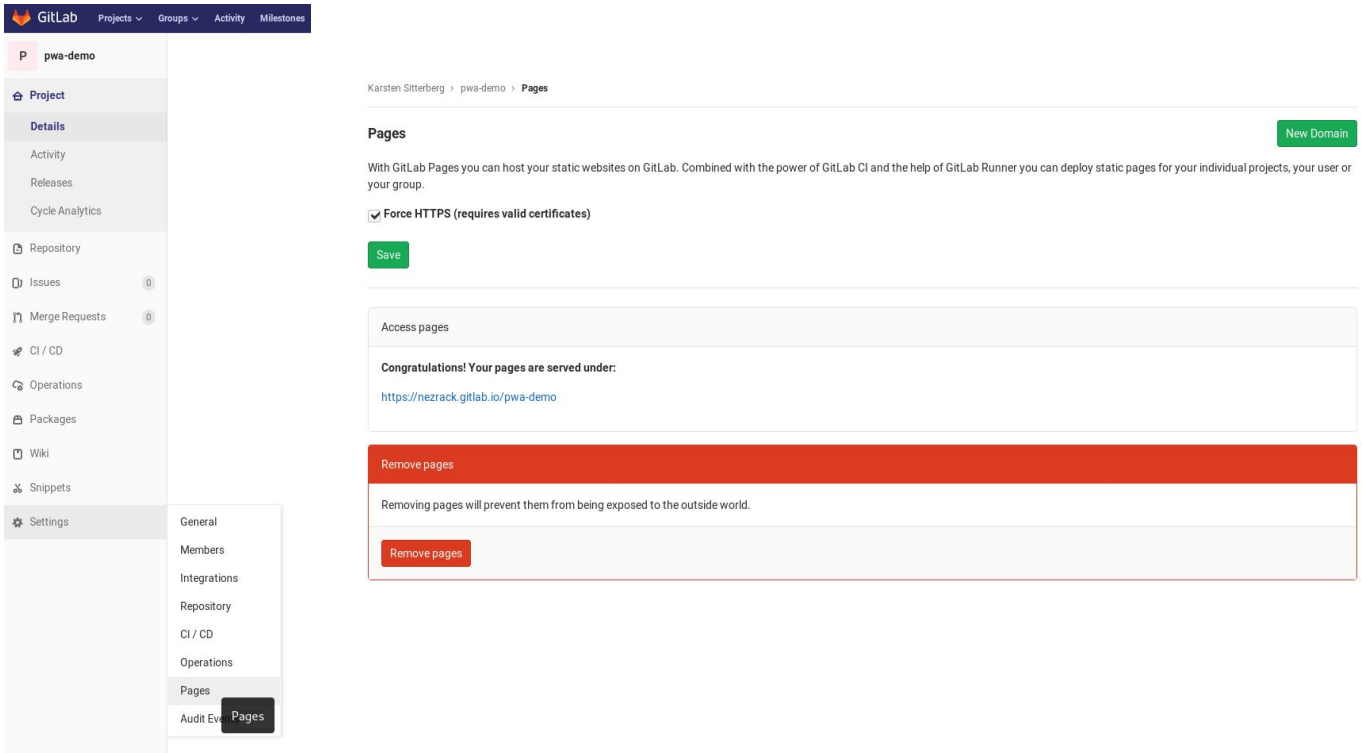

## XCode iPhone-Simulator

# GitLab

# GitLab Pages

- Automatischen Deployment via GitLab-CI
- Voraussetzungen
  - `.gitlab-ci.yml`
  - CI-Task, der Dateien in “`/public`”-Ordner kopiert

# GitLab Pages



# GitLab Pages

```
...


pages:
  stage: deploy
  script:
    - mkdir public
    - cp -r $DIST_DIR/* public
  artifacts:
    paths:
    - public
  only:
    - master
```

# HTML 5 APIs

Auch für "normale" Webseiten

# Achtung

**HTTPS** nutzen

Mit `localhost/127.0.0.1` reicht (meist) HTTP

**Aufpassen** mit Incognito/Private Browsing

Funktionen teils nicht verfügbar!

