Grundlagen der Programmierung von Progressive Web Apps

Einleitung

Alexander Miller
alles.mil@gmail.com

Ausbildung Junior Software-Entwickler
WIFI Salzburg

Frühjahr 2020

Version 23. April 2020

Allgemeines

Informationen zum Kurs

- Abhaltung über Zoom
- Skriptum und Übungen über WIFI Lernplattform oder GitHub unter `https://github.com/allesmi/progressive-web-apps-2020`

Termine

- Do, 23. April: Einführung
- Di, 28. April
- Do, 30. April
- Di, 5. Mai
- Do, 7. Mai
- Di, 12. Mai
- Mi, 13. Mai
- Mi, 20. Mai: Wiederholung
- Mi, 27. Mai: Prüfung

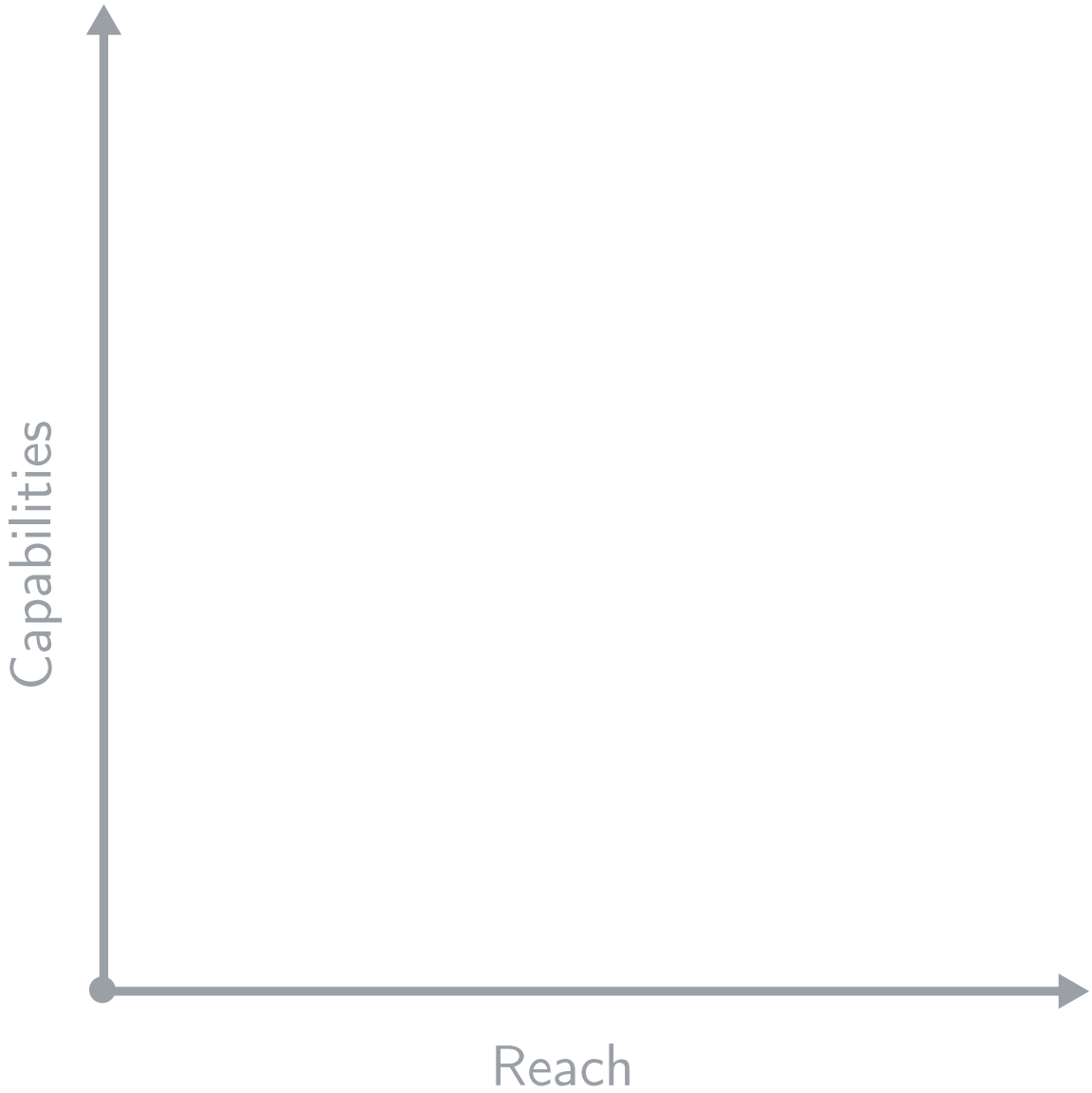
Themen

- Was ist eine progressive Web App (PWA)?
- AJAX
- IndexedDB
- Web Worker
- Service Worker
- Browser und Geräte APIs

