Teil IV
Lokale Datenbanken

Lokale Datenbanken
Übersicht

Was? Wie? Warum?

Was? Wie? Warum?

Client-Side

Wir speichern Daten beim Client, und rufen Sie ab, wenn wir sie brauchen.

Performance

Wenn Daten lokal gespeichert werden können, müssen sie nicht immer neu geladen werden. Das spart Ladezeit.

User Experience

Bietet zusätzliche UX-Möglichkeiten (Shopping Cart der letzten Session, Eingeloggt bleiben usw.).

Nutzerspezifisch

Als Beispiel können Seiteneinstellungen für die Darstellung lokal pro Nutzer und Gerät abgelegt werden.

On- und Offline

"*Because it lets you create web applications with rich query abilities regardless of network availability, your applications can work both online and offline.*"

Digitaler Perso

Können Client über kleine Speicherdaten wiedererkennbar für den Server machen.

Lokale Datenbanken
Cookies

Mehr Kekfe!

Cookies - Was ist das?

Fortune Cookie

Name kommt vom **Glückskeks**, das eine kleine Nachricht enthält.

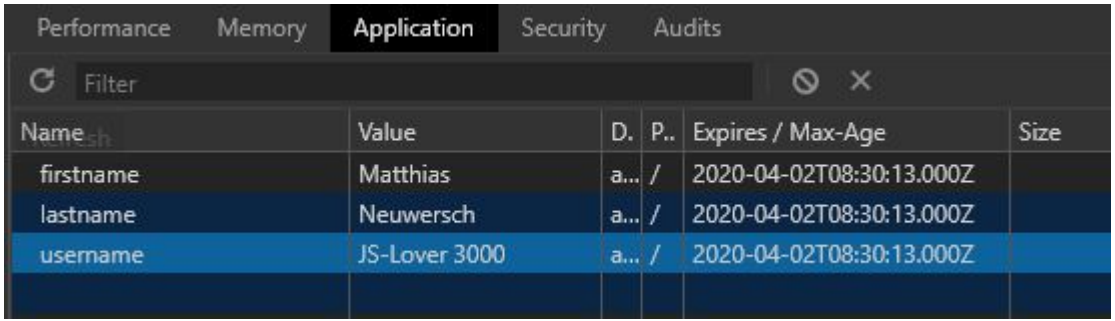
Kleiner Speicher

Dient zur Speicherung, kleiner Infos, wie z.B. der **bevorzugten Sprache**

Befristet

Cookies haben ein **Ablaufdatum**. Danach verfallen sie einfach.

Cookies in echt



Aufruf mit:

```
document.cookie
```

Ergebnis:

```
"firstname=Matthias; lastname=Neuwersch; username=JS-Lover 3000"
```

Cookies - Limits

Folgende Limits können sich je nach Browser ein wenig unterscheiden:

- Meistens als **Klartext** gespeichert (security)
- In ihrer **Größe beschränkt** (meist 4kB)
- **Anzahl** an Cookies pro Domain **beschränkt** (meist 20)
- Können vom User **deaktiviert/gelöscht** werden.
- Ermöglicht **nur Plain-Text** Daten.
- **Werden** mit jedem HTTP Request **mitgesendet**.

Lokale Datenbanken
Local Storage

Kekfe 2.0

Local Storage - Was ist das?

Key-Value Store

Kann **Schlüssel-Wert** Paare abspeichern (bis zu 5MB).

Synchron

Blockiert den Main-Thread, sollte also mit Bedacht verwendet werden.

Unbefristet

Daten im Local Storage haben **kein Ablaufdatum**.

Local Storage in echt

- `setItem()`: Key/Value Paar hinzufügen
- `getItem()`: Value für einen Key auslesen
- `removeItem()`: Key/Value Paar entfernen
- `clear()`: Den ganzen Storage leeren
- `key()`: Gibt den Key an der n-ten Stelle zurück

Local Storage in echt

```
window.localStorage.setItem('name', 'James L. Essig');
window.localStorage.getItem('name');
window.localStorage.removeItem('name');
window.localStorage.clear();
let keyName = window.localStorage.key(index);
// ODER
localStorage.name = 'James L. Essig';
```

Objects?

```
//WRITE
let person = {
   name: "James L. Essig",
   location: "Austria",
};
localStorage.setItem('user', JSON.stringify(person));
localStorage.user = JSON.stringify(person);


//READ
person = JSON.parse(window.localStorage.getItem('user'))
person = JSON.parse(localStorage.user)
```

Lokale Datenbanken
Session Storage

“Das mit uns ist nichts Langfristiges.”