# Progressive Enhancement

```
if('serviceWorker' in navigator){ ... }

if('Notifications' in window) { ... }

if ('share' in navigator) { ... }
```

•

•

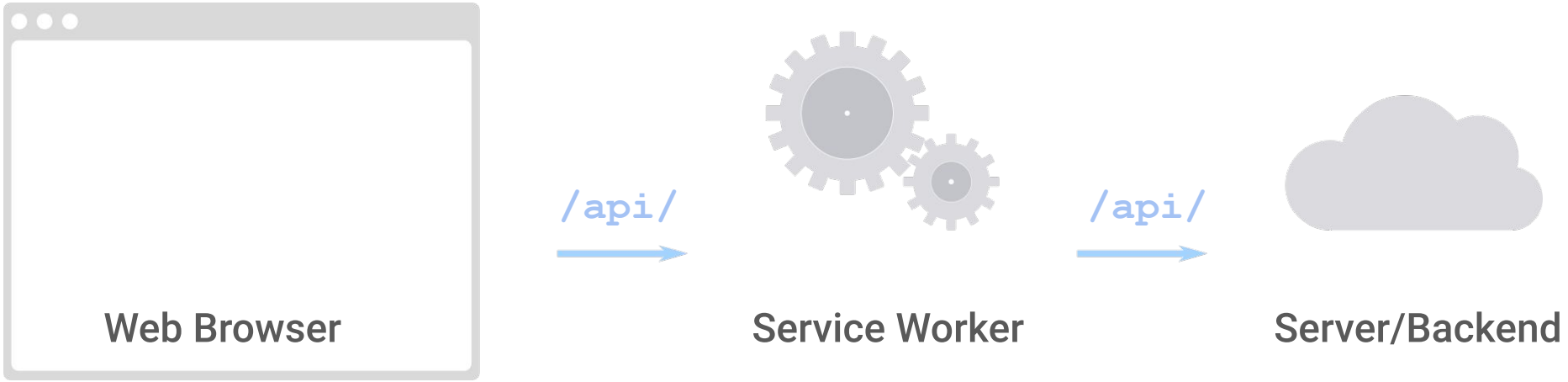
•

# Offline First

Anwendung für offline gebaut: “**Default**”

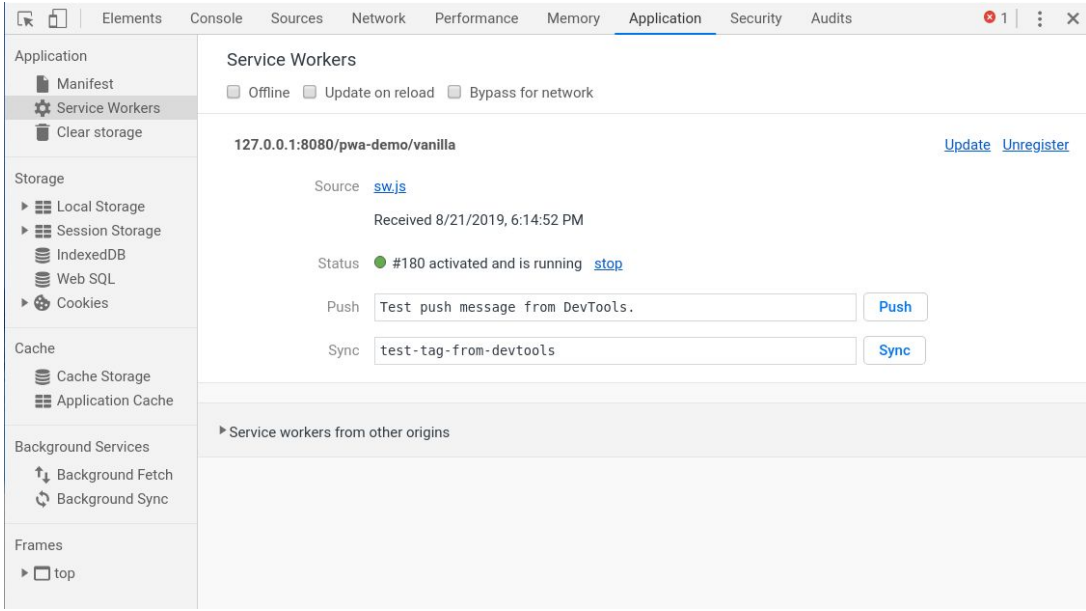
**Online** sein ist “**Feature**”

