



1. Erstellen Sie ein *neues Angular-Projekt* ohne Routing, integrieren Sie *Angular Material*, und erstellen Sie die Klasse *BookStoreService* laut Ausführungen. Es sollen folgende Methoden zur Verfügung gestellt werden:

```
resetStore()      getAllBooks()      getAllBooksSearchTerm(searchTerm)
createBook(book)  deleteBook(isbn)   getBook(isbn)
updateBook(book)  checkBook(isbn)    rateBook(isbn, rating)
```

Die Dateien *book.ts*, *thumbnail.ts* und *book-factory.ts* werden zur Verfügung gestellt.

Entwickeln Sie dann die *Komponente* *BookStoreServiceTestComponent*, welche die nachfolgende Benutzerschnittstelle aufweist und anhand welcher der *BookStoreService* getestet werden kann.

DELETE	/books		Resets store to initial state
GET	/books		Get all books
GET	/books/search/{searchTerm}	searchTerm	Get all books matching search term
POST	/book		Creates a new (random) book Creates a new (random) book with existing ISBN
DELETE	/book/{ISBN}	ISBN	Deletes a book
GET	/book/{ISBN}	ISBN	Returns a single book by ISBN
PUT	/book/{ISBN}	ISBN	Updates an existing book
GET	/book/{ISBN}/check	ISBN	Return whether ISBN exists or not
POST	/book/{ISBN}/rate	ISBN rating	Updates rating of a book to a given value

Ausgaben

Titel: Angular **ISBN:** 9783864903571 **Autoren:** Gregor Woiwode, Ferdinand Malcher, Danny Koppenhagen, Johannes Hoppe **Bewertung:** 5
Titel: AngularJS **ISBN:** 3864901545 **Autoren:** Philipp Tarasiewicz, Robin Böhm **Bewertung:** 4

In einem *Ausgabebereich* sollen die Ergebnisse der Operationen angezeigt werden. Diese können sein:

- Die *gefundenen Bücher* bzw. das *gefundene Buch* mit Titel, ISBN-Nummer, Autoren und Bewertung
- Der *Fehler* (error) mit status und satusText, ok und message
- Das *Ergebnis* (response) mit status und satusText, ok und body

Bei *Creates a new (random) book* soll ein neues, mit *zufälligen Werten gefülltes Buch* erstellt und dieses dem Web-Service übergeben werden (**HINWEIS:** *BookFactory.random()*).

Der Knopf *Creates a new (random) book with existing ISBN* soll nur dann gedrückt werden können, wenn gerade eben ein neues Buch angelegt wurde. Es wird dann ein neues Buch angelegt und diesem die bereits existierende ISBN-Nummer übergeben. Das Buch wird in die Datenbank aufgenommen, und der Web-Service reagiert mit dem entsprechenden Fehler (error), der ausgegeben werden soll.

Bei *Updates an existing book* soll ein neues, mit *zufälligen Werten gefülltes Buch* erstellt, dieses mit der *einggegebenen ISBN-Nummer* ausgestattet dem Web-Service übergeben werden.

Bei **Returns whether ISBN exists or not** ist zu beachten, dass der Web-Service im response in der Eigenschaft `body` das Ergebnis `true` zurückliefert wenn das Buch existiert.

Bei **Updates rating of a book to a given value** muss die Bewertung in JSON-Format `{ rating: 5 }` übergeben werden.

