Javascript im LAMP Stack

Javascript im Allgemeinen

- Seit 1995
- Früher eher stiefmütterlich behandelt
- Bekannteste ECMAScript Implementierung
- Dynamisch typisierte Skriptsprache
  - Oft Event-Getrieben
  - Empfehlung: FWPM “Javascript” von Sebastian Springer
- Multi-Purpose, aber ursprünglich Browser als Laufzeitumgebung
  - Dynamisches Verhalten über DOM Manipulation
- Mittlerweile omnipotent
  - Serverseitig z.B. mit Node.js
  - Embedded Scripting Sprache in CI Systemen
  - Als Basis für Standalone Anwendungen z.B. mit Electron

Prozedural vs. OOP vs. Funktional

- JS biete (auch in der Praxis) mehrere Programmierparadigmen
  - Daher sehr vielseitig in der Einsetzbarkeit

Prozedural

```
<script>
    let elem = document.getElementById("animate");
    let pos = 0;
    let id = setInterval(frame, 5);
    function frame() {
        if (pos === 350) {
            clearInterval(id);
        } else {
            pos++;
            elem.style.top = pos + "px";
            elem.style.left = pos + "px";
        }
    }
</script>
```

OOP

```
class Rectangle {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
    // Getter
    get area() {
        return this.calcArea();
    }
    // Method
    calcArea() {
        return this.height * this.width;
    }
}
```

Funktional

```
let barLogger = function() {
    console.log('bar');
};

let logHolder = {
    main: barLogger
}

let logger = function (logging) {
    logging();
}

logger(logHolder.main);
```

Warum Javascript?

- Dynamische Oberflächen von Webanwendungen
  - Animationen
  - Nutzerinteraktion
- Verbesserung der Usability
  - Z.B. direktes Feedback bei Validierungen
- Nutzung von Client APIs
  - Local Storage, DOM
- Inter-Plattform Entwicklung
  - Webapps mit cordova, Desktop Anwendungen mit Electron
- Stateful Look an Feel

>> https://www.w3schools.com/HTML/tryit.asp?filename=tryhtml5_draganddrop

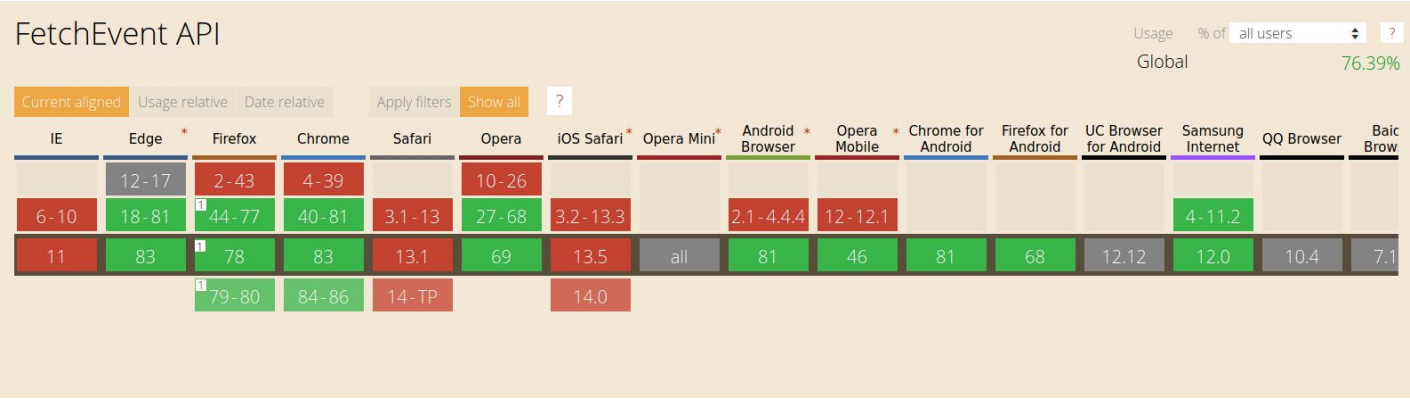
Warum nicht?

