Angular – Buchhandel – Teil 5

HTTP – Server Backend

Für den Austausch der Daten über eine serverbasierte Schnittstelle stellt Angular die Klasse „http“ zur Verfügung. Dafür gibt es ein eigenes Modul, das HttpClientModule. Dieses muss eingebunden werden in „app.module.ts“.

Info siehe auch: https://www.techiediaries.com/angular-httpclient/

Das Nachladen der Daten kann mithilfe verschiedener Frameworks realisiert werden. Angular verwendet dafür das bekannte Framework „RxJS“.

Hier wird somit das Service, der uns die Daten zu den Büchern liefert, umgebaut, damit man auf ein im Internet abgelegtes API zugreifen kann und dort die Daten von der API geladen werden. Die Daten werden von einer REST-Schnittstelle bezogen. Als serverseitigen Endpunkt wird diese API verwendet:

Schnittstelle zu den Daten der Bücherliste:
https://book-monkey2-api.angular-buch.com/

Diese API greift auf eine Datenbank mittels CRUD-Operationen zu und sorgt für eine persistente Speicherung der Daten.

Diese API liefert alle nötigen Funktionen, die man für die Darstellung und Verwaltung benötigt:

- Abruf aller Informationen aller Bücher
- Abruf von Infos zu einem speziellen Buch
- Hinzufüge eines neuen Buches
- Bearbeiten eines bereits existierenden Buches
- Löschen eines Buches aus dem Bestand

Allgemein: HTTP, Observables und RxJS

1. http get request von einem Service (book-store.service.ts)
2. empfangen des observables (Antwort vom Server)
3. die Dateien, die Daten anzeigen sollen, benötigen ein Abo (subscribe), dass sie das dürfen
4. Anzeigen in einem lokalen View

1)HttpClientModul

1.1.) einbinden in „app.module.ts“

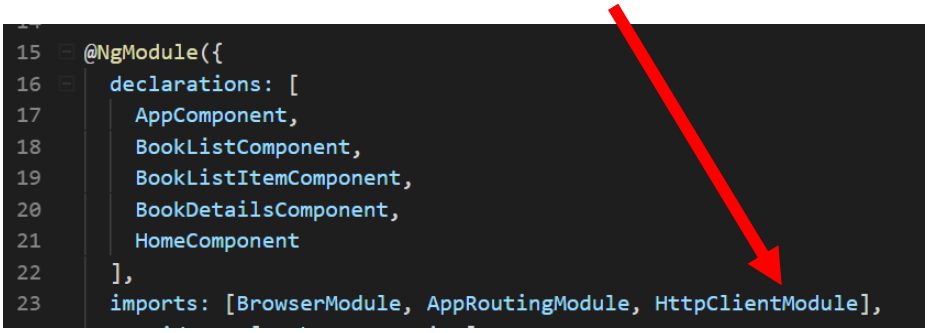
Um ein http-Modul nutzen zu können, benötigt man zuerst das HttpClientModule.
(In früheren Versionen (bis Angular 4) hieß es noch „HttpModule“.)

Öffne die Datei „app.module.ts“ und füge zu den Importen hinzu:

```
import { HttpClientModule } from '@angular/common/http';
```

```
4  import { HttpClientModule } from '@angular/common/http';
```

Dazu muss man es auch noch im „@NgModule“-Array bei den „imports“ einbinden:



Speichern.

Theorie:

1.2.) Nach der Integration kann man in den Komponenten damit arbeiten.

- Dazu wird die Klasse „http“ importiert und
- in der Komponente injiziert.

Der Service stellt dann diverse Schnittstellen zur Kommunikation mit einem HTTP-Backend zur Verfügung.

Beispiel:

```
import { HttpClient } from '@angular/common/http';
.....

export class ApiService {
  apiURL: string = 'http://www.server.com/api/';

  constructor(private httpClient: HttpClient) {}
}
```

Die Anbindung basiert dabei auf der Klasse „observable“ aus der Bibliothek „RxJS“.

