

# Angular Workshop

## Before You Start

When you see a **section that is highlighted in red** it means that this is code that needs to be removed.

When you see a **section that is highlighted in green** it means that this is code that needs to be added/modified.

The sections that are not highlighted refer to code that is unchanged.

## Lab 1 – Setup

Install angular cli by issuing the following command

**npm install -g @angular/cli**

**Note:** If you are facing “eperm operation not permitted error” you may want to try to use the -f flag (flag that forces to continue in the event of an error during the installation process). This usually happens when you have internal policies that make it challenging for npm to operate normally.

Create a new Angular project using the following command:

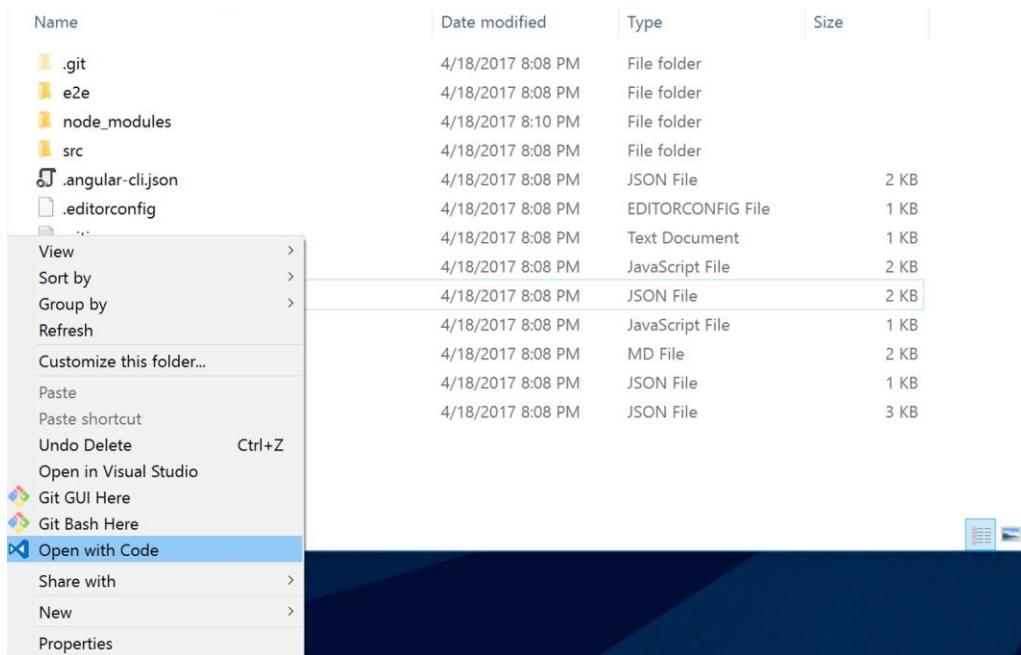
**ng new Angular-Workshop --routing**

Notice that the --routing flag was used to pre-generate the app-routing.module.ts file which will be used later to setup the routing collection.

Navigate to the newly created project using the following command:

**cd Angular-Workshop**

We will assume that VS Code is the editor of choice for this workshop but you can use Visual Studio 2015/2017 as well. Open the project with VS Code. From inside the folder right click on open with VS Code. If you don't see the option this means that you forgot to enable VS Code context menu option during installation.



Alternatively, you can type “code .” on the command line inside the project directory.

```
C:\Windows\system32\cmd.exe
create src\index.html
create src\main.ts
create src\polyfills.ts
create src\styles.css
create src\test.ts
create src\tscconfig.app.json
create src\tscconfig.spec.json
create src\typings.d.ts
create .angular-cli.json
create e2e\app.e2e-spec.ts
create e2e\app.po.ts
create e2e\tscconfig.e2e.json
create .gitignore
create karma.conf.js
create package.json
create protractor.conf.js
create tsconfig.json
create tslint.json
Successfully initialized git.
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Project 'Angular-Workshop' successfully created.

C:\Users\waelkdouh\Desktop>cd Angular2Workshop
C:\Users\waelkdouh\Desktop\Angular2Workshop>cd ..
C:\Users\waelkdouh\Desktop>cd Angular-Workshop
C:\Users\waelkdouh\Desktop\Angular-Workshop>code .
```

Run the application using the following command inside the vs code command line

**ng serve -o**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\waelkdouh\Desktop\Angular-Workshop>npm start

> angular-workshop@0.0.0 start C:\Users\waelkdouh\Desktop\Angular-Workshop
> ng serve

** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 **
Hash: 539363b49c1a20cd9cbd
Time: 12019ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 158 kB {4} [initial] [rendered]
chunk    {1} main.bundle.js, main.bundle.js.map (main) 5.36 kB {3} [initial] [rendered]
chunk    {2} styles.bundle.js, styles.bundle.js.map (styles) 10.5 kB {4} [initial] [rendered]
chunk    {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.11 MB [initial] [rendered]
chunk    {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.
```

The -o flag will load the application in the browser

**Tip:** You can also click ctrl + hover over the link shown in the command line window to navigate to the url (<http://localhost:4200>).

---

**Welcome to app!!**



Here are some links to help you start:

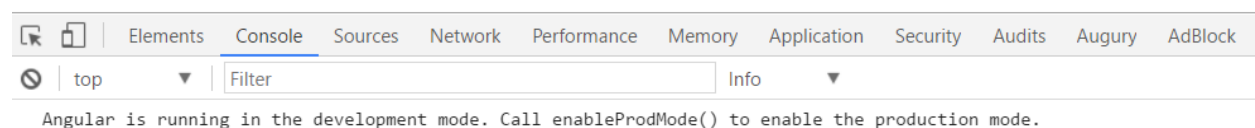
- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

You have now successfully created/run your first Angular application. You can keep the application running as the server will support live page reloading when making any future changes to the code. Make sure you open a new command windows under VS Code as you will use that to execute different cli commands without stopping the server.

Lets try to modify the title property under app.component.ts file to demonstrate how changing a file and saving it will automatically refresh the browser to reflect the changes. Expand the app folder and change the title property to “My Tiny Library App!!!”. **Save.**



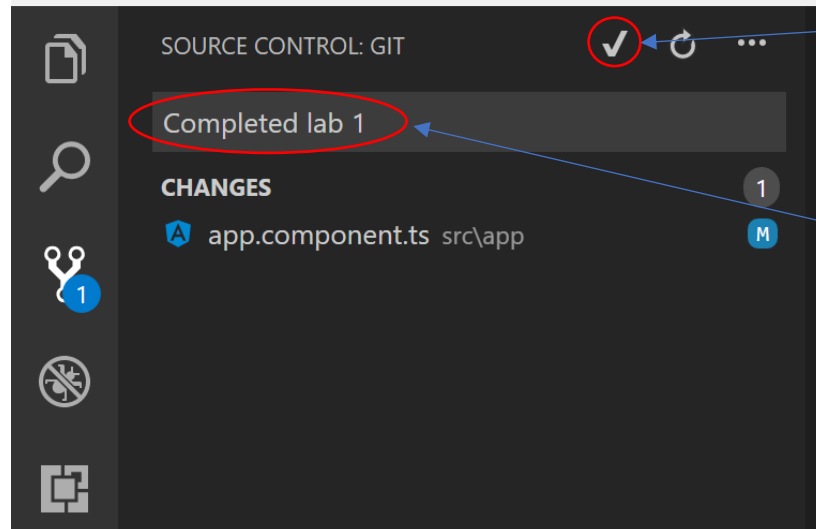
**Tip:** Make sure to always have the F12 tools open under your browser (navigate to the console tab) as Angular provides you with valuable debugging information there. Notice by default the Angular runtime is telling you that you are running in development mode. More on that later.



One of the things that you will notice is that the angular cli automatically created a git file. This makes it possible to check in our code (note that you are checking in locally as git is distributed by nature. If you want to push to a server you will need to use the push command, but its outside the scope of this workshop) after completing each lab which will make it easier to undo any challenging issues that may arise by rolling back to previous check ins. Go ahead and commit the completed lab 1 by entering a message in the message field and clicking on the check mark as shown in the figure below. Ask the instructor for help if you are not familiar with Git.

app.component.ts - Angular-Workshop - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

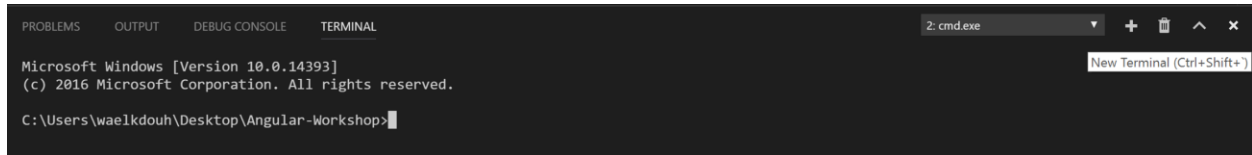


Step 2:  
Commit  
changes

Step 1:  
Enter  
commit  
message

## Lab 2 – Template Containing a Component

Open a new command window in VS Code by navigating to the terminal tab and clicking on the + icon . This way the first command window will keep the server running for live reload to work as mentioned before.

A screenshot of a VS Code terminal window. The terminal title bar shows '2: cmd.exe'. The terminal content displays the Windows version and copyright information, followed by the current directory path: C:\Users\waelkdouh\Desktop\Angular-Workshop>. A 'New Terminal (Ctrl+Shift+T)' button is visible in the top right corner of the terminal panel.

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\waelkdouh\Desktop\Angular-Workshop>
```

