Angular: Progressive Web Apps

- Vorteile und Eigenschaften von PWAs einordnen können
- Begriff des Service Workers einordnen können
- Verstehen, dass eine einzige App für das Web, als Desktop- oder Smartphone-App geschrieben werden kann
- Begriffe Mobile- und Offline-First einordnen können
- Design-Patterns die in PWAs eingesetzt werden verstehen
- Eine Angular App zu einer PWA umformen können
- Service Worker und Installierbarkeit konfigurieren können
- Den Deploy-Vorgang einer PWA verstehen und anwenden können
- Mit IndexedDB Daten im Browser offline halten können

Literaturhinweis

Progressive Web Apps, Das Praxisbuch, Christian Liebel, Rheinwerk Computing, ISBN 978-3-8362-6494-5

Was können Progressive Web Apps?

Web App soll so aussehen und sich so verhalten wie native Anwendung

- *Offlinefähigkeit*
  Quelldaten der App befinden sich auf dem Gerät, Benutzerdaten werden dort zwischengespeichert

Single-Page-Anwendung

IndexedDB

- *Push-Benachrichtigungen*
  auch dann wenn die App oder der Browser nicht geöffnet sind

Service Worker

- *Home-Bildschirm und Fensterdekoration*
  Eigenes Icon, im Taskmanager, eigenes Fenster ohne Adressleiste und Menü des Browsers

Web App-Manifest

PWAs können nur jene Schnittstellen verwenden die auch der Browser verwenden kann (Audio, Video, Canvas, Gamepad API zur Ansteuerung von Spiele-Controllern, Web Speech API, Zugriff auf Kamera und Mikrofon, Zugriff auf Gerätesensoren, Geolocation, usw.)

Mobile First | **Offline First**

Design-Patterns für Progressive Web Apps

1. *Progressive*[1]
   Anwender älterer Browser sollten nicht ausgeschlossen werden
2. *Responsive*
   PWAs passen sich unterschiedlichen Geräten an
3. *Connectivity Independent*
   PWAs können dank Service Worker und dessen Zwischenspeicherung auch offline ausgeführt werden
4. *App-like Interactions*
   PWAs setzen App-typische Navigationsstrukturen und Interaktionsmodelle ein
5. *Fresh*
   PWAs verwenden online ausgeführt trotz Offlinekopie im Zwischenspeicher immer die neueste Variante. Das stellt Service Worker-Updateprozess sicher
6. *Safe*
   PWAs werden über HTTPS ausgeführt
7. *Discoverable*
   PWAs sind als Anwendungen identifizierbar, können von gewöhnlichen Websites unterschieden werden und deshalb mit besonderen Eigenschaften ausgestattet werden (Manifest-Datei)
8. *Re-engageable*
   PWAs können Anwender durch Push-Benachrichtigungen zum Wiederverwenden der Anwendung bewegen
9. *Intallable*
   PWAs können auf dem Home-Bildschirm bzw. in der Programmleiste des Anwenders hinterlegt werden
10. *Linkable*
    PWAs müssen nicht installiert werden, laufen sofort und lassen sich mithilfe der URL problemlos mit anderen teilen

---

[1] Firefox und Safari unterstützen PWAs derzeit nicht auf Desktop

PWA und Angular

Notwendige Tools

- *Visual Studio Code*
- *Git evtl. mit GitHub*
- *Google Chrome*
- *Node.js*
- *Angular CLI* (Abk. Command Line Interface)
- *lite-server* (zum Testen auf Mobilgeräten)
  startet lokalen Web-Server im Arbeitsverzeichnis
- *ngrok* (zum Testen auf Mobilgeräten)
  Tunnel für Netzverkehr. Stellt eigentlich nur lokal erreichbare Endpunkte unter einer öffentlich erreichbaren Adresse auch über https zur Verfügung (`http://ngrok.com/download`)

1. Schritt: Angular-Projekt anlegen

```
$ ng new PWA-Notes --prefix=no --routing=true --style=scss
$ cd PWA-Notes
```

Integriert das Routing und das Stylesheet-Format SCSS

2. Schritt: Angular Material und Flex-Layout importieren

```
$ ng add @angular/material
$ npm install @angular/flex-layout
```

Setzt das Responsive Web-Design um

3. Schritt: Integriere Navigationskomponente

```
$ ng generate @angular/material:material-nav --name nav
```

Von Angular Material bereitgestellte Komponente `material-nav` wird in die Anwendung integriert. Komponente wird im Ordner nav abgelegt und kann über <no-nav> integriert werden