# Service Worker


/api/
/api/
Web Browser
Service Worker
Server/Backend

# Service Worker

- Debugging in Chrome-Devtools
- SW stoppen/entfernen
- SW aktualisieren
- Cache leeren
- Push/Sync-Events auslösen



# PWA-Manifest

# PWA soll installierbar sein

**Home-Icon**

**Splash-Screen**

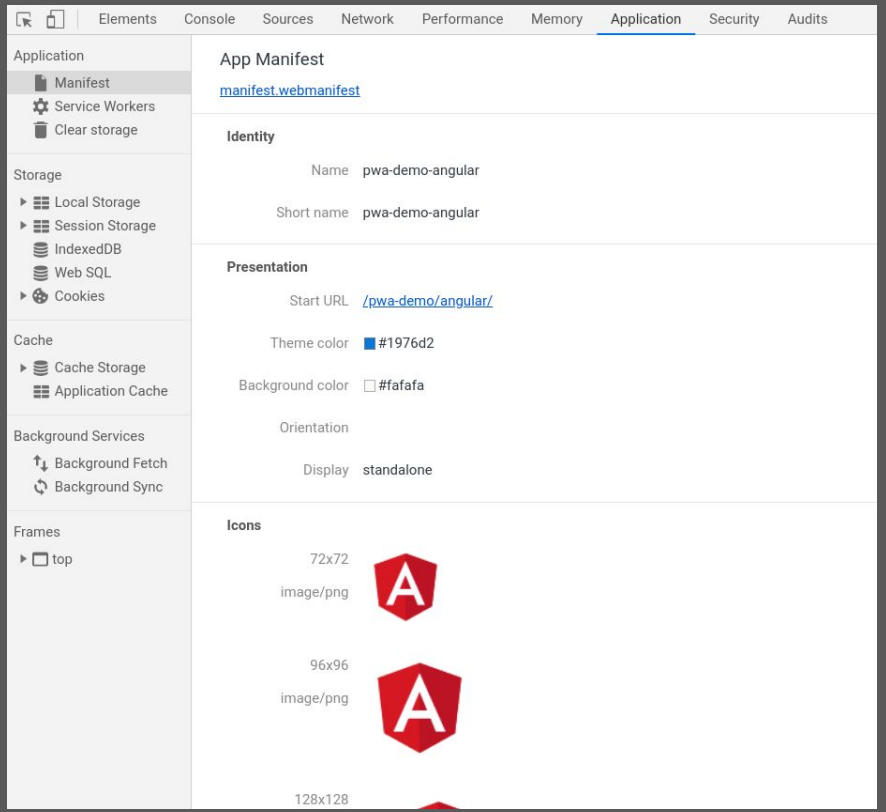
# PWA soll installierbar sein


# Web-Manifest

*index.html*

```
<head>
  <link rel="manifest" href="manifest.webmanifest">
  ...
</head>
```