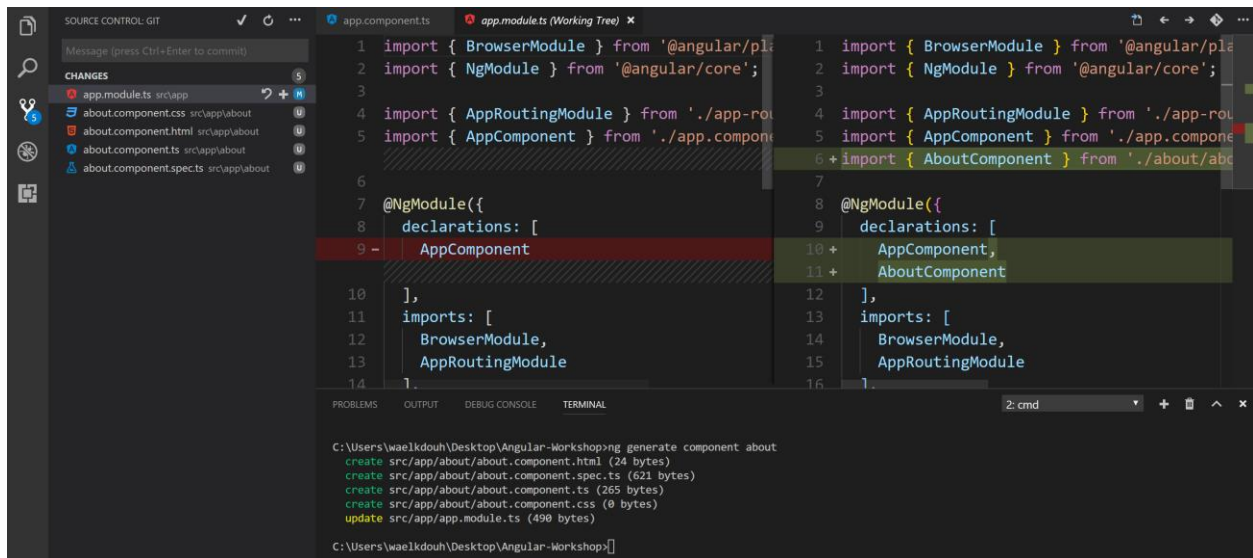
Add a new component to your page by issuing the following command  
**ng generate component about**

This will generate a new component in a new folder called about

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate component about
installing component
  create src\app\about\about.component.css
  create src\app\about\about.component.html
  create src\app\about\about.component.spec.ts
  create src\app\about\about.component.ts
  update src\app\app.module.ts
```

Notice that the app.module.ts file has been updated to include the newly added component. Specifically, the AboutComponent has been added to the declarations array and the import statement for the same component has been added at the top of the file. You can easily spot that by heading to the source control tab under VS Code which allows you to see the changes since the last commit as shown here:



Modify about.component.ts file to include a pageTitle property that we will bind to in the associated template. Also change the default selector name from app-about to my-about (we could have kept the default that got generated by the cli as well but its better to use a more descriptive selector).

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'my-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {
  constructor() { }

  ngOnInit() {}

  pageTitle:string = 'About Me';
}
```

Next lets modify about.component.html file to display the property we just added under the about.component.ts using one way interpolation. Replace the content of about.component.html file with the following:

```
<h3>{{pageTitle}}</h3>
<p>Welcome to your personal library </p>
```

Since we are bootstrapping our app using the AppComponent we will need to include the newly added AboutComponent under the AppComponent's template. Modify the AppComponent template to include the newly introduced About Component by replacing the app.component.html file with the code below. **Save.**

```
<h1>My Tiny Library App</h1>
<my-about></my-about>
```

The application should now display the about page with the text “My tiny Library App” above it.

---

# My Tiny Library App

## About Me

Welcome to your personal library

Remember to commit lab 2.



## Lab 3 – Data Binding

For this lab we will utilize a framework called [Angular Material](#) which offers UI controls for Angular applications (e.g. tabs controls, Modal box, etc.). We will also utilize Angular Material for theming our application.

Start by installing Angular Material and Angular CDK inside your project using the following command:

```
npm install --save @angular/cdk
```

```
npm install --save @angular/material
```

Some Material components depend on the Angular animations module to be able to do more advanced transitions. If you want these animations to work in your app, you have to install the @angular/animations module and include the BrowserAnimationsModule in your app.

```
npm install --save @angular/animations
```

Add the newly installed modules in addition to the FormsModule (this will be used for binding) under the app.module.ts file:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { MatListModule, MatCardModule, MatSlideToggleModule, MatDialogModule,
        MatIconModule, MatInputModule, MatSnackBarModule, MatTabsModule,
        MatButtonModule, MatLineModule, MatToolbarModule } from '@angular/material';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
```