- Bringt zusätzliche Sprache ins Projekt
  - Teils sehr komplexer Syntax
- Verwendung nicht immer nötig
  - Verkompliziert oft einfache Oberflächen
  - Vieles mittlerweile nativ mit HTML und CSS lösbar
- Bringt Performance-Probleme zum Nutzer
  - Z.B. “unresponsive script” Nachricht
- Frage muss sein: Warum nicht im geplanten Umfang?
  - Umfang variiert enorm
  - Muss zur Anforderung passen
- Nicht immer verfügbar (aber fast immer)
- Oft Teil schnelllebiger Technologie
  - Achtung vor dem Hype Train!

```
const f = (m) => () => console.log(m)
const f2 = (f3) => f3()
f2(f( m: 'Test'))
```

Achtung: Browser Runtime

- Javascript ist von Laufzeitumgebung im Browser abhängig
- Genutzte Features müssen überprüft werden
  - Vergleich mit eigener Zielgruppe
  - Testing unter verschiedenen Bedingungen
  - Nutzung von caniuse.com, browserstack.com, etc.
- polyfills helfen



Integration - Inline Script

- Teil des HTML
  - An beliebiger Stelle möglich
- Sollte weit unten integriert sein
  - Render wird gestört
- Wird mit HTML Interpretation ausgeführt
- Sinnvoll für kleine Snippets
  - Ad-hoc Funktionen die sofort gebraucht werden
    - Z.B. Registrieren von Event Handlern

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            function myFunction() {
                alert('I was called');
            }
        </script>
    </head>
    <body>
    <h1>A Web Page</h1>
        <script type="text/javascript">
            function myFunction() {
                alert('I was called');
            }
        </script>
    </body>
</html>
```

>> https://www.w3schools.com/js/js_whereto.asp

Inline Scripts

- Oft zur Integration von Drittanbieter Leistungen
  - Analytics, Social Media Buttons, etc.
- Sinnvoll für zwingend benötigte Logik
  - Z.B. zur Seitendarstellung
- Teil von dynamisch gerenderten Inhalten
  - In genutzten Komponenten enthalten
- Sollte wenn möglich vermieden werden
  - Weniger Preprocessing Möglichkeiten
  - Zerstört technologische Trennung
  - Kann kaum ge-cached werden

```
<script>
var gaProperty="YOUR_ID_HERE",disableStr="ga-disable-"+gaProperty;-1
<document.cookie.indexOf(disableStr+"\x3dtrue")&&(window[disableStr]=!0);
function gaOptout(){document.cookie=disableStr+
    "\x3dtrue; expires\x3dThu, 31 Dec 2099 23:59:59UTC; path\x3d/";
window[disableStr]=!0;alert("Das Tracking wurde erfolgreich deaktiviert")}
</script>
```

Inline Scripts - Beispiele

Animationen

```
function myMove() {
    var elem = document.getElementById( elementId: "animate");
    var pos = 0;
    var id = setInterval(frame,  timeout: 5);
    function frame() {
        if (pos === 350) {
            clearInterval(id);
        } else {
            pos++;
            elem.style.top = pos + 'px';
            elem.style.left = pos + 'px';
        }
    }
}
```

Validierung

```
function fieldValidation() {
    let inpObj = document.getElementById( elementId: "id1");
    if (!inpObj.checkValidity()) {
        document.getElementById( elementId: "demo").innerHTML = inpObj.validationMessage;
    } else {
        document.getElementById( elementId: "demo").innerHTML = "Input OK";
    }
}
```

Event Handling

```
<script>
function myFunction() {
    // Declare variables
    var input, filter, ul, li, a, i, txtValue;
    input = document.getElementById('myInput');
    filter = input.value.toUpperCase();
    ul = document.getElementById("myUL");
    li = ul.getElementsByTagName('li');

    // Loop through all list items, and hide those who don't
    for (i = 0; i < li.length; i++) {
        a = li[i].getElementsByTagName("a")[0];
        txtValue = a.textContent || a.innerText;
        if (txtValue.toUpperCase().indexOf(filter) > -1) {
            li[i].style.display = "";
        } else {
            li[i].style.display = "none";
        }
    }
}
</script>
```

>> https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_animate_3
>> https://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_check
>> https://www.w3schools.com/howto/tryit.asp?filename=tryhow_js_filter_list

Integration - Externe Ressource

- Erlaubt technologische Trennung
- Bietet asynchrones Laden (mit *async* und *defer*)
  - Löst “Render Blocking” Probleme
  - Reduziert initiale Dateigröße
  - Aber auch synchron möglich
- Erlaubt Nutzung externer Ressourcen
  - Ohne Notwendigkeit eigener Pflege
  - Erlaubt Nutzung von CDN
- Nützlich für abgeschlossenen Code
  - Bibliotheken
  - Eigene Funktionen
  - Bei Mehrfachnutzung

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://cdn.test/script.js" async defer></script>
    </head>
    <body>
    <h1>A Web Page</h1>
        <script src="../js/script.js"></script>
    </body>
</html>
```