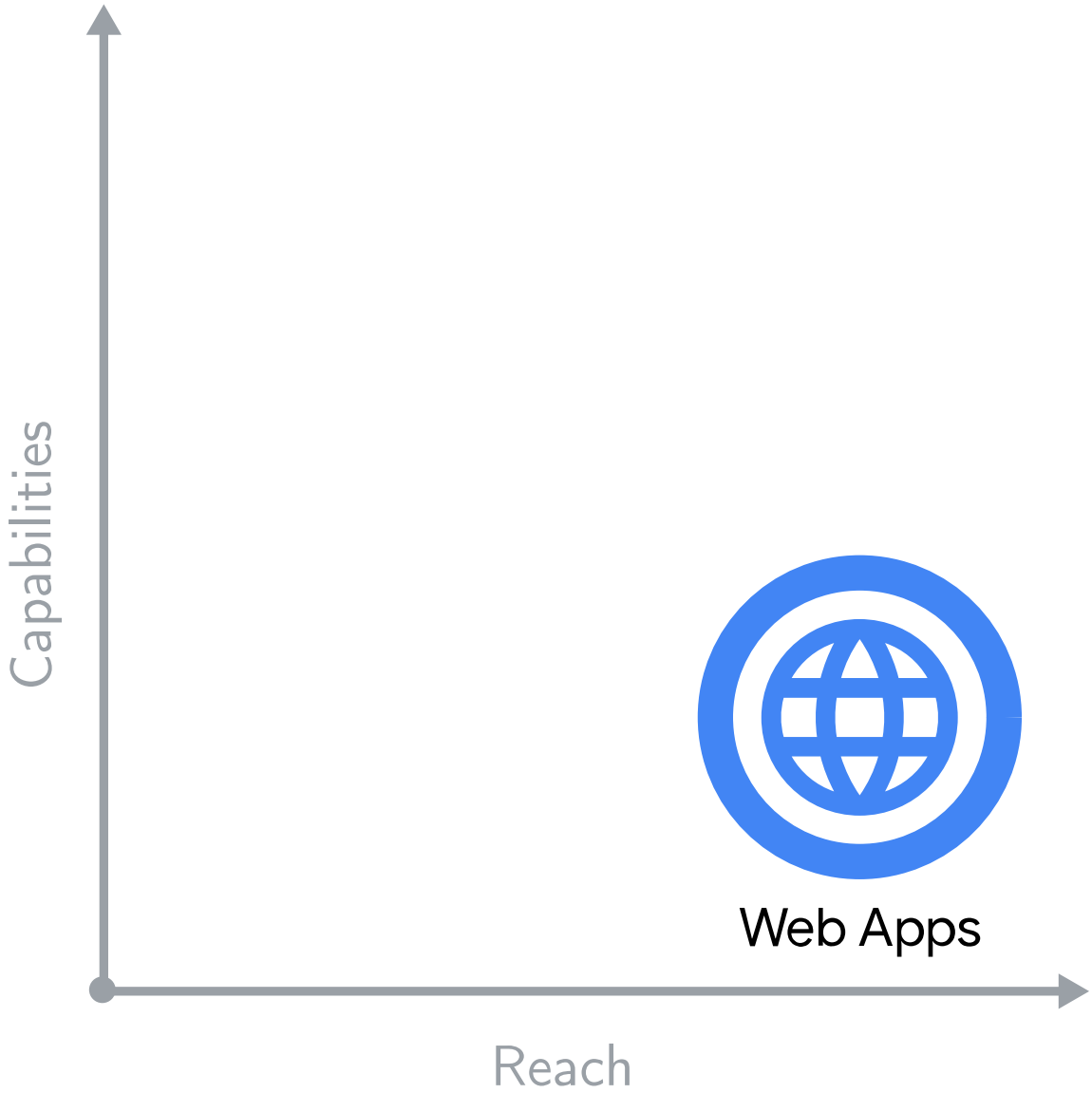
Benötigte Software

- Visual Studio Code
- Google Chrome

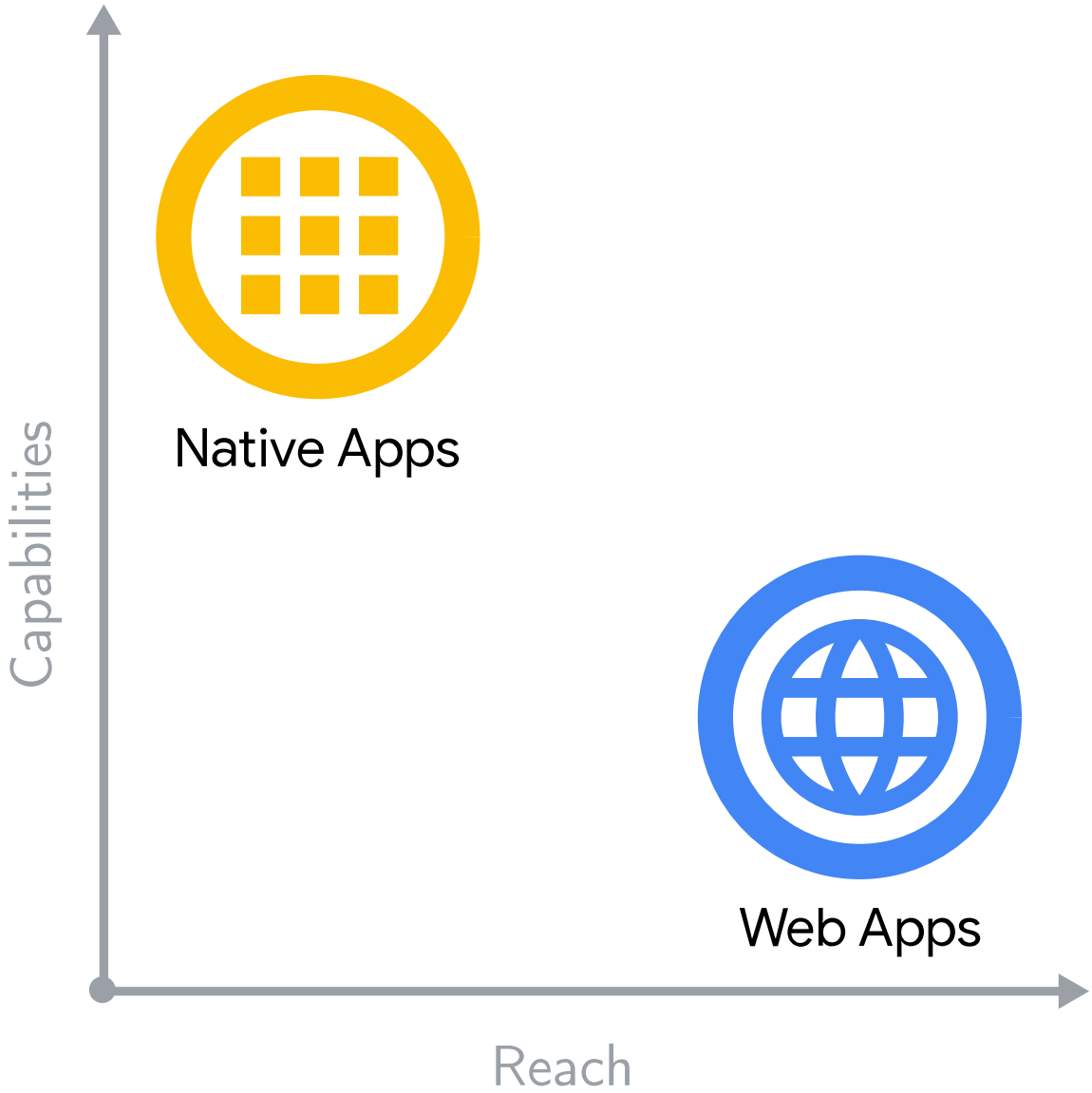
Einführung



Einführung



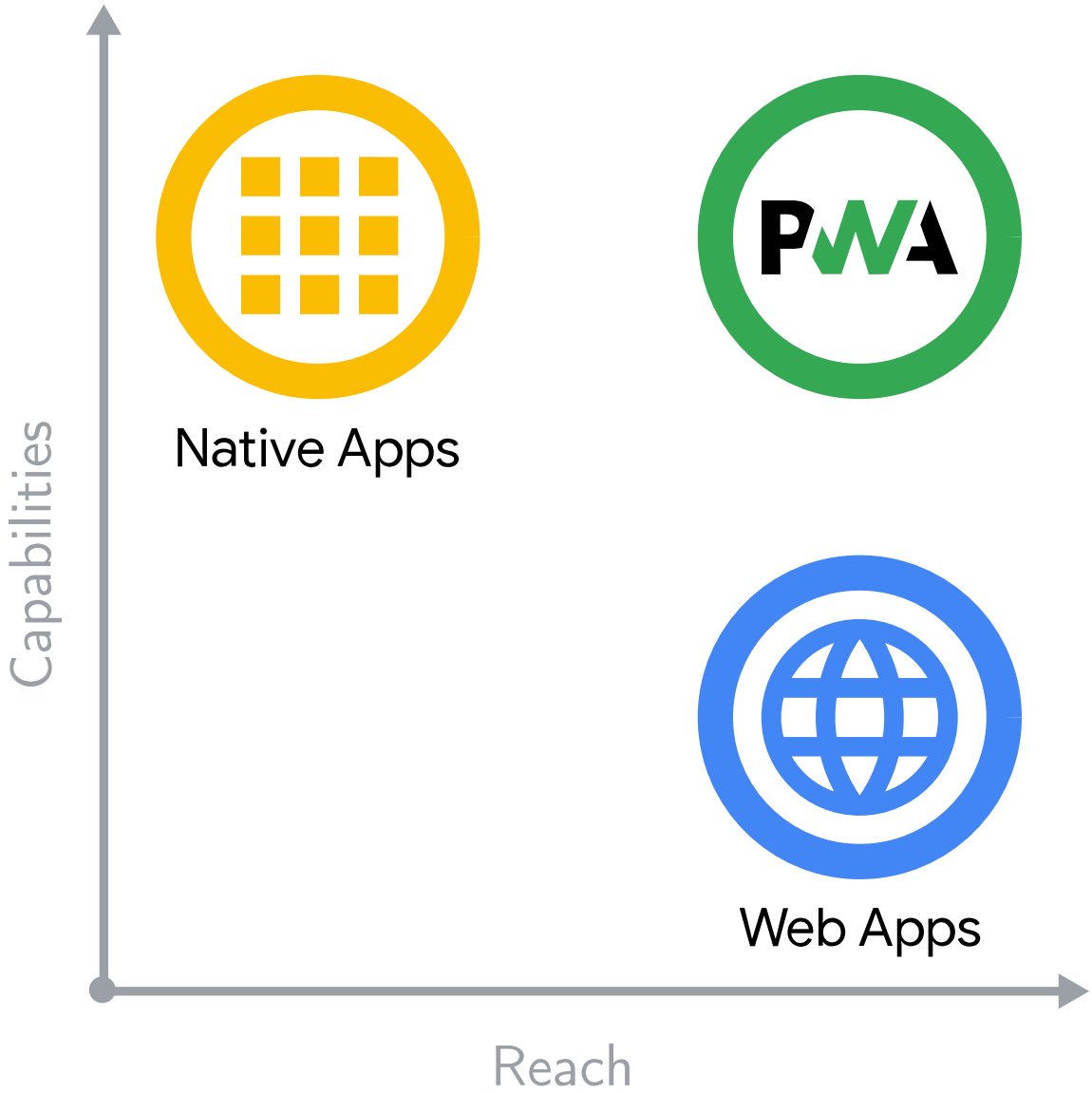
Einführung



Unterschiede

Unterschiede zwischen einer nativen App und einer Web App

Einführung



Eigenschaften

- Großer Funktionsumfang
- Zuverlässig
- Installierbar

JavaScript

Variablen

- Namen für Werte
- Sind nur in ihrem Geltungsbereich (scope) sichtbar

Variablen Verwendung

```
let a; // Deklaration der Variable a

a = 10; // Zuweisung: a wird der Wert 10 zugewiesen

let b = 10; // Das geht auch in einer Zeile.
```

Primitive Typen

```
null; // kein Wert
undefined; // Variable ist deklariert aber hat keinen Wert
true; false; // boolean
10; // number
'text'; // string
```