# In Chrome Dev-Tools

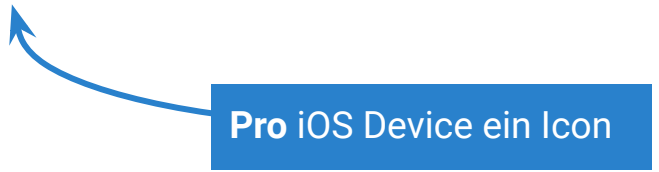


# Splash Screen

**iOS**: WebClip-Technologie (seit etwa **2008**!)

Mehrere **<meta>**- bzw. **<link>**-Angaben im HTML-Header erforderlich, z.B.:

```
<link rel="apple-touch-startup-image" href="/launch.png">
<meta name="apple-mobile-web-app-title" content="AppTitle">
<meta name="apple-mobile-web-app-capable" content="yes">
```



# Angular PWA

# Angular PWA


```
ng add @angular/pwa --project pwa-demo-ng
```

Installiert **`@angular/service-worker`**-Paket

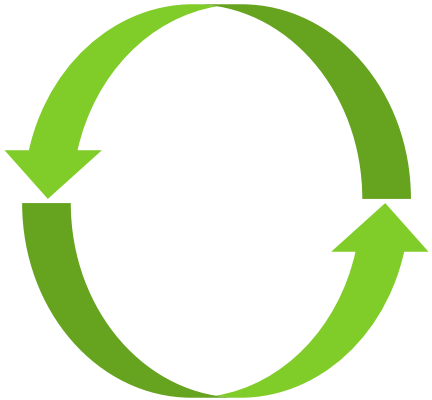
Aktualisiert **`index.html`** mit **Manifest** und **Theme**

Registriert + Konfiguriert Serviceworker:

**`/src/app/app.module.ts`**

**`/ngsw-config.json`**

# **SwUpdate**-Service



**a.** Event: **Erkennung** von App-Versionen

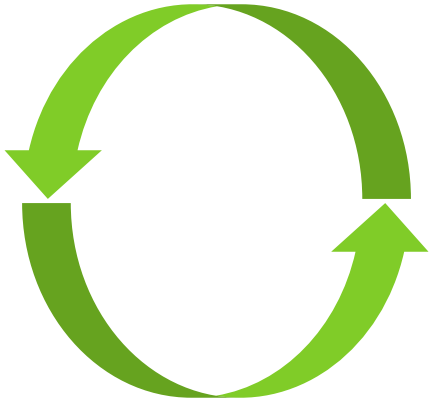
Auslieferung durch SW bei **Page-Refresh**

```
swUpdate.available.subscribe(event => {

  console.log('Momentanversion:', event.current);

  console.log('Verfügbar:', event.available);
});
```

# `SwUpdate`-Service



**b.** Event: **Aktivierung** einer Version

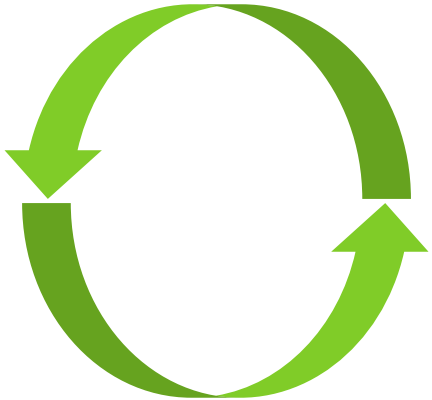
Service Worker liefert
neue Version **bei nächster Anfrage**

```
swUpdate.activated.subscribe(event => {

  console.log('Alte Version', event.previous);

  console.log('Neue Version', event.current);
});
```