Integration - APIs

- Laufzeitumgebung bietet zahlreiche Einhängepunkte
  - Koordinieren eventbasiert JS Aufrufe
  - Liefern Informationen oder Zugriffsmöglichkeiten an JS
- Z.B.:
  - HTML Manipulation (DOM)
  - Audio und Video (HtmlMediaElement, Web Audio API, ...)
  - Zeichnen und Bildmanipulation (Canvas, WebGL, …)
  - Geräteschittstellen (Vibration API, …)
  - Speichererweiterung (IndexedDB, SessionStorage, …)
  - ...

```
<input type="text" onchange="myScript">
```

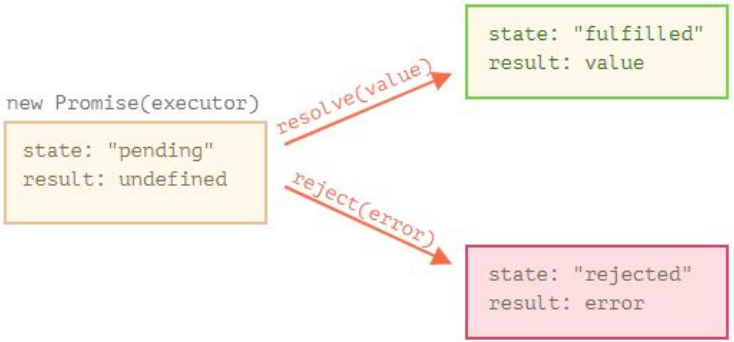
```
object.onchange = function(){myScript};
object.addEventListener("change", myScript);
```

AJAX/Async Load

- **A**synchronous **J**avascript **a**nd **X**ML
- Hat Request/Response Modell revolutioniert
- Über XMLHttpRequest API
- Asynchrones Laden von Inhalten innerhalb Server-Side Anwendungen
  - Ist in HTML eingebettet
  - Tauscht selektiv Daten im DOM aus
- Promises lassen sich nutzen um asynchron Daten zu laden
  - Laden von Inhalten blockiert Interaktion nicht
- Security im Blick behalten
  - Policies über Header beachten

Promises und Callbacks

- Moderne Oberflächen brauchen Nebenläufigkeit
  - Synchrone Abfragen blockieren (siehe Ladezeit bei neuem Request)
- Javascript ist aber Single-Threaded
  - Nutzt sog. Event Loop als Warteschlange von Funktionsaufrufen
    - Erschwert Blockieren
- Promises sind Einhängepunkte in Event Loop in Abhängigkeit anderer Funktionen

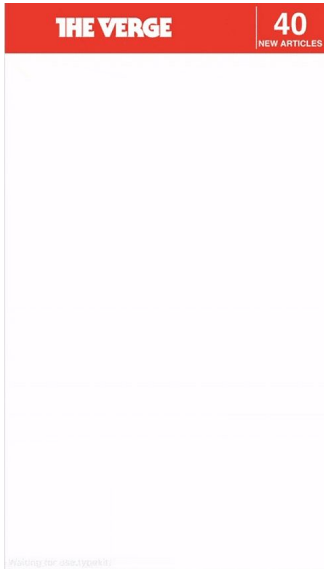


Promises und Callbacks

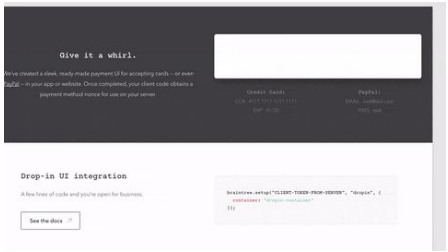
```
// This promise does nothing than sleep a second then resolve
const promise = new Promise( executor: (resolutionFunc,rejectionFunc) => {
    setTimeout(resolutionFunc, timeout: 1000);
    setTimeout(rejectionFunc, timeout: 2000);
});
// Register a resolution and an rejection callback with actual logic
promise.then(
    () => console.log("I come a second later"),
    () => console.log("I will never be called")
);
console.log("I come first");

// Result will be:
// I come first
// I come a second later
```