- In der nächsten Übung sollen Sie auf einen zur Verfügung gestellten *Web-Service* zugreifen, welchen Sie zuerst installieren und starten sollen. Wie dieser funktioniert und wie die Installation erfolgen soll, wird Ihnen im zur Verfügung gestellten Exkurs *Server-Backend zur Artikelmanipulation mittels REST* erklärt.
- Nachdem der Web-Service läuft sollen Sie ein *neues Angular Projekt* mit *Angular Material* und *Rooting* erstellen. Folgende *Funktionalitäten* sollen zur Verfügung gestellt werden:

Artikelverwaltung

Artikelliste Neuer Artikel Alle Artikel löschen Alle Artikel neu anlegen						
Bild	Id	Beschreibung	Anzahl	Einkaufspreis	Verkaufspreis	Einführungsdatum
	DE12345678	Umwälzpumpe	200	25,30 €	77,90 €	1. Februar 2019
	IT123456	Kupplung	100	5,30 €	7,90 €	2. Februar 2019

Artikelliste

Es soll vom zu programmierenden `ItemService` die *Liste aller Artikel* geholt und deren Eigenschaften angezeigt werden. Über einen *Klick* auf den *Code des Artikels* kann der Artikel bearbeitet werden (siehe nächster Punkt).

Bemerkungen

- Die Dateien `item.ts`, `image.ts` und `item-factory.ts` werden Ihnen zur Verfügung gestellt.
- Programmieren Sie die Service-Klasse `ItemService`, welche Ihnen den Zugriff auf den Web-Service anhand der benötigten Funktionen realisiert:

```

getItem()      checkIdExists()  createItem()
updateItem()   deleteItem()    deleteAllItems()  createAllItems()

```

- Ein Artikel enthält mindestens ein Bild. Das *erste* Bild soll in der Tabelle in Miniaturform angezeigt werden.
- Damit für die *länderspezifische Ausgabe* (de) von Zahlen, Währungen und Datumsangaben die *Angular-Pipes* richtig verwendet werden können, müssen in `app.modules.ts` die folgenden Zeilen unmittelbar untereinander eingefügt werden:

```

import { registerLocaleData } from '@angular/common';
import localeDe from '@angular/common/locales/de';
import { LOCALE_ID } from '@angular/core';
registerLocaleData(localeDe);

```

Weiters muss in derselben Datei unter `providers` folgendes eingefügt werden:

```

providers: [ ..., { provide: LOCALE_ID, useValue: 'de' } ]

```

Folgende *Angular-Pipes* formatieren ihre Ausgaben in der Tabelle entsprechend:

```

{{ item.number | number: '1.0-0' }}
{{ item.purchasingPrice | currency:'EUR': 'symbol':'1.2-2' }}
{{ item.launchDate | date: 'longDate' }}

```

Voraussetzung damit obige Einstellungen greifen ist, dass die *bevorzugte Sprache im Browser* auf *Deutsch* eingestellt ist.

Artikel bearbeiten bzw. Neuer Artikel

Id IT123456	
Beschreibung Kupplung	
Anzahl 100	
Einkaufspreis 5,3	Verkaufspreis 7,9
Einführungsdatum 02.02.2019	
URL https://upload.wikimedia.org/wikipedia/commons/c/	Titel Kraftfahrzeugkupplung
URL https://upload.wikimedia.org/wikipedia/commons	Titel Kupplungsscheibe
<div>Ändern</div> <div>Löschen</div>	

Folgende *Validierungen* müssen realisiert werden:

- **Id:** Primärschlüssel, muss eingegeben werden, muss entweder mit IT oder DE beginnen, bei IT müssen sechs Zeichen folgen bei DE acht. Bei einem bestehenden Artikel darf die Id nicht mehr geändert werden.

Bei neuen Artikeln muss über einen asynchronen Validierer kontrolliert werden, ob die eingegebene Id bereits verwendet wird (**HINWEIS:** Der Web-Service liefert bei einer nicht existierenden Id den Http-Fehler 404, welcher – ähnlich wie bereits beim Postleitzahl-Validierer – in den Rückgabewert null umgewandelt werden muss).