RxJS bietet eine Menge von Operatoren um den Datenstrom zu transformieren und zu steuern.
Um die Daten zu empfangen, muss man das Observable abonnieren mit der Methode „subscribe()“.
Die http-Klasse von Angular bietet eine Reihe von Methoden an:

- get(url, options)
- post(url, body, options)

- put(url, body, options)
- delete(url, options)
- patch(url, body, options)
- head(url, options)

Beispiel bei „books“:

DELETE	/books	Resets store to initial state
GET	/books	Get all books
GET	/books/search/{searchTerm}	
	Get all books matching the given search term (case insensitive). The properties isbn, title, authors, published (interpreted as ISO string), subtitle and description are evaluated for a match.	
POST	/book	Creates a new book
DELETE	/book/{isbn}	Deletes a book
GET	/book/{isbn}	Returns a single book by ISBN
PUT	/book/{isbn}	Updates an existing book
GET	/book/{isbn}/check	Returns whether ISBN exists or not
POST	/book/{isbn}/rate	Updates rating of a book to a given value

2)neues File erstellen: Book Factory

Da man nun die Daten von einer Schnittstelle beziehen wird, die lediglich Rohdaten zu einem Buch liefert, muss man die Daten so umwandeln, dass man wieder den Typ „Book“ erhält.

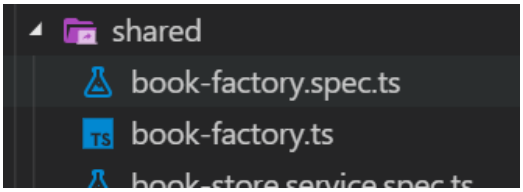
Dort soll eine Klasse eines „books“ angelegt werden. Dort wird später die Antwort des Servers (ist ein observable) „gegossen“ werden.

Dazu erstelle eine separate neue Klasse mit Hilfe der Angular CLI:

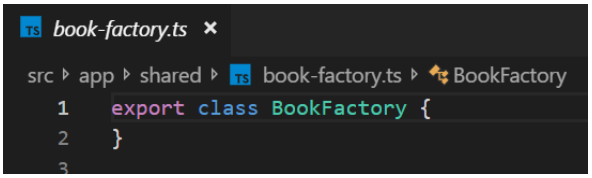
ng g class shared/book-factory

```
C:\angular_buchhandel\buch> ng g class shared/book-factory
```

Ergebnis:

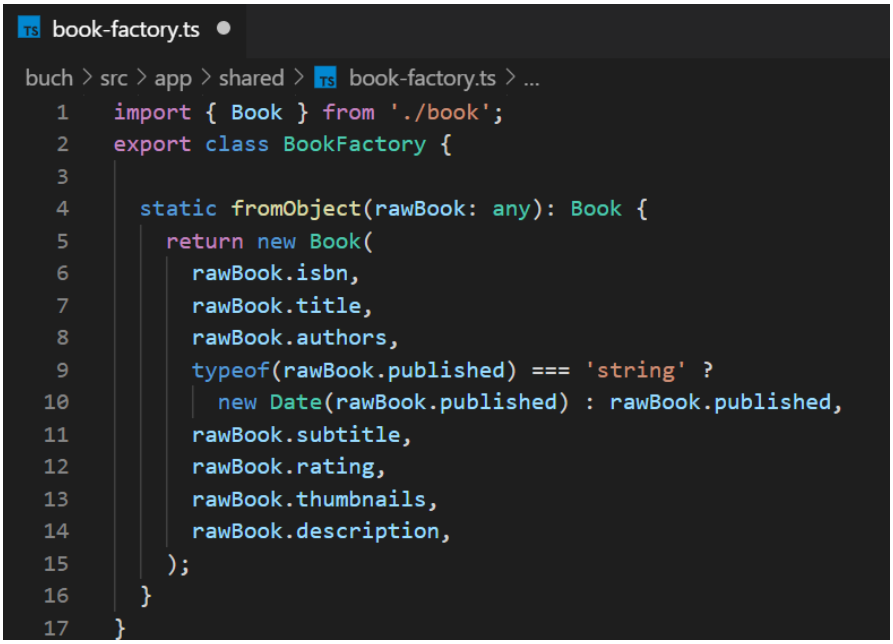


Öffne die „book-factory.ts“



Die erzeugte Klasse soll eine Methode besitzen.

- Methode fromObjekt()
  diese nimmt die Rohdaten entgegen; für das Datum prüft man, ob der Wert als String vorliegt. Wenn ja, dann erzeugt man daraus ein „Date-Objekt“.




```
static fromObject(rawBook: any): Book {
   return new Book(
     rawBook.isbn,
     rawBook.title,
     rawBook.authors,
     typeof(rawBook.published) === 'string' ?
       new Date(rawBook.published) : rawBook.published,
     rawBook.subtitle,
     rawBook.rating,
     rawBook.thumbnails,
     rawBook.description,
   );
 }
```

3)BookStoreService anpassen

Öffne „book-store.service.ts“:

Da man jetzt die „Books“ über das Backend beziehen wird, kann man einiges vorhandenes entfernen.