Zusammengesetzte Typen

```
[]; // leeres Array, auch Liste genannt
[1, 2, 3]; // Array
{}; // leeres Objekt

// Objekt mit den Properties name und age
{ name: 'Anton', age: 12 };

// Zugriff auf Properties von Objekten erfolgt durch den .
o.age;
```

Funktionen

```
// Anonyme Funktion
function (x, y) { return x + y; }

// Eine Funktion namens sum1
function sum1(x, y) { return x + y; }

// Funktionen können wie Werte zugewiesen werden
let sum2 = function(x, y) { return x + y };
```

Zugriff auf DOM-Elemente

```
// querySelector gibt das erste gefundene Element
// im DOM-Baum zurück.
document.querySelector('h1');
document.querySelector('#id');
document.querySelector('.class');

// querySelectorAll gibt eine Liste von gefundenen Elementen
// zurück.
document.querySelectorAll('h2');

// oder
document.getElementById('id');
document.getElementsByClassName('h1');
// ...
```

DOM Events

- Der Browser löst Events aus
- Ursache: Usereingabe, Netzwerk, Sensoren, ...
- Events steigen im DOM-Baum bis zur Wurzel auf

DOM Events Verwendung

```
let button = document.querySelector('button');

button.onclick = function(event) { /**/ };
button.addEventListener('click', function(event){ /**/ });

// Das Event Objekt hat zahlreiche Properties
event.target; // Das Element, wo das Event ausgelöst wurde
```

DOM Events

Event	Bedeutung
“click”	Ein Element wurde mit Maus oder Finger angeklickt
“change”	Der Wert eines Input-Element wurde geändert
...	...

AJAX Wiederholung

- Asynchronous JavaScript and XML
- Kommunikation mit einem Server ohne die Seite neu zu laden
- Empfangen und verarbeiten der Daten

HTTP Request

```
POST / HTTP/1.1
Host: localhost:8000
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Content-Type: text/plain;charset=utf-8
Content-Length: 12

Hallo Server
```

HTTP Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Encoding: gzip
Content-Type: text/plain;charset=utf-8

Hallo Client
```

HTTP Status Code Klassen

Klasse	Bedeutung
200 - 299	Success
400 - 499	Client Fehler
500 - 599	Server Fehler

HTTP Status Codes

Code	Name
200	OK
400	Bad Request
404	Not Found
500	Internal Server Error

HTTP Methoden

Methode	Bedeutung
GET	Lesen einer Resource
POST	Schreiben
DELETE	Löschen

AJAX Methoden

- XMLHttpRequest (IE11 und moderne Browser)
- fetch (moderne Browser)
- jQuery (Library)

XMLHttpRequest

```
let httpRequest = new XMLHttpRequest();

httpRequest.onreadystatechange = function() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    if (httpRequest.status === 200) {
      alert(httpRequest.responseText);
    }
    else {
      alert('Something bad happened');
    }
  }
};
httpRequest.open('GET', 'test.json');
httpRequest.send();
```

fetch

```
fetch('test.json')
  .then(function(response) {
    if (!response.ok) {
      throw new Error('Something bad happened');
    }
    return response.text();
  })
  .then(function(data) {
   alert(data);
  })
  .catch(function(error) {
    alert('Error');
  });
```

JSON

- JavaScript Object Notation
- Serialisiert Daten um sie zu versenden

JSON Beispiel

```
{
  "name": "Hans",
  "age": 12,
  "active": true,
  "friends": [
    "Franz",
    "Max"
  ]
}
```

JSON

```
// Von JSON-Text zu JavaScript Objekt
object = JSON.parse(jsonString);

// Von JavaScript Objekt zu JSON Text
jsonString = JSON.stringify(object);
```

Same-Origin Policy

- Zwei URLs haben denselben Ursprung wenn Protokoll, Host und Port übereinstimmen
- AJAX Aufrufe sind nur für URLs mit dem selben Ursprung erlaubt wie das HTML Dokument

Same-Origin Policy

- Zwei URLs haben denselben Ursprung wenn Protokoll, Host und Port übereinstimmen
- AJAX Aufrufe sind nur für URLs mit dem selben Ursprung erlaubt wie das HTML Dokument
- Ausser die Antwort vom anderen Server erlaubt das mittels CORS
- Cross-Origin Resource Sharing
- HTTP Header: `Access-Control-Allow-Origin: *`