# `SwUpdate`-Service

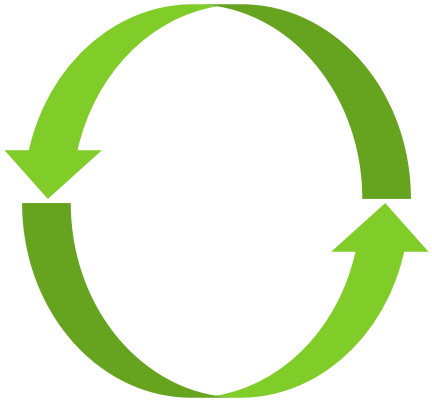


**c.** Server-Anfrage zur **Update-Erkennung triggern**

```
swUpdate.checkForUpdate()
```

Oder simpel: **Page Reload**

# **SwUpdate**-Service



**d. Versions-Aktivierung triggern**

Schaltet **Version im SW** “scharf”
Aktualisierung der App selbst: **Page-Reload**

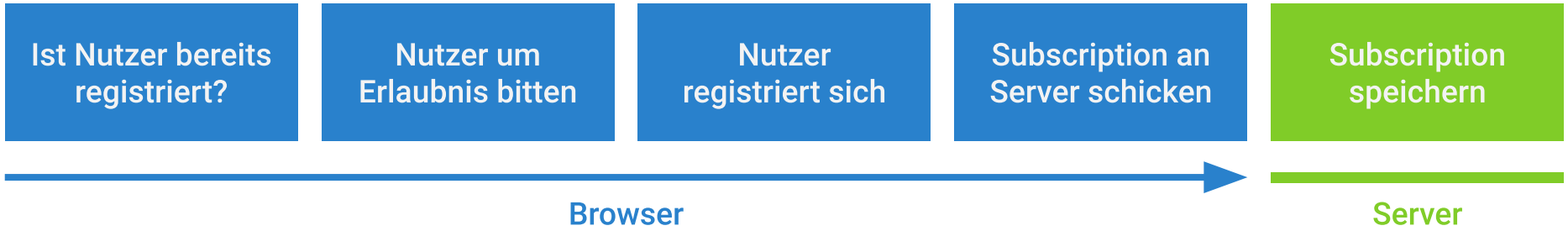
```
swUpdate.activateUpdate()
  .then(() => document.location.reload());
```