- in @Injectable entferne die Eigenschaft „books“
- entferne alles aus dem constructor – alle hinterlegten Bücher
- das Thumbnail im Import Zeile 3 kann auch entfernt werden

Ergebnis:

```
3   import { Book } from './book';
4
5   @Injectable()
6   export class BookStoreService {
7
8     constructor() {
9     }
```

- Lege gleich die benötigen Importe an: Zeile 2 und 3
    - HttpClient
    - Observable - Die Observable-Klasse für die Typisierung und Verarbeitung

book-store.service.ts

src ▸ app ▸ shared ▸ book-store.service.ts ▸ BookStoreService

```
1   import { Injectable } from '@angular/core';
2   import { HttpClient } from '@angular/common/http';
3   import { Observable } from 'rxjs/Observable';
4
5   import { Book } from './book';
```

- In der „export class“ des Injectable lege die Eigenschaft „api“ mit einem String an, der auf den Endpunkt unserer API verweist.

  private api = 'https://book-monkey2-api.angular-buch.com/books';

  Die Adresse des Servers wird in die Konstante „api“ ausgelagert.

```
7   @Injectable()
8   export class BookStoreService {
9     private api = 'https://book-monkey2-api.angular-buch.com/books';
```

- Im Konstruktor injiziiere die private Klasse „http“. Danach hat man eine lokale Variable http.

  constructor(private http: HttpClient) { }

```
10
11    constructor(private http: HttpClient) { }
```

3.1.) get-Methode erstellen - die vorhandene Methode umändern

Mit der geänderten „getAll“ soll eine Liste aller Bücher geliefert werden.

Dafür muss zuerst das alte „books“ in ein „http“ geändert werden:

Füge ganz oben nach den Klammern ein:

```
12
13   getAll(): Observable<Book[]> {
14     return this.http
```

Theorie:

- Die Methode „http.get()“ fragt die Daten per GET vom Server ab.
- In der get-Methode wird die URL eingefügt.
- Die „get-Methode“ gibt eine Observable zurück.

```
12
13   getAll(): Observable<Book[]> {
14     return this.http.get<Book[]>(this.api);
15   }
```

3.2.)Test ob bis jetzt alles funktioniert:

Da bis jetzt gewisse Methoden im Service und deren Schnittstelle geändert wurden, liefern diese nun eine Observable zurück.

Nun muss man die Komponente aktualisieren, damit sie das Observable abonnieren, um die Daten zu erhalten. Dazu brauch man ein „subscribe()“.

Öffne „book-list.component.ts“.

Info:
Diese Listansicht muss jetzt die Antwort (Request) der von dem Service gestellten Anfrage mittels http an den Server irgendwie bekommen. Dazu dient das „subscribe“, also abonniert die „List“ das Service.

Übung:

Füge in Zeile 12 ein und füge in Zeile 14 zum „constructor“ hinzu:

```
10  export class BookListComponent implements OnInit {
11
12    books: Book[];
13
14    constructor(private bs: BookStoreService) {}
```

Info: das „Book“ in Zeile 12 ist die Klasse, die oben importiert wurde.

Ändere die „ngOnInit“, um die Servicemethode in der Listenansicht nun zu verwenden.

Hier wird die Servicemethode „getAll()“ aufgerufen, das Observable abonniert und dann die empfangenen Daten in die Eigenschaft „this.books“ gespeichert:

Alt:

```
16  ngOnInit() {
17    this.books = this.bs.getAll();
18  }
```

Danach:

```
16  ngOnInit() {
17    this.bs.getAll().subscribe(res => this.books = res);
18  }
```

Das Linke „res“ ist das Argument zur Funktion, das rechte „this.books = res“ der Body der Funktion.

3.2.1.)einiges löschen

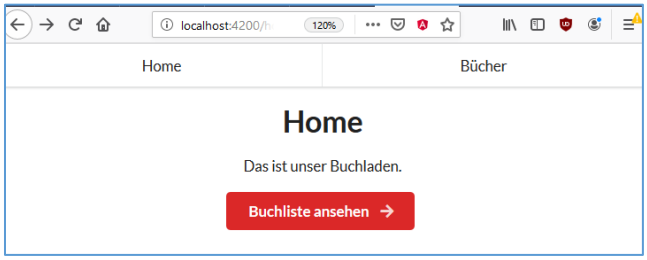
Da es zu einigen Fehlermeldungen kommt, trennen wir uns von der Möglichkeit die Detailansicht zu zeigen.