AJAX/Async Load - Probleme



- Aber: bringen neue Anforderungen an Nutzer-Feedback
  - Z.B. Spinner/Loader wo Daten fehlen
- DOM Manipulation wirkt bei langen Requests willkürlich
- Nachvollziehbarkeit durch Nicht-Digital Natives suboptimal
- Größe von Inhalten kann DOM verschieben
  - Sog. Layout Shifting
  - Benötigt zusätzliche Arbeit in CSS
    - Technologien müssen besser abgestimmt sein
  - Oft schwer bei Fremdinhalten, z.B. Werbung
- Nebenläufigkeit führt zu Seiteneffekten
  - Schwerer zu managen als SSR



AJAX/Async Load - Vorteile

- Asynchronität sorgt für bessere Performance
  - Es werden nur benötigte Daten geladen
  - Teilaktualisierung benötigt weniger Rendering
  - Schnelleres Feedback an User
- Vorteil von Fake Performance
  - Mehrstufiges Laden suggeriert Aktivität

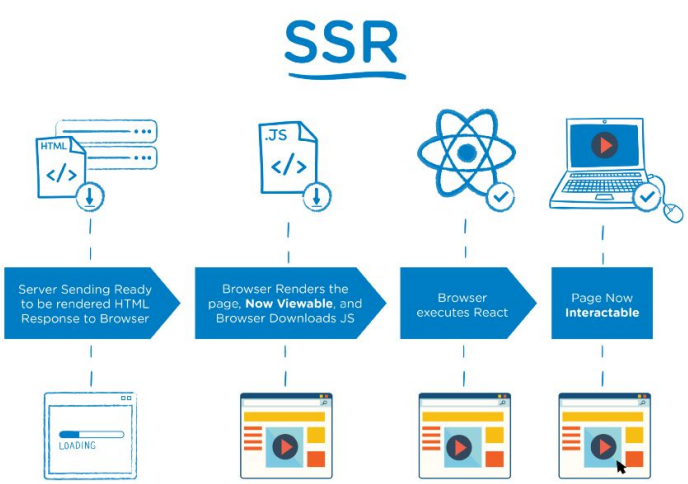
>> https://cloudcannon.com/deconstructions/2014/11/15/facebook-content-placeholder-deconstruction.html

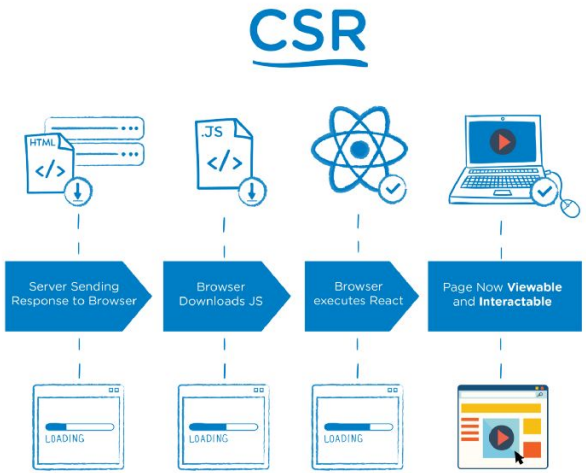
SPA

- **S**ingle **P**age **A**pplication
- Wollen State im Client halten
  - Nicht nur selektives Laden einzelner Teile
- Moderne JS Ansätze als Framework für komplette Client Anwendung
  - Kein serverseitiges Rendering mehr
    - Außer eventuell initialer *index.html*
  - Inhalte in Form von Daten z.B. per Webservice Aufrufe
- Lösen sichtbares Request/Response Modell auf
  - Wirken Stateful, Nötige Requests im Hintergrund über asynchrone Aufrufe
- Große Auswahl an aktuellen Frameworks
  - Angular, Vue.js, Ember, …