# Push-Notifications

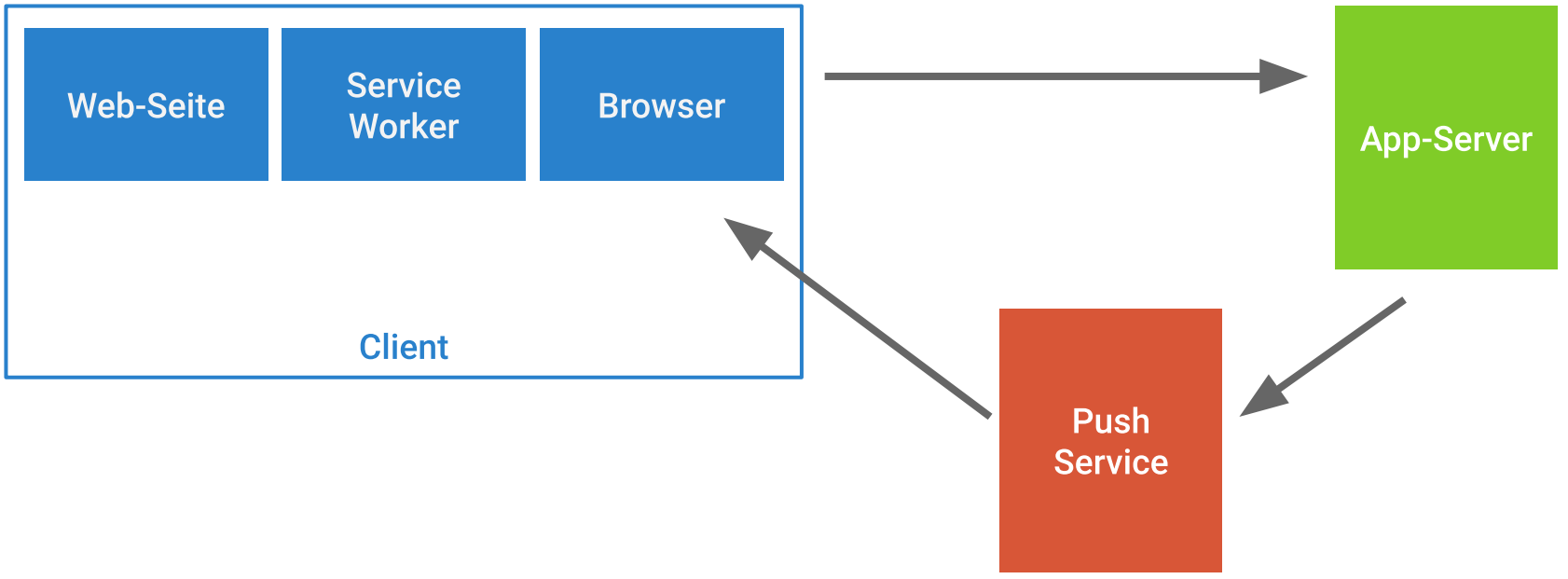
# Push Notifications

1. **Notification**: Nachricht, angezeigt **außerhalb** des Browser
2. **Push Message**: Vom Server zum Client gesendete Nachricht
3. **Push Notification**: Als Resultat der Push Message angezeigte Notification