```

    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
    MatToolbarModule,
    BrowserAnimationsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Next you will need to add a new component called collection which will be responsible for displaying your book library. Execute the following command to add the new component:

### **ng generate component collection**

This will generate a new component in a new folder called collection.

```

C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate component collection
create src/app/collection/collection.component.html (29 bytes)
create src/app/collection/collection.component.spec.ts (656 bytes)
create src/app/collection/collection.component.ts (285 bytes)
create src/app/collection/collection.component.css (0 bytes)
update src/app/app.module.ts (1234 bytes)

```

Update the collection.component.ts file to include the following properties:

- pageTitle: string; property which will be shown in the panel header
- books: Array<Ibook>; property which will be shown in the table rows
- showOperatingHours: boolean = false; property for message visibility and toggle button text
- openingTime:Date; property that shows the opening time
- closingTime:Date; property that shows the closing time

Update the constructor to initialize the opening and closing time. Initialize the book array to point to an array of four books. Note that the book data will be eventually moved to a backend service in later labs.

Also change the selector name from app-collection to my-collection. You can ignore the error message notifying you that IBook is an unknown type as you will add it next.

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {
  pageTitle: string;
  books: Array<Ibook>=
  [
    {
      id: 1,
      title: "JavaScript - The Good Parts",
      author: "Douglas Crockford",
      isCheckedOut: true,
      rating: 3
    },
    {
      id: 2,
      title: "The Wind in the Willows",
      author: "Kenneth Grahame",
      isCheckedOut: false,
      rating: 4
    },
    {
      id: 3,
      title: "Pillars of the Earth",
      author: "Ken Follett",
      isCheckedOut: true,
      rating: 5
    },
    {
      id: 4,
      title: "Harry Potter and the Prisoner of Azkaban",
      author: "J. K. Rowling",
      isCheckedOut: false,
      rating: 5
    }
  ];
  showOperatingHours: boolean;
  openingTime: Date;
```

```

closingTime:Date;

constructor() {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
}

ngOnInit() {
}
}

```

Add IBook interface using the following command

```
ng generate interface IBook
```

```
C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate interface IBook
installing interface
  create src\app\ibook.ts
```

Modify the IBook interface to include the following properties:

```
export interface Ibook {
  id: number,
  title: string,
  author: string,
  isCheckedOut: boolean,
  rating: number
}
```

Modify the `collection.component.html` file to include the following code:

```
<h3>{{pageTitle}}&nbsp;    
  <mat-slide-toggle class="plr-15" color="primary" [(ngModel)]="showOperatingHours">  
    {{showOperatingHours ? 'Hide' : 'Show'}} library hours  
  </mat-slide-toggle>  
</h3>  
<div [hidden]="!showOperatingHours">  
  <mat-card>  
    <mat-card-subtitle><strong>Operating Hours</strong></mat-card-subtitle>  
    <mat-card-content>{{openingTime}} - {{closingTime}} (M-F)</mat-card-content>  
  </mat-card>
```

```

</div>
<div>
  <mat-list>
    <mat-list-item *ngFor="let book of books">
      <mat-icon mat-list-icon>book</mat-icon>
      <h3 mat-line><strong>{{book.title}}</strong></h3>
      <p mat-line>
        <span>{{book.author}}</span>
      </p>
      <p mat-line>
        {{book.rating}}
      </p>
      <p mat-line>
        <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-success'" >
          {{book.isCheckedOut ? 'Checked-Out' : 'Available'}}
        </span>
      </p>
    </mat-list-item>
  </mat-list>
</div>

```

Modify the collection.component.css to include styling for the collection component:

```

.mat-list .mat-list-item .mat-list-icon, .mat-nav-list .mat-list-item .mat-list-
icon {
  width: 48px;
  height: 48px;
  font-size: 48px;
  color: #b4bcc2;
}

.chip {
  display: inline;
  padding: .2em .6em .3em;
  font-size: 85%;
  font-weight: bold;
  line-height: 1;
  color: #ffffff;
  text-align: center;
  white-space: nowrap;
  vertical-align: baseline;
  border-radius: .25em;
}

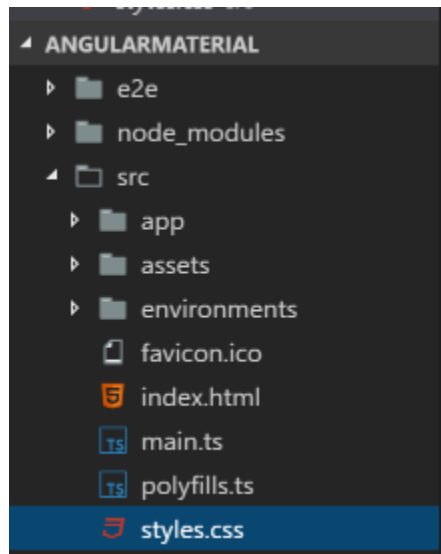
```

```
.chip-success {
  background-color: #18bc9c;
}
.chip-danger {
  background-color: #e74c3c;
}
.add-btn {
  padding: 8px 65px;
}
```

Modify the app.component.html file to include the newly added collection component. Make sure you remove the about component element.