>> https://www.proffile.de/app/profile/techdivision-gmbh

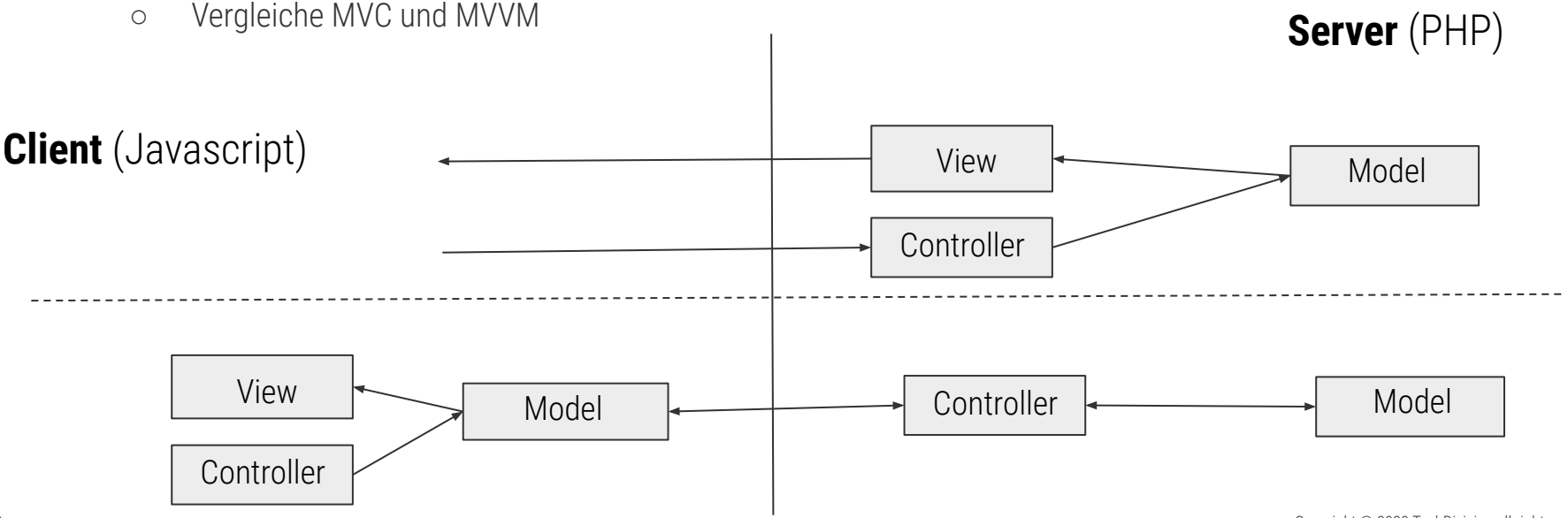
SPA - Client Side Rendering (CSR)





SPA - Unterscheidung

- SPAs nutzen ausschließlich CSR
- Alle Vor- und Nachteile von AJAX/Async Load
- SPAs beinhalten eigenes Framework mit eigener Architektur
  - Vergleiche MVC und MVVM



PWA

- **P**rogressive **W**eb **A**pplication
- Ergänzen SPAs sinnvoll
- Erlauben Offline Einsatz und Integration in Host APIs
  - Cachen Inhalte aggresiv
  - Sollen wie native Apps nutzbar sein
- Nutzen erweiterte Speicherkapazitäten im Browser
  - IndexedDB, SessionStorage, etc.

PWA - Manifest

- Datei zur Weitergabe von Metadaten
  - W3C Draft
- Enthält Basisinformation zur Anwendung
  - Name
  - Verlinkung von Icons und Bildmaterial
  - Konfiguration
- Einheitliche Schnittstelle zur App Integration
- Noch nicht immer unterstützt
  - Apple geht z.B. Sonderweg