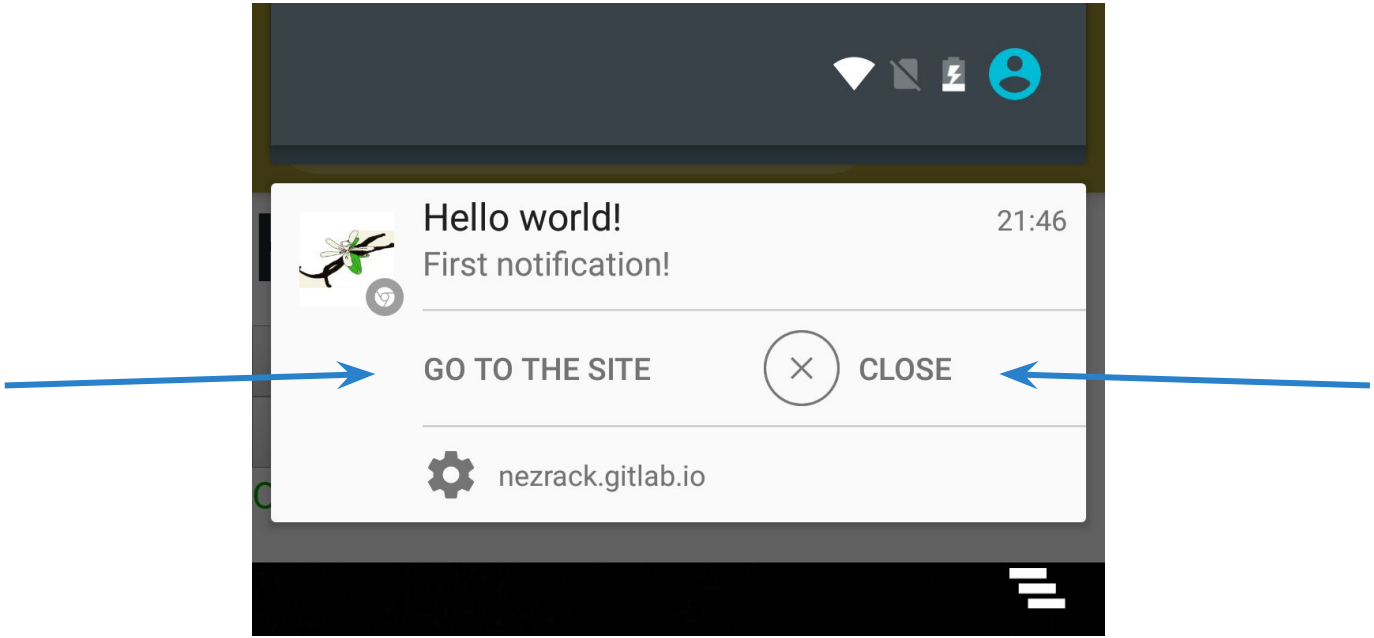
# Subscribe auf Push-Messages



# Architektur Push-Messages



# Notification Actions



# SW Push

# SW-Push

Angular bringt eigenen Push-Service (**SwPush**) mit

Im Paket @angular/service-worker

```
import { SwPush } from '@angular/service-worker';
/* ... */
export class AppComponent {
  constructor(readonly swPush: SwPush) {}
/* ... */
}
```

# SW-Push

Reaktion auf ankommende Push: **Automatisch**

Stattdessen: Subscription auf Notification-Clicks

```
this.swPush.notificationClicks.subscribe(
      ({action, notification}) => {
        console.log(action, notification);
      });
```

# SW-Push

Notwendig dafür:

Spezielles Format der Notification
(von Server zu Push-Service!)

```
{
  "notification": {
    "actions": NotificationAction[],
    "badge": USVString
    "body": DOMString,
    "data": any,
    "dir": "auto"|"ltr"|"rtl",
    "icon": USVString,
    "image": USVString,
    "lang": DOMString,
    "renotify": boolean,
    "requireInteraction": boolean,
    "silent": boolean,
    "tag": DOMString,
    "timestamp": DOMTimeStamp,
    "title": DOMString,
    "vibrate": number[]
  }
}
```