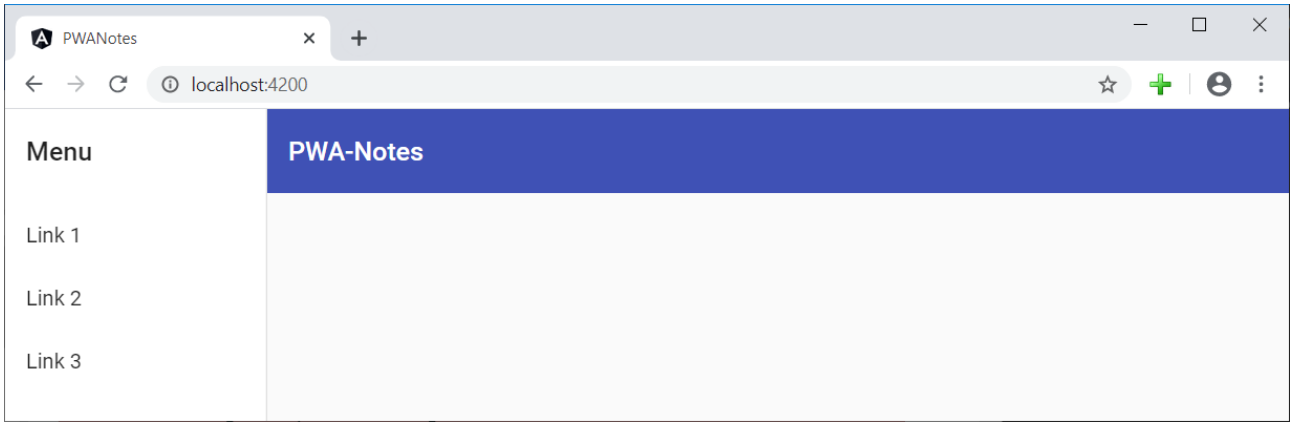
Änderung der Datei app.component.html

```
<no-nav>
  <router-outlet></router-outlet>
</no-nav>
```

Änderung der Datei nav.component.html

Ersetzen von `<!-- Add Content here -->` durch `<ng-content></ng-content>`

Ergebnis



4. Schritt: PWA-Unterstützung installieren

```
$ ng add @angular/pwa
```

Erzeugt Web App-Manifest und integriert Service Worker-Unterstützung

5. Schritt: Anpassen des Web App-Manifest

```
{
  "name": "My PWA-Notes",
  "short_name": "PWA-Notes",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "assets/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    …
  ]
}
```

`fullscreen`
alle Browser- und Betriebssystemsteuerelemente und das Fenster werden ausgeblendet (für Spiele)

`standalone`
App läuft im Fenster mit Symbol in Taskleiste aber ohne Browser-Steuerelemente. `theme-color` bestimmt die Farbe (für Business-Apps)

`minimum-ui`
wie `standalone` kann aber Browser-Steuerelemente anzeigen

`browser`
Anwendung läuft im Browser-Fenster

`start_url`
Route welche in der App beim Öffnen angesprungen werden soll

`scope`
Nur URLs im angegebenen Scope können von der App angesprungen werden (erhöht Sicherheit der Anwendung)

HINWEIS
Das Manifest der App kann in Chrome in der Entwicklerconsole unter **Application** eingesehen werden

6. Schritt: Service Worker anpassen, Quelldaten werden offlinefähig

Service Worker wird durch 4. Schritt aktiviert (siehe Datei `app.module.ts`) aber nur dann, wenn die Angular-Anwendung ein *produktiver Build* ist (siehe hinten)

Konfiguration ändern in Datei `ngsw-config.json`

`assetGroups`
regelt das *statische Caching* von Anwendungsquelldateien, Bilder, Schriftarten, usw. *mit* Versionsverwaltung

`dataGroups`
regelt das *dynamische Caching* von Anfragen an einen API-Endpunkt, Web-Services, usw. und ist unabhängig von Versionsverwaltung

```
{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico", "/index.html", "/*.css", "/*.js"
        ],
        "urls": [
          "https://fonts.googleapis.com/**",
          "https://fonts.gstatic.com/**"
        ]²
      }
    }, {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/assets/**", "/*.(eot|svg|cur|jpg|png|…|woff2|ani)"
        ]
      }
    }
  ],
  "dataGroups": [
    {
      "name": "wetter-daten",
      "urls": ["https://daten.buergernetz.bz.it/services/**"],
      "cacheConfig": {
        "strategy": "freshness",
        "timeout": "3s",
        "maxAge": "3d12h10s",
        "maxSize": 10
      }
    }
  ]
}
```