```
<h1>My Tiny Library App</h1>
<my-about></my-about>
<my-collection></my-collection>
```

To enable theming using angular material replace the styles.css file generated by the angular cli with the file provided to you in the lab 3 folder assets. **Save.**



**Note:** At this point the about page is not included in our application anymore. This is temporary as we will restore it later when we introduce material tabs into our application.

The application should now display the collection page. The two images below show the page with the slide toggle enabled/disabled.

## My Tiny Library App

Books ☐ Show library hours



### JavaScript - The Good Parts

Douglas Crockford

3

Checked-Out



### The Wind in the Willows

Kenneth Grahame

4

Available



### Pillars of the Earth

Ken Follett

5

Checked-Out



### Harry Potter and the Prisoner of Azkaban

J. K. Rowling

5

Available

## My Tiny Library App

Books ☒ Hide library hours

Operating Hours

Mon Sep 25 2017 10:00:02 GMT-0700 (Pacific Daylight Time) - Mon Sep 25 2017 15:00:02 GMT-0700 (Pacific Daylight Time) (M-F)



### JavaScript - The Good Parts

Douglas Crockford

3

Checked-Out



### The Wind in the Willows

Kenneth Grahame

4

Available



### Pillars of the Earth

Ken Follett

5

Checked-Out



### Harry Potter and the Prisoner of Azkaban

J. K. Rowling

5

Available

Remember to commit Lab 3.

## Lab 4 – Pipes

Add a new pipe called rating-category.pipe and make sure you provide an explicit path to add under pipes folder by executing the following command:

**ng generate pipe pipes/rating-category --spec=false**

Notice that we set the `--spec` flag to false to inform the cli that we don't want to generate a spec file for the pipe. Of course this is completely optional and you could have included a spec file for your component.

**Note:** Spec means test under jasmine which is the default framework utilized by the angular cli for JavaScript unit testing.

The rating category pipe will replace the rating numbers with either "Poor", "Fine", or "Excellent" depending on the book rating.

Modify the rating-category.pipe.ts file to return Poor (1-2), Fine (3-4), Excellent (5)

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'ratingCategory'
})
export class RatingCategoryPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    transform(value: number): string {
      if (value <= 2) {
        return 'Poor';
      }
      if (value <= 4) {
        return 'Fine';
      }
      return 'Excellent';
    }
  }
}
```



Update the collection.component.html file to use the date pipe (built-in pipe into Angular) to the openingTime and closingTime. Also apply the newly introduced Rating Category pipe (custom Pipe). **Save.**

```
<h3>{{pageTitle}}&nbsp;  </h3>
  <mat-slide-toggle class="plr-15" color="primary" [(ngModel)]="showOperatingHours">
    {{showOperatingHours ? 'Hide' : 'Show'}} library hours
  </mat-slide-toggle>
</h3>

<div [hidden]="!showOperatingHours">
  <mat-card>
    <mat-card-subtitle><strong>Operating Hours</strong></mat-card-subtitle>
    <mat-card-content>{{openingTime | date:'shortTime'}} - {{closingTime
      | date:'shortTime'}} (M-F)
    </mat-card-content>
  </mat-card>
</div>

<div>
  <mat-list>
    <mat-list-item *ngFor="let book of books">
      <mat-icon mat-list-icon>book</mat-icon>
      <h3 mat-line><strong>{{book.title}}</strong></h3>
      <p mat-line>
        <span>{{book.author}}</span>
      </p>
      <p mat-line>
        {{book.rating | ratingCategory}}
      </p>
      <p mat-line>
        <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-success'">
          {{book.isCheckedOut ? 'Checked-Out' : 'Available'}}
        </span>
      </p>
    </mat-list-item>
  </mat-list>
</div>
```

The collection page should now show the rating numbers with text. In addition, the operating hours should now be replaced with a short time.

## My Tiny Library App

Books ☐ Show library hours



### JavaScript - The Good Parts

Douglas Crockford

Fine

Checked-Out



### The Wind in the Willows

Kenneth Grahame

Fine

Available



### Pillars of the Earth

Ken Follett

Excellent

Checked-Out



### Harry Potter and the Prisoner of Azkaban

J. K. Rowling

Excellent

Available

## My Tiny Library App

Books ☒ Hide library hours

### Operating Hours

10:00 AM - 3:00 PM (M-F)



### JavaScript - The Good Parts

Douglas Crockford

Fine

Checked-Out



### The Wind in the Willows

Kenneth Grahame

Fine

Available



### Pillars of the Earth

Ken Follett

Excellent

Checked-Out



### Harry Potter and the Prisoner of Azkaban

J. K. Rowling

Excellent

Available

Remember to commit Lab 4.