- **Beschreibung:** Muss eingegeben werden.
- **Anzahl:** Darf keine Dezimalzahl enthalten, muss größer oder gleich 0 sein, Standard 0 (**HINWEIS:** type="number").
- **Einkaufspreis:** Muss größer oder gleich 0 sein, Standard 0 (**HINWEIS:** type="number").
- **Verkaufspreis:** Muss größer oder gleich 0 sein, Standard 0 (**HINWEIS:** type="number").
- **Preise:** Wenn beide Werte gesetzt sind, muss der Einkaufspreis kleiner oder gleich dem Verkaufspreis sein.
- **Einführungsdatum:** Muss eingegeben werden, Standard heute (**HINWEIS:** type="date").
- **Bilderliste:** Ein Artikel muss mindestens ein Bild haben.
- **Bilderliste:** Alle URLs der Bilder müssen gesetzt werden, die Titel der Bilder können leer sein. Es muss nicht kontrolliert werden, ob eine korrekte URL eingegeben wurde. Weiters müssen die URLs aller Bilder für den Artikel unterschiedlich sein (**HINWEIS:** Diese letzte Fehlermeldung kann auch mehrmals in den Eingabefeldern für die URL angezeigt werden).

Bemerkungen

- Zu einem Eingabefeld dürfen nicht mehrere Fehlermeldungen gleichzeitig angezeigt werden.
- Alle Fehlermeldungen müssen bereits während der Eingabe erscheinen. Registrieren Sie dazu auch einen globalen `ErrorStateMatcher`.
- Die Id kann sowohl in Groß- als auch in Kleinbuchstaben eingegeben werden. Der Web-Service soll aber dafür sorgen, dass ein Artikel immer mit großgeschriebener Id abgespeichert wird und dass beim Suchen nach einem Artikel immer mit großgeschriebener Id gesucht wird.
- Nur beim *Ändern* eines Artikels darf der Knopf [Löschen] sichtbar sein.
- Die Knöpfe [+] und [-] erlauben es, Bilder zum Artikel *hinzuzufügen* bzw. Bilder vom Artikel zu *löschen*. Verfügt der Artikel nur über ein Bild, so wird für dieses der [-]-Knopf deaktiviert.
- Sollte das *Ändern*, *Hinzufügen* oder *Löschen* eines Artikels vom Web-Service nicht erfolgreich durchgeführt werden können, bzw. sollte bei diesen Operationen der Web-Service nicht erreichbar sein, muss dies dem Benutzer mitgeteilt werden.

Alle Artikel löschen

Löscht alle Artikel, und gibt eine *Fehlermeldung* aus, wenn beim Löschen Probleme aufgetreten sind. Ansonsten wird zur **leeren Artikelliste** gesprungen.

Bemerkung

Da der Web-Service keine Möglichkeit bietet direkt alle Artikel zu löschen, muss zuerst nach allen Artikeln gesucht werden um diese dann einzeln zu löschen. Dies soll durch folgende Methode der Klasse `ItemService` geschehen:

```
deleteAllItems(): Observable<any> {
    return this.getAllItems()
        .pipe(
            switchMap(
                items => {
                    if (items != null && items.length > 0) {
                        return forkJoin(items.map(item => this.deleteItem(item.id)));
                    } else {
                        return new Observable(obs => {
                            obs.next(null);
                            obs.complete();
                        });
                    }
                }
            )
        );
}
```

Mit `switchMap()` wird ein „inneres“ Observable gestartet und dieses anstelle des „äußeren“ Observable zurück geliefert.

Da in Wirklichkeit mehrere „innere“ Observables gestartet werden, müssen diese mit `forkJoin()` zu einem Observable zusammengefasst werden.

Alle Artikel neu anlagen

Löscht zuerst alle Artikel und legt die zwei Standardartikel neu an. Es wird eine *Fehlermeldung* ausgegeben, wenn dabei Probleme aufgetreten sind. Ansonsten wird zur **Artikelliste** gesprungen.

Bemerkungen

- In der Klasse `ItemFactory` liefert die Methode `items()` die zwei Standardartikel zurück.
- Erstellen Sie in der Klasse `ItemService` in Anlehnung an die obige Methode die Methode `createAllItems()` welche zuerst alle Artikel löscht und dann die Standardartikel anlegt.