Session Storage - Was ist das?

Wie Local Storage

Nur auf einen Tab gebunden. Wenn die Session beendet wird, werden die Daten gelöscht.

Limits

Im Session Storage können bis zu 5MB gespeichert werden.

Praktischer Zwischenspeicher

z.B. zum Übertragen von Informationen zwischen mehreren Views

Session Storage

Use cases

- Als globaler Zwischenspeicher für ein Objekt, das in mehreren Views bearbeitet wird.
- Den Status (bsp. geöffneter Reiter) von Interface-Elementen merken
- Für “Once per Session” Aktionen (z.B. News-Popup)

Session Storage in echt

```
window.sessionStorage.setItem('name', 'James L. Essig');
window.sessionStorage.getItem('name');
window.sessionStorage.removeItem('name');
window.sessionStorage.clear();
let keyName = window.sessionStorage.key(index);
// ODER
sessionStorage.name = 'James L. Essig';
```

Lokale Datenbanken

~~WebSQL~~

Just don't.

Lokale Datenbanken
IndexedDB

And the Award goes to...!

IndexedDB - Was ist das?

Großer Object Store

Speichert und gibt komplexe JavaScript **Objekte** zurück. Je nach System können auch mehrere GB gespeichert werden.

Der Index machts

Benutzt Indizes für **hochperformante** Suchen

Asynchron

Operationen **blockieren** also **nicht** den Main-Thread.

IndexedDB - Was ist das?

Lokale NoSQL Datenbank

Ist **nicht** als relationale Datenbank wie MySQL zu verstehen.

Same Origin Policy

Sicherheitsprotokoll, das dafür sorgt, dass Resourcen einer Origin (Domain) zugeordnet werden.

IndexedDB 2.0

W3C Recommendation seit 30.01.2018

IndexedDB - In echt

- Besteht in der Regel aus mehreren **ObjectStores** (wie Tables)
- Jeder Store kann mehrere **Objekte** enthalten (wie Rows)
- Die Objekte sind nach ihrem **Key** aufsteigend sortiert.

IndexedDB - In echt

- Keys können sein: **`string`**, **`date`**, **`float`**, **`binary blob`** oder **`array`**
- Values können sein: **`boolean`**, **`number`**, **`string`**, **`date`**, **`object`**, **`array`**, **`regexp`**, **`undefined`** oder **`null`**.

IndexedDB - In echt

Basis-Schritte, um etwas in der IndexedDB zu tun:

1. Datenbank öffnen `window.indexedDB.open(dbName, dbVersion);`

   Wenn Datenbank nicht vorhanden, wird sie angelegt.

2. Objectstore anlegen `db.createObjectStore("favourite_hotels", {keyPath: "_id"});`
3. Transaktion starten und Daten abfragen/einfügen.

```
request = db.transaction(["favourite_hotels"], 'readonly').objectStore("favourite_hotels").get(key);
```

4. Asynchron: Warten bis das Ergebnis da ist.

```
request.onsuccess = function(){
    callback(request.result);
};
```

IndexedDB - In echt

```
const dbName = "kwm-store";
const dbVersion = 1;


export default class KWM_Model {
   constructor() {
      this.idb = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;
   }
...

}
```

IndexedDB - In echt

```
read(objectStoreName, callback){
  let request = this.idb.open(dbName, dbVersion);
  request.onsuccess = function(e){
      let db = e.target.result;
      let request = db.transaction([objectStoreName], 'readonly').objectStore(objectStoreName).getAll();
      request.onsuccess = function(){
          callback(request.result);
      };
      request.onerror = function(e){
          console.error(e);
      };
  };
  request.onerror= function(e){
      console.error(e);
  };
  request.onupgradeneeded = function(e){
    KWM_Model.upgradeDB(e);
  }
}
```

IndexedDB - In echt

```
static upgradeDB(e){
   let db = e.target.result;
   db.createObjectStore("favourite_pets", {keyPath: "_id"});
}
```

CSS Preprocessors

Übersicht

Preprocessors - Was ist das?

"CSS Upgrade"

Erweitert CSS um praktische Funktionen

Kompiliert

Ergebnis wird in eine .css Datei kompiliert

Sass / Less

Die beiden relevanten Vertreter

Preprocessors - Was können sie?