# Angular App-Shell

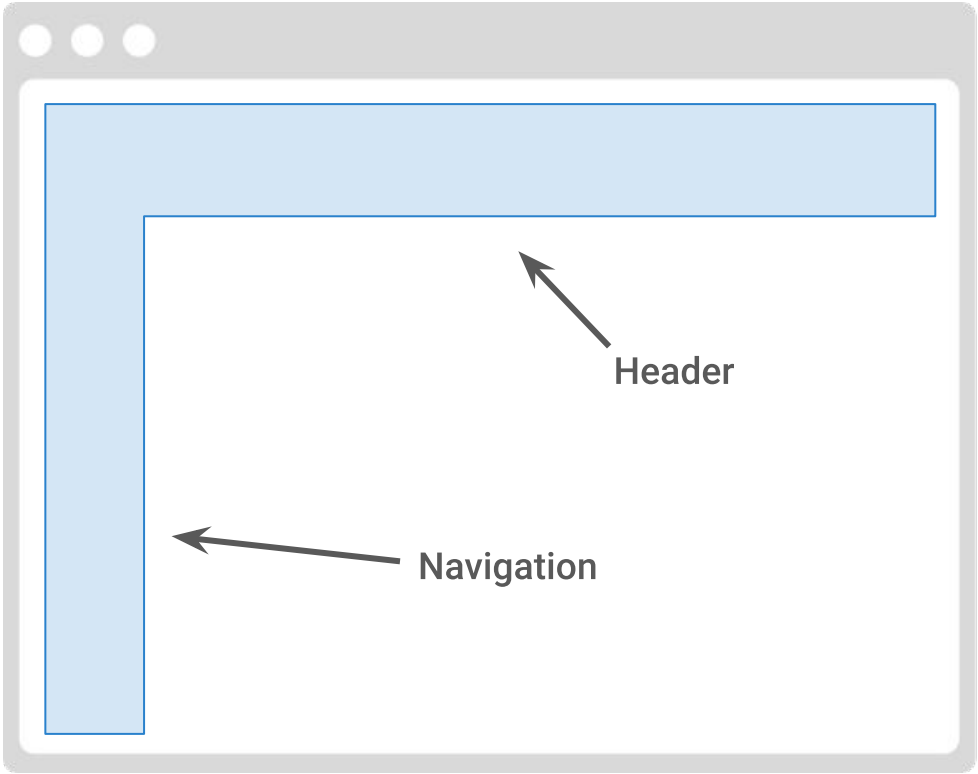
# App-Shell

Minimales HTML/CSS/JS
(Statischer) Content

**-> Direkt da - dank Cache!**

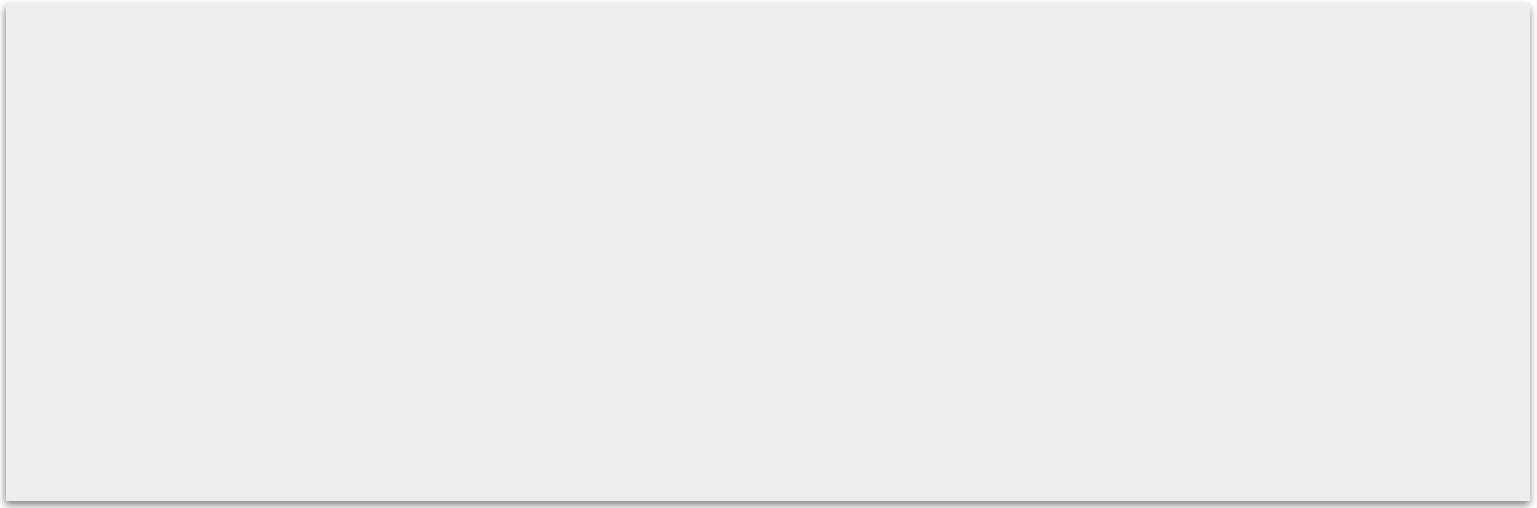
(Dynamischer) Content

-> Laden/Netzwerk



# Angular App-Shell

Generiert via Server-Side-Rendering



# Angular App-Shell

Generiert via Server-Side-Rendering

Benötigt Angular-Routing

**`RouterModule.forRoot()`** bzw.
**`AppRoutingModule`**

# Angular App-Shell

App-Shell per Server-Side-Rendering:

```
ng generate app-shell           \
 --client-project pwa-demo-ng   \
 --universal-project server-app
```

@angular/universal

# App-Shell bauen

```
ng run sensor-pwa:app-shell -c production
```

# Zusammenfassung

# PWA Einstieg

- Fokus auf Nutzer legen
- Features mit höchstem Nutzen zuerst
- SPA + PWA ~ App

# Deployment

- Statische Webseite
- Server Side rendered Anwendung
- Single Page Applications
- SPA als mobile App paketieren

Nutzung von nativen APIs

pwabuilder.com

HTTPS

# Vielen Dank!

Fragen?