[2] Muss ergänzt werden, da Material-Schriftarten und -Icons über Google Dienst Fonts bezogen werden. Bei fehlender Internetverbindung könnten Schriftarten nicht abgerufen werden und folglich App nicht offline starten

`index`
Einsprungspunkt der Anwendung

`prefetch`
Ressourcen werden bei Neuinstallation/Update sofort heruntergeladen und zwischengespeichert auch dann, wenn sie nicht angefordert werden

`lazy`
Ressourcen werden bei Neuinstallation/Update erst heruntergeladen und zwischengespeichert, wenn Benutzer sie anfordert

`strategy: freshness | performance`
`freshness` (Network falling back to cache) zuerst wird versucht Anfrage über das Netz zu bedienen. Ist Anwender offline oder tritt Zeitüberschreitung (`timeout`) ein so wird auf Cache zurückgegriffen
`performance` (Cache falling back to network) zuerst wird im Cache nach der Anfrage gesucht. Ist dort nichts zu finden wird über das Netz Anfrage beantworten

`maxAge`
bestimmt wie lange Anfrage im Cache zwischengespeichert wird

`maxSize`
wie viele Einträge soll der Cache für die Datengruppe maximal zwischenspeichern

7. Schritt: Produktiven Build erstellen und starten

PROBLEME

1. Nur produktiver Build der Angular-Anwendung enthält *aktivierten Service Worker*
2. Build muss über *https* zur Verfügung gestellt werden.

Deshalb…

```
$ ng build --watch                                    im ersten Terminal
```

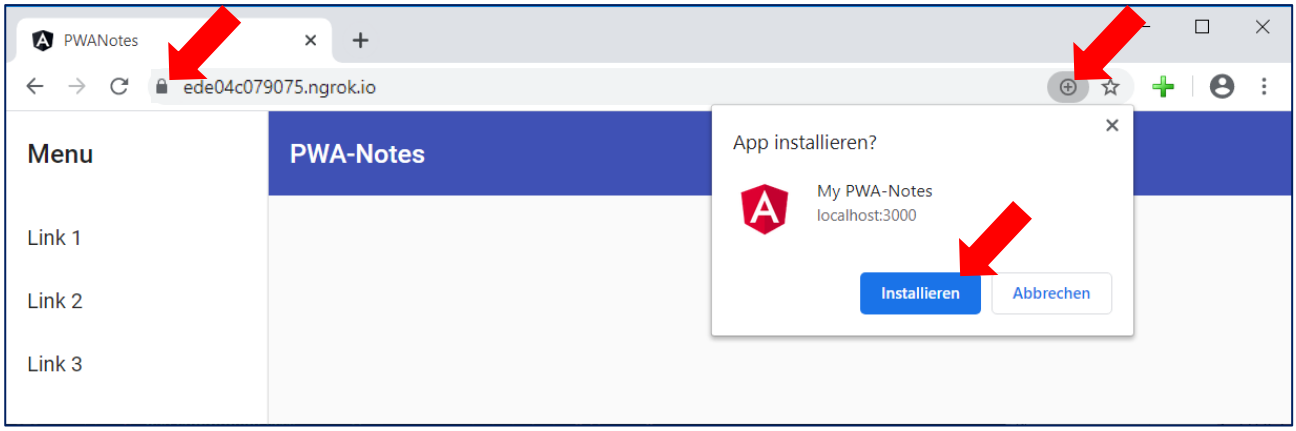
Buildausgabe wird in den Ordner `dist\PWA-Notes` kopiert