## Lab 5 – Communication between Parent and Child Components

In this lab we will add a new component called rating component which will allow us to demonstrate the parent/child components relationship under Angular. Specifically, the collection component will act as the parent component and will pass the rating number to the child component which is the rating component. Once the rating is received, the rating component will replace the rating with star/s. In addition, the rating component will also communicate back to the collection component a message confirming the updated rating.

Add a Rating Component by executing the following command:

**ng generate component rating**

```
C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate component rating
create src/app/rating/rating.component.html (25 bytes)
create src/app/rating/rating.component.spec.ts (628 bytes)
create src/app/rating/rating.component.ts (269 bytes)
create src/app/rating/rating.component.css (0 bytes)
update src/app/app.module.ts (1409 bytes)
```

Modify the rating.component.ts file to include:

- @Input() rating: number; property which will receive the rating from the parent component
- @Input() book: Ibook; property which will receive the book for which the rating will be modified
- @Output() ratingClicked: EventEmitter<Ibook> = new EventEmitter<Ibook>(); property which will be used to emit a message back to the parent component
- Add click(rating:number): void method which emits the updated book rating via @Output() EventEmitter when the user changes the rating
- Change the app-rating selector name to my-rating

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { Ibook } from '../ibook';

@Component({
  selector: 'my-rating',
  templateUrl: './rating.component.html',
  styleUrls: ['./rating.component.css']
})
export class RatingComponent implements OnInit {

  @Input() rating: number;
```

```

@Input() book: Ibook;
@Output() ratingClicked: EventEmitter<Ibook> = new EventEmitter<Ibook>();
constructor() { }

ngOnInit() {
}

click(rating:number): void {
  this.book.rating = rating;
  this.ratingClicked.emit(this.book);
}
}

```

Modify the rating.component.css file to include the following styling:

```

.material-icons {
  cursor: pointer;
  color: rgba(103,58,183,.15);
}
.material-icons:hover {
  color: rgba(103,58,183,.35);
}
.material-icons.active {
  color: #000;
}

```

Modify the rating.component.html file to show the correct number of stars for the rating:

```
<div>
  <a (click)="click(1)">
    <i class="material-icons {{rating >= 1 ? 'active' : ''}}">
      star_rate
    </i>
  </a>
  <a (click)="click(2)">
    <i class="material-icons {{rating >= 2 ? 'active' : ''}}">
      star_rate
    </i>
  </a>
  <a (click)="click(3)">
    <i class="material-icons {{rating >= 3 ? 'active' : ''}}">
      star_rate
    </i>
  </a>
  <a (click)="click(4)">
    <i class="material-icons {{rating >= 4 ? 'active' : ''}}">
      star_rate
    </i>
  </a>
  <a (click)="click(5)">
    <i class="material-icons {{rating >= 5 ? 'active' : ''}}">
      star_rate
    </i>
  </a>
</div>
```

Modify the collection.component.html to utilize the newly added rating component:

```
<h3>{{pageTitle}}&nbsp;<mat-slide-toggle class="plr-15" color="primary"
  [(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library
  hours
</mat-slide-toggle>
</h3>
<div [hidden]="!showOperatingHours">
  <mat-card>
    <mat-card-subtitle><strong>Operating Hours</strong></mat-card-subtitle>
    <mat-card-content>{{openingTime | date:'shortTime'}} - {{closingTime |
      date:'shortTime'}} (M-F)</mat-card-content>
  </mat-card>
```

```

</div>
<div>
  <mat-list>
    <mat-list-item *ngFor="let book of books">
      <mat-icon mat-list-icon>book</mat-icon>
      <h3 mat-line><strong>{{book.title}}</strong></h3>

      <p mat-line>
        <span>{{book.author}}</span>
      </p>
      <p mat-line>
        {{book.rating | ratingCategory}}

        <my-rating [rating]="book.rating" [book]="book"
          (ratingClicked)="onRatingUpdate($event)">
        </my-rating>
      </p>
      <p mat-line>
        <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>
      </p>
    </mat-list-item>
  </mat-list>
</div>

```

Modify collection.component.ts to add a method to show the rating update message confirmation and listen to the click event that is triggered by the rating component (the child component in this example). **Save.**

```

import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar } from '@angular/material';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>=
  [
    {
      id: 1,
      title: "JavaScript - The Good Parts",
      author: "Douglas Crockford",
      isCheckedOut: true,
      rating: 3
    },
    {
      id: 2,

```