- Variablen
- Mathematische Operationen
- Funktionen
- Schleifen
- Verschachtelungen
- Mixins

In echt

```
@appColor: #C19B76;
@images: "../img";
.background-image-mixin(@url){
  background-image: url(@url);
  background-size: contain;
  background-repeat: no-repeat;
  background-position: center center;
}
#app_header{
  background-color: @appColor;
  .logo{
    Background-image: ("@{images}/logo.png");
    width: 100px;
    height: 100px;
  }
}
```

```
#app_header {
  background-color: #C19B76;
}
#app_header .logo {
  background-image: url("../img/logo.png");
  background-size: contain;
  background-repeat: no-repeat;
  background-position: center center;
  width: 100px;
  height: 100px;
}
```

Sass oder Less?

Syntactically **A**wesome **S**tyle **S**heets

Ruby

Scss / Sass Syntax

Berühmt durch Bootstrap 4

Leaner **S**tyle **S**heets

JavaScript

Scss Syntax (Superset of css)

Berühmt durch Bootstrap 1

Es bleibt Geschmackssache

Variablen

```
@appColor: #C19B76;
@images: "../img";

#app_header{
   background-color: @appColor;
}
.logo{
     Background-image: ("@{images}/logo.png");
}
```

```
#app_header {
   background-color: #C19B76;
}
.logo {
   background-image: url("../img/logo.png");
}
```

Verschachtelung

```
#app_header{
   background-color: @appColor;
   .logo{
     Background-image: ("@{images}/logo.png");
     width: 100px;
     height: 100px;
   }
}
```

```
#app_header {
   background-color: #C19B76;
}
#app_header .logo {
   background-image: url("../img/logo.png");
   width: 100px;
   height: 100px;
}
```

Parent Selector “&”

```
#app_header{
   background-color: @appColor;
   .logo{
     Background-image: ("@{images}/logo.png");
     width: 100px;
     height: 100px;
     &:hover{
           border: 1px solid black;
     }
   }
}
```

→

```
#app_header {
   background-color: #C19B76;
}
#app_header .logo {
   background-image: url("../img/logo.png");
   width: 100px;
   height: 100px;
}
#app_header .logo:hover {
   border: 1px solid black;
}
```

Mixins

```
.background-image-mixin(@url){
 background-image: url(@url);
 background-size: contain;
 background-repeat: no-repeat;
 background-position: center center;
}
.logo{
   .background-image-mixin("../img/logo.png");
   width: 100px;
   height: 100px;
}
```

```
.logo {
   background-image: url("../img/logo.png");
   background-size: contain;
   background-repeat: no-repeat;
   background-position: center center;
   width: 100px;
   height: 100px;
}
```

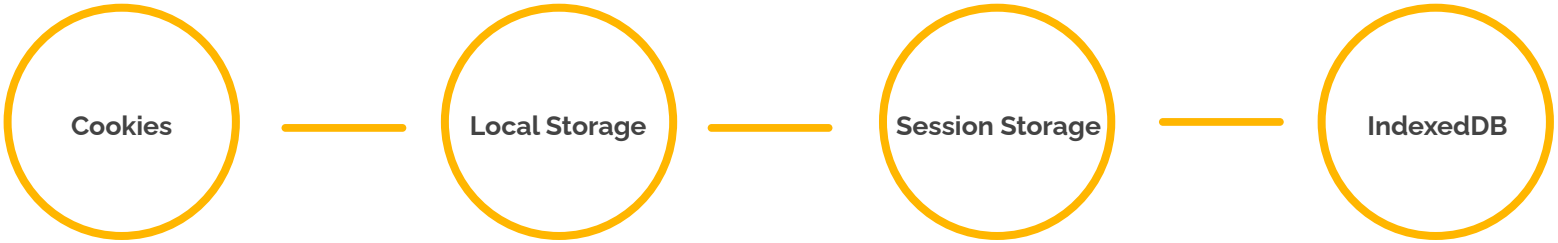
Less - Installation

1. Node.js installieren
2. Terminal:

```
npm install --global less
```

3. Create File Watcher

Auf einen Blick



Kleiner Textspeicher

4KB Reintext, wird bei jedem Request **übermittelt**.

Key-Value Speicher

Bis zu **5MB** zeitlich unbegrenzt. Operationen werden **synchron** durchgeführt. Arbeitet nur mit **Strings**.

Speicher auf Zeit

Genau **wie Local Storage,** nur eben **auf** eine **Session begrenzt**. Wird die Session beeendet, wird der Speicher **geleert**.

Großer, komplexer Speicher

Lokale **NoSQL** Datenbank, versteht sich ebenso auf **Key-Value Pairs**. Kann JavaScript **Objekte** lesen und schreiben. **Same Origin Policy!**