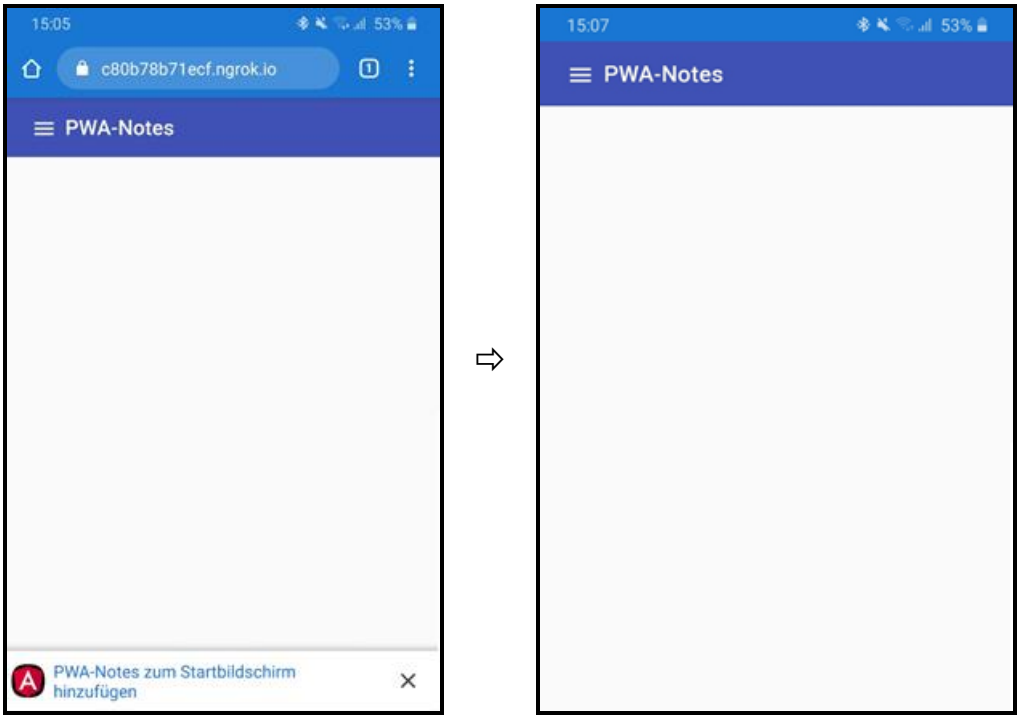
Installation und Starten des *Lite-Servers*

```
$ npm install lite-server --global                    im zweiten Terminal
$ lite-server --baseDir="dist/PWA-Notes"
```

Auf Mobilgerät testen mit *ngrok*

```
$ npm install ngrok --global                          im dritten Terminal
$ ngrok http <Port des Lite-Servers>
```





Auch dann, wenn keine Internetverbindung besteht kann App gestartet werden

ACHTUNG: App muss über *https* zur Verfügung gestellt werden!

IndexedDB (Abk. Indexed Database API)

JavaScript-basierte, objektorientierte Datenbank zur Speicherung strukturierter Daten im Browser

- Arbeitet auch ohne Service Worker, dieser kann auf Datenbanken zugreifen
- Speicherplatz beschränkt je nach Browser (Chrome erlaubt Belegung von 6%, Firefox 10% des freien Festplattenspeichers pro App)[3]
- Speichert Objekte mit unterschiedlichen Eigenschaften ab, gesucht wird über Indexe (Objektspeicher)

Dexie.js

Wrapper für IndexedDB

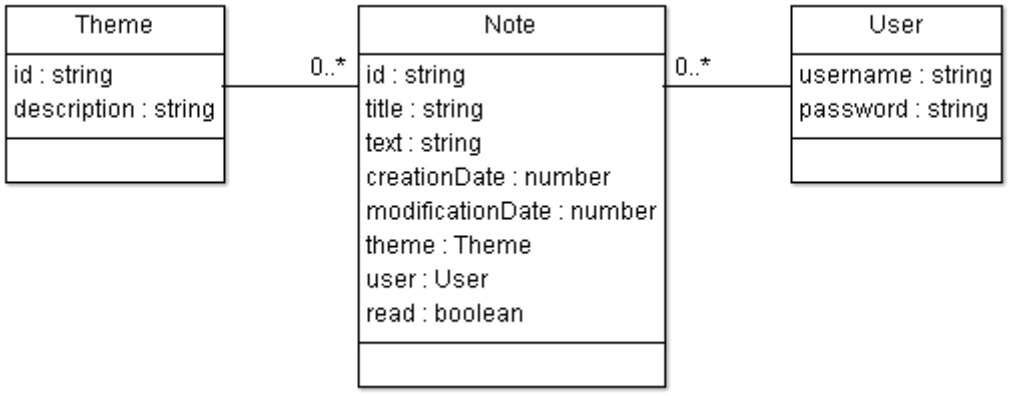
`https://dexie.org/docs`

```
$ npm install dexie

$ npm install uuid
$ npm install @types/uuid[4]


$ npm install moment
```

zur Generierung von Primärschlüsseln (Abk. Universally Unique Identifier)