```

        title: "The Wind in the Willows",
        author: "Kenneth Grahame",
        isCheckedOut: false,
        rating: 4
    },
    {
        id: 3,
        title: "Pillars of the Earth",
        author: "Ken Follett",
        isCheckedOut: true,
        rating: 5
    },
    {
        id: 4,
        title: "Harry Potter and the Prisoner of Azkaban",
        author: "J. K. Rowling",
        isCheckedOut: false,
        rating: 5
    }
];

showOperatingHours: boolean;
openingTime: Date;
closingTime: Date;

constructor(private _snackBar: MatSnackBar) {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
}

ngOnInit() {
}

updateMessage(message: string, type: string): void {
    if (message) {
        this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
            duration: 3000
        });
    }
}

```

```


onRatingUpdate(book: Ibook): void {
  this.updateMessage(book.title, " Rating has been updated");
}
}

```

The collection page should now display stars reflecting the rating. You should also see a message at the bottom of the page upon changing the rating a book.

## My Tiny Library App


Books ☐ Show library hours

- 

JavaScript - The Good Parts

Douglas Crockford


★ ★ ★ ☆ ☆

Checked-Out
- 

The Wind in the Willows

Kenneth Grahame


★ ★ ★ ★ ☆

Available
- 

Pillars of the Earth

Ken Follett

★ ★ ★ ★ ★

Checked-Out
- 

Harry Potter and the Prisoner of Azkaban

J. K. Rowling

★ ★ ★ ★ ★

Available

Rating has been updated: JavaScript - The Good Parts

DISMISS

Remember to commit lab 5.



## Lab 6 – Build a Service

Up to this point we have been embedding the book array inside the collection component. This is not a recommended coding practice as the data does not belong in the component, but instead it should be fetched from a backend service like a Asp.Net WebApi service hosted on Azure for example.

In this lab we will add a Angular service that will be responsible for serving the data to be displayed on the book collection page. Keep in mind that we will ultimately move the data to a database in later labs.

Start by generating a new Angular service using the following command:

**ng generate service services/data**

```
C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate service services/data
create src/app/services/data.service.spec.ts (362 bytes)
create src/app/services/data.service.ts (110 bytes)
```

Carry the following modifications on the data.service.ts file:

- Add a method called `getBooks` which has the following signature:
  - `getBooks(): Array<Ibook>`
- Move the array of books from the `collection.component.ts` file into `getBooks()` method under the `DataService`. As mentioned above the data doesn't belong under the angular service either, but rather the data must be stored on some persistent storage like a DB. We will modify this in later labs

Here is the updated content of data.service.ts file:

```
import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';

@Injectable()
export class DataService {

  constructor() { }

  getBooks(): Array<Ibook> {
    return [
      {
        id: 1,
        title: "JavaScript - The Good Parts",
        author: "Douglas Crockford",
        isCheckedOut: true,
        rating: 3
      },
      {
        id: 2,
        title: "The Wind in the Willows",
        author: "Kenneth Grahame",
        isCheckedOut: false,
        rating: 4
      },
      {
        id: 3,
        title: "Pillars of the Earth",
        author: "Ken Follett",
        isCheckedOut: true,
        rating: 5
      },
      {
        id: 4,
        title: "Harry Potter and the Prisoner of Azkaban",
        author: "J. K. Rowling",
        isCheckedOut: false,
        rating: 5
      }
    ];
  }
}
```

Next, we will need to modify the `collection.component.ts` file to start utilizing the newly introduced data service. You will need to carry the following modifications on the collection component:

- Remove the book array if you haven't done so yet
- Constructor should now expect an instance of `DataService`. We will utilize the built in dependency injection feature under Angular. Add the following parameter to the constructor: `private _dataService: DataService`
- Initiate a call to `getBooks()` method under the `ngOnInit` method and assign the result to its `books` property:  
`this.books = this._dataService.getBooks();`

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar } from '@angular/material';
import { DataService } from '../services/data.service';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>=
  [
    {
      id: 1,
      title: "JavaScript - The Good Parts",
      author: "Douglas Crockford",
      isCheckedOut: true,
      rating: 3
    },
    {
      id: 2,
      title: "The Wind in the Willows",
      author: "Kenneth Grahame",
      isCheckedOut: false,
      rating: 4
    },
    {
      id: 3,
      title: "Pillars of the Earth",
      author: "Ken Follett",
      isCheckedOut: true,
      rating: 5
    },
    {
      id: 4,
```

```

        title: "Harry Potter and the Prisoner of Azkaban",
        author: "J. K. Rowling",
        isCheckedOut: false,
        rating: 5
    }
];

books: Array<Ibook>;

showOperatingHours: boolean;
openingTime: Date;
closingTime: Date;

constructor(private _snackBar: MatSnackBar, private _dataService: DataService) {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
}

ngOnInit() {
    this.books = this._dataService.getBooks();
}

updateMessage(message: string, type: string): void {
    if (message) {
        this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
            duration: 3000
        });
    }
}

onRatingUpdate(book: Ibook): void {
    this.updateMessage(book.title, " Rating has been updated");
}
}

```

At this point you may be getting an error informing you that there is no provider for DataService. This is expected as we still need to include the newly added DataService to the list of providers under the AppComponent as its only going to be utilized under this component. **Save.**

**Note:** We could have added DataService to the AppModule's list of providers which would have exposed the service to all the components within the angular module.

```
import { Component } from '@angular/core';
import { DataService } from '../services/data.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [DataService]
})
export class AppComponent {
  title = 'My Tiny Library App!!!';
}
```

You should now be able to go back and see the list of books being displayed again under the book collection page.