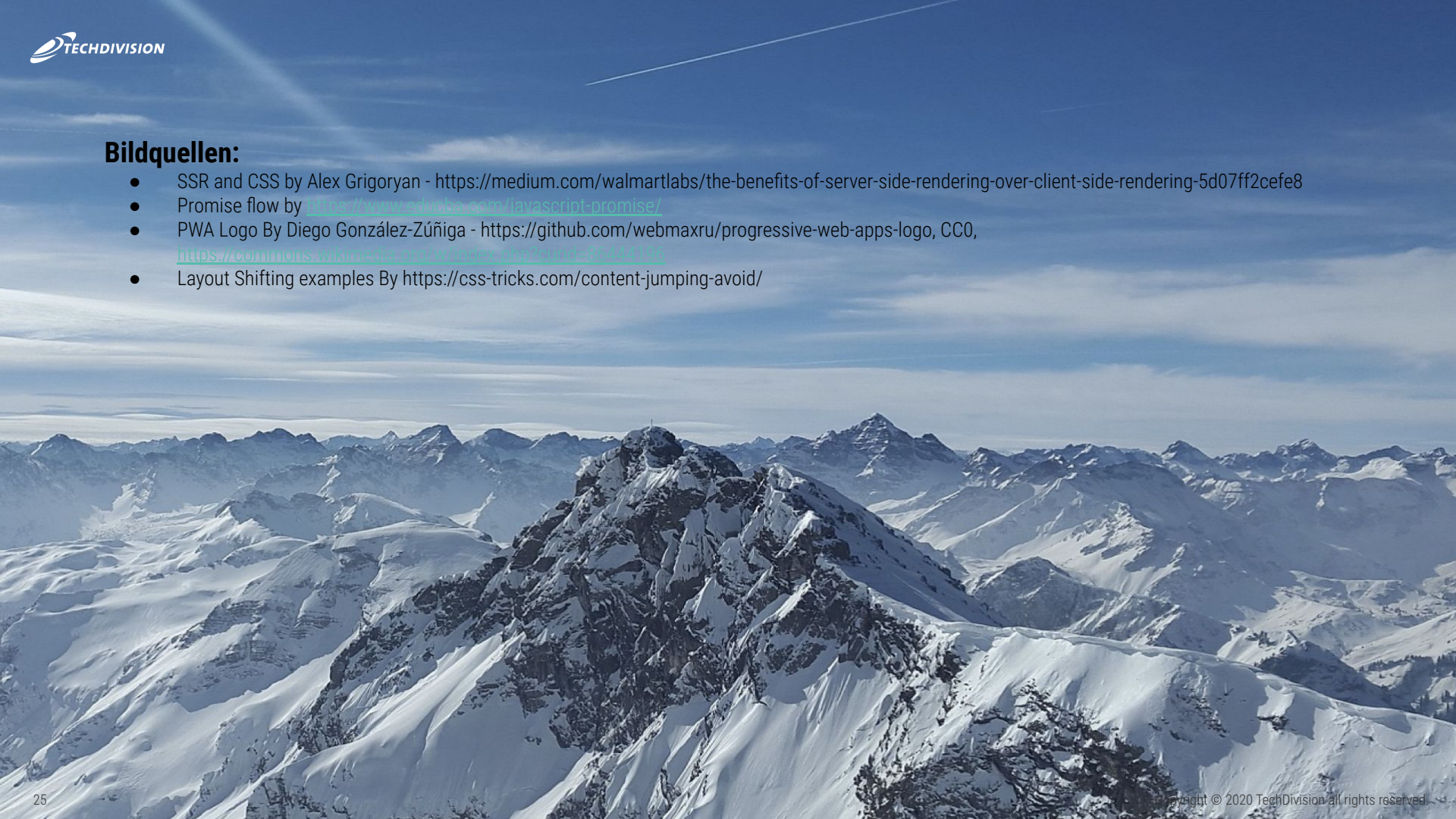
```
{
  "name": "TH Rosenheim Blog",
  "short_name": "TH Blog",
  "start_url": ".",
  "display": "standalone",
  "background_color" : "#fff" ,
  "description": "Eine einfach lesbare Blog App fü
  "icons": [{
    "src": "images/touch/homescreen48.png",
    "sizes": "48x48",
    "type": "image/png"
  }],
  "related_applications": [{
    "platform": "Web"
  }, {
    "platform": "play",
    "url": "https://play.google.com/store/apps/det
  }]
}
```

PWA - Service Worker

- API zur Programmierung unabhängiger, nebenläufiger Skripte
- Service Worker
  - Laufen parallel zur Anwendungs JS
  - Können Netzwerkverkehr auswerten
  - Haben keinen Zugriff auf DOM
  - Sind an Domains gebunden
  - Können untereinander kommunizieren
- Erlauben z.B.
  - Dynamisches Caching von Anfragen im Client
    - Auch Caching von Anfragen möglich für Offline Nutzung
  - Push Nachrichten auch bei geschlossener Seite
  - Nachladen von Informationen (Polling)

Quellen:

- https://www.omgubuntu.co.uk/2019/02/best-electron-apps
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop
- https://caniuse.com/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- https://css-tricks.com/content-jumping-avoid/
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
- https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API



Bildquellen:

- SSR and CSS by Alex Grigoryan - https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8
- Promise flow by https://www.educba.com/javascript-promise/
- PWA Logo By Diego González-Zúñiga - https://github.com/webmaxru/progressive-web-apps-logo, CC0, https://commons.wikimedia.org/w/index.php?curid=86444135
- Layout Shifting examples By https://css-tricks.com/content-jumping-avoid/