zur Datumsmanipulation

Klassendiagramm



---

[3] Abfragbar an beliebiger Stelle durch

`navigator.storage.estimate().then(storage=>console.log(storage));`

[4] Dieses Paket enthält die Typdefinitionen von *uuid* und muss importiert werden

```
import Dexie from 'dexie';
import { v4 as uuidv4 } from 'uuid';
import * as moment from 'moment';

export class DbService extends Dexie {

  private notes!: Dexie.Table<Note, string>;
  private themes!: Dexie.Table<Theme, string>;

  constructor() {
    super('notes-db[5]');
    this.version(1[6]).stores({
      notes: 'id[7], title, [theme.description+title][8], theme.id,
        [modificationDate+creationDate]',
      themes: 'id, &[9]description'
    });

    this.notes.mapToClass(Note);[10]
    this.themes.mapToClass(Theme);

    this.on('populate[11]', async () => {
      try {
        const t1 = new Theme(uuidv4(), 'Bananen');
        const t2 = new Theme(uuidv4(), 'Birnen');
        const t3 = new Theme(uuidv4(), 'Ananas');
        await this.themes.bulkAdd[12]([t1, t2, t3]);

        const u = new User('sepp@hintner.com', 'sepp');

        const n1 = new Note(uuidv4(), 'Titel1', 'Text1',
          moment().valueOf(), 0, t2, u, false);

        const n2 = new Note(uuidv4(), 'Titel2', 'Text2',
          moment().valueOf(), 0, t1, u, false);

        await this.addNote(n1)[13];
        await this.addNote(n2);

      } catch (err) {
        console.log(err);
      }
    });
  }

  // Fortsetzung übernächste Seite
```

---

[5] Datenbankname

[6] Versionsnummer, damit können Datenbank-Updates erkannt und durchgeführt werden

[7] Erster Eintrag entspricht dem Primärschlüssel, es folgend weitere Indexe nach denen gesucht werden kann

[8] Zusammengesetzter Index zum Suchen bzw. Sortierung über mehrere Eigenschaften, **ACHTUNG:** + ohne Leerzeichen

[9] Eindeutiger Index

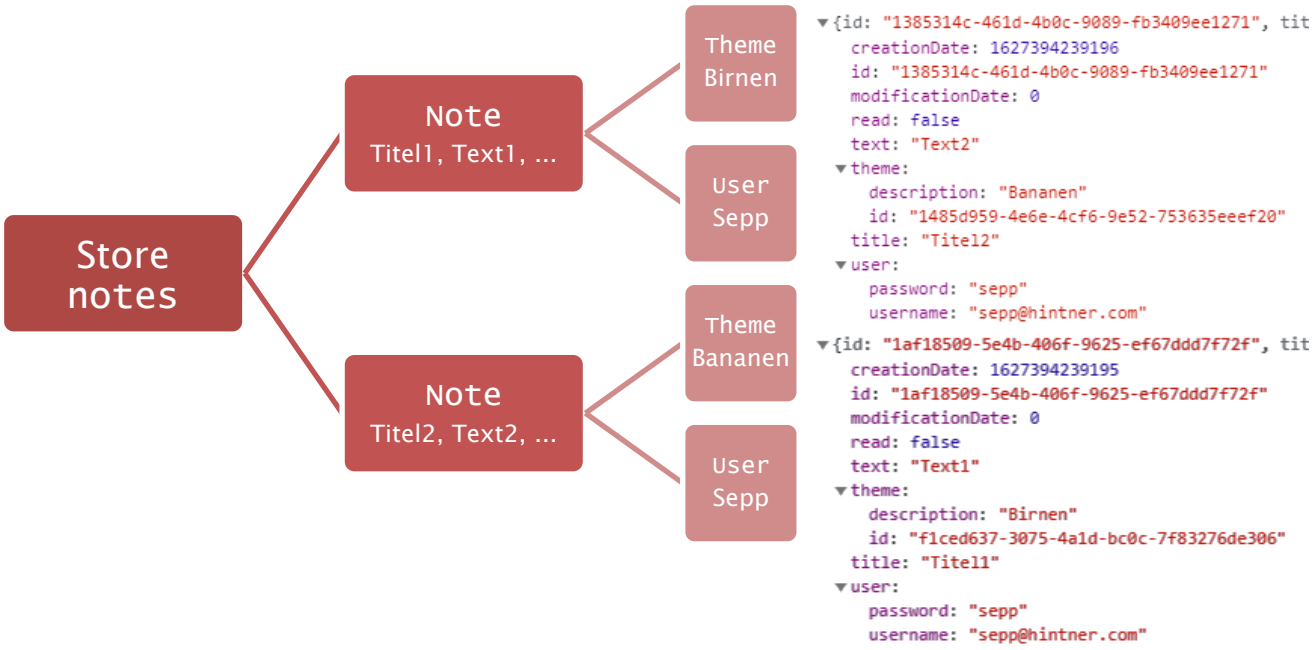
[10] Dadurch liefert Service Objekte vom Typ `Note` bzw. `Theme` zurück