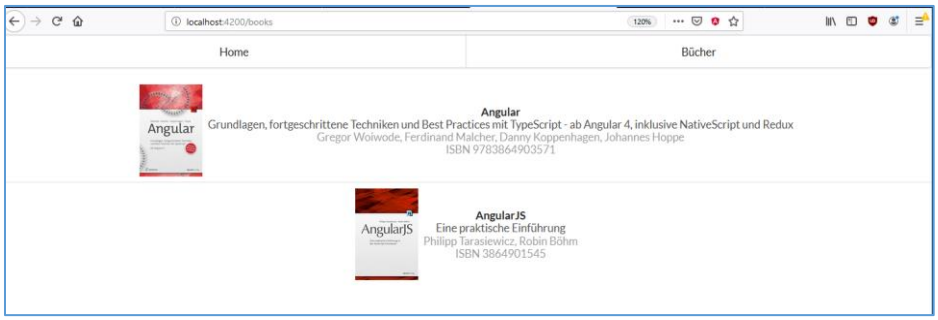
Lösche

- book-store.service.ts
  Lösche die Methode „getSingle()“
- book-details.component.ts
  lösche die Zeile bezüglich „getSingle“ – Zeile 20.

```
18  ngOnInit() {
19    const params = this.route.snapshot.params;
20    this.book = this.bs.getSingle(params['isbn']);
21  }
```

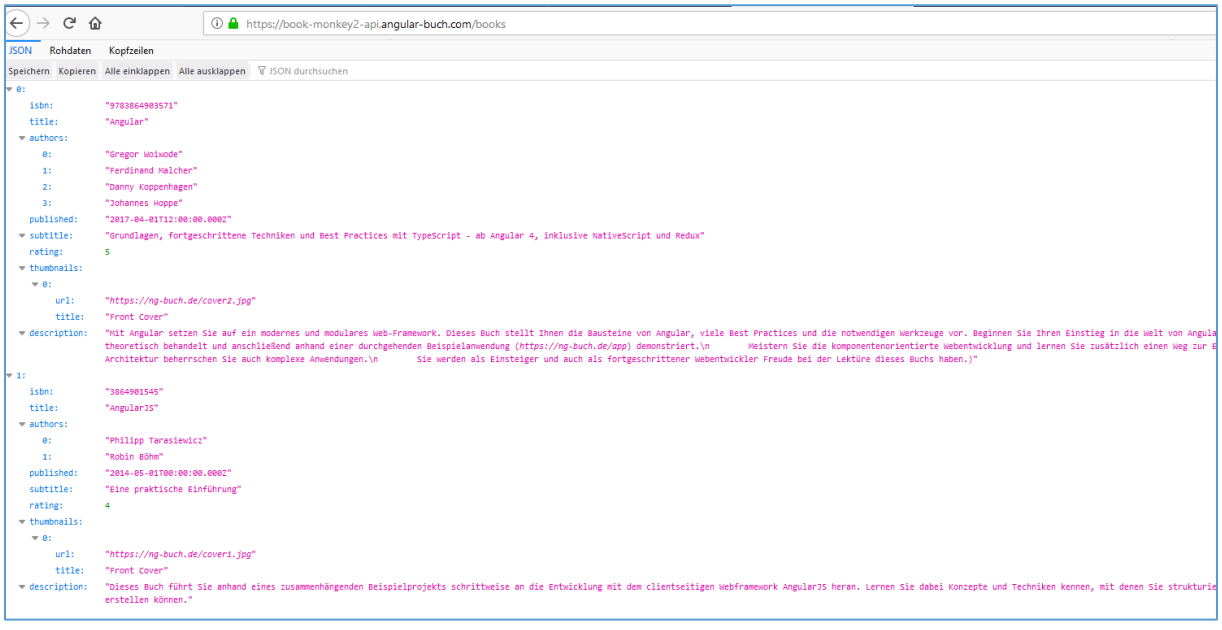
Ergebnis:





Inhalt der Seite

https://book-monkey2-api.angular-buch.com/books



Quellen:

Woiwode, Malcher u.a. in: Angular; dpunkt-Verlag, 2018, S. 169-185

https://www.youtube.com/watch?v=LmIsbzt-S_E&list=PLC3y8-rFHvwhBRAgFinJR8KHIrCdTkZcZ&index=21

https://www.youtube.com/watch?v=Fdf5aTYRW0E ab ca. 49 min