[11] Wird nur dann aufgerufen, wenn Datenbank neu angelegt wird

[12] Sämtliche Abfragemethoden liefern *Promise* (siehe letzten Exkurs) zurück

[13] Siehe hinten

Inhalt des Objekt-Stores notes[14]



Indexe sortieren gespeicherte Objekte nach

- `id`
- `title`
- `[theme.description+title]`
- `theme.id`
- `[modificationDate+creationDate]`

und lassen *schnelle Suche und Sortierung* zu, z.B.

```
this.notes
  .where('theme.description')
  .equals('Bananen')
  .sortBy('title');
```

Sucht alle Notizen deren Themenbeschreibung Bananen ist und sortiert diese nach Titel der Notiz

```
this.notes
  .orderBy(
    '[theme.description+title]'
  )
  .reverse()
  .toArray();
```

Liefert alle Notizen sortiert nach Themenbeschreibung und bei gleicher Beschreibung nach Titel in umgekehrter Reihenfolge

Genaue Verwendung folgt…

---

[14] In der *Entwicklerkonsole* von Chrome einseh- und löschbar unter `App.Speicher.IndexedDB.notes-db`

```
  // Fortsetzung vorvorherige Seite
  async getThemesByDescription() {
    return this.themes.orderBy('description').toArray();
  }
  async getThemeByDescription(description: string) {
    const theme = await this.themes
      .where('description')
      .equals(description)
      .first();
    return theme ? theme : Promise.reject('Theme not found');
  }
  async getNotesByTheme() {
    return this.notes
      .orderBy('[theme.description+title]')
      .reverse()
      .toArray();
  }
  async getNotes(description: string) {
    return this.notes
      .where('theme.description')
      .equals(description)
      .sortBy('title');
  }
  async addTheme(theme: Theme) {
    return this.themes.add(theme);
  }
  async deleteTheme(theme: Theme) {
    const notes = await this.getNotes(theme.description);
    return notes.length ? Promise.reject('Theme in use') :
      this.themes.delete(theme.id);
  }
  async updateTheme(theme: Theme) {
    await this.themes.update(theme.id, theme);
    return this.notes
      .where('theme.id')
      .equals(theme.id)
      .modify({
        'theme.description': theme.description,
        modificationDate: moment().valueOf()
      });
  }
  async addNote(note: Note) {
    note.creationDate = moment().valueOf();
    note.modificationDate = 0[15];
    return this.notes.add(note);
  }
}
```

[15] Datum muss auf 0 gesetzt werden, denn Notizen mit leerem Änderungsdatum (`null`) werden im Index `[modificationDate+creationDate]` nicht mitberücksichtigt und fallen weg

Verwendung

```
const theme: Theme = new Theme(uuidv4(), 'Banane');

this.dbService.addTheme(theme)
  .then(res => {
    console.log('Erfolgreich hinzugefügt');
    theme.description = 'Banana';

    this.dbService.updateTheme(theme)
      .then(res => {
        console.log('Erfolgreich geändert');

        this.dbService.deleteTheme(theme)
          .then(res => console.log('Erfolgreich gelöscht'));
      });
  })
  .catch(err => console.log(err));

this.dbService.getThemeByDescription('Birnen')
  .then(res =>
    this.dbService.deleteTheme(res)
      .then(res => console.log('Erfolgreich gelöscht'))
  )
  .catch(err => console.log(err));

this.dbService.getThemeByDescription('Birnen')
  .then(res => {
    res.description = 'Birnem';

    this.dbService.updateTheme(res)
      .then(res => console.log('Erfolgreich geändert'));
  })
  .catch(err => console.log(err));
```

1

2

3

```
Erfolgreich hinzugefügt
Theme in use
Erfolgreich geändert
Erfolgreich gelöscht
Erfolgreich geändert
```

ACHTUNG: Alle drei Methoden werden nacheinander gestartet ohne auf das Ergebnis zu warten, wegen async.

Ausgabe kann deshalb variieren!!!