Remember to commit lab 6.

## Lab 7 – Http Service

In the previous lab we moved the array of books from the collection component to the data service. In this lab we will remove the books array from the data service and instead point our data service to an Asp.Net Core WebApi which is hosted on Azure. The service returns an array of books in json format.

For this lab we will utilize a library called reactive extensions which allows us to fetch the data from a backend service using the observable pattern.

Modify the data.service.ts file to include code that is capable of making server side calls instead of simply returning hard coded array of books. Here are the steps that are required:

- Constructor should now receive an instance of Http service using dependency injection. Add the following parameter to the constructor:  
private \_http: Http.
- Change getBooks() return type to Observable<Ibook[]> and remove the hard coded array. Modify the body of getBooks() calls to initiate an http call to the server using Angular's built in http service.
- Add a method to handle potential errors when communicating to the backend server.

```

import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';

@Injectable()
export class DataService {

  _booksUrl:string = 'http://waelsbookservice.azurewebsites.net/books';

  constructor(private _http: Http) { }

  private handleError(error: any) {
    let errMsg = (error.message) ? error.message : error.status ?
    `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Observable.throw(errMsg);
  }

  getBooks(): Array<Ibook> {
    return [
      {
        id: 1,
        title: "JavaScript - The Good Parts",
        author: "Douglas Crockford",
        isCheckedOut: true,
        rating: 3
      },
      {
        id: 2,
        title: "The Wind in the Willows",
        author: "Kenneth Grahame",
        isCheckedOut: false,
        rating: 4
      },
      {
        id: 3,
        title: "Pillars of the Earth",
        author: "Ken Follett",
        isCheckedOut: true,
        rating: 5
      }
    ]
  }
}

```

```

    },
    {
      id: 4,
      title: "Harry Potter and the Prisoner of Azkaban",
      author: "J. K. Rowling",
      isCheckedOut: false,
      rating: 5
    }
  ];
}

getBooks(): Observable<Ibook[]> {
  return this._http.get(this._booksUrl+"/GetBooks")
    .map((response: Response) => {
      let data: Ibook[] = <Ibook[]>response.json();
      return data;
    })
    .catch(this.handleError);
}
}

```

Update the app.module.ts file to include the HttpClientModule by including it in the imports array:

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { FormsModule } from '@angular/forms';
import { MatListModule, MatCardModule, MatSlideToggleModule, MatDialogModule,
  MatIconModule, MatInputModule, MatSnackBarModule, MatTabsModule,
  MatButtonModule, MatLineModule, MatToolbarModule } from '@angular/material';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { CollectionComponent } from './collection/collection.component';
import { RatingCategoryPipe } from './pipes/rating-category.pipe';
import { RatingComponent } from './rating/rating.component';
import { HttpClientModule } from '@angular/http';

```



```
@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    CollectionComponent,
    RatingCategoryPipe,
    RatingComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
    MatToolbarModule,
    BrowserAnimationsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Now introduce a `getbooks()` method under `CollectionComponent` which returns the list of books and has the following signature: `getBooks(): void`. The new method will subscribe to the newly introduced observable angular http service.

Save.

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar } from '@angular/material';
import { DataService } from '../services/data.service';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>;

  showOperatingHours: boolean;
  openingTime: Date;
  closingTime: Date;

  constructor(private _snackBar: MatSnackBar, private _dataService: DataService) {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
  }

  ngOnInit() {
    this.books = this._dataService.getBooks();
    this.getBooks();
  }

  updateMessage(message: string, type: string): void {
    if (message) {
      this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
        duration: 3000
      });
    }
  }

  onRatingUpdate(book: Ibook): void {
    this.updateMessage(book.title, " Rating has been updated");
  }
}
```

```

getBooks(): void {
  this._dataService.getBooks().subscribe(
    books => this.books = books,
    error => this.updateMessage(<any>error, 'ERROR'));
}
}

```

To further emphasize the advantage of using observables we will implement a book searching capability. Start by modifying the getbooks method inside the data service to receive a search query. We will also introduce a new method called search which will utilize several operators which are part of the rxjs library what will allow us to efficiently search. The first operator is the debounceTime which will wait for a specific number of milliseconds before it triggers the search. We will also utilize another operator called distinctUntilchanged which will only trigger a new search when a new term is entered. Finally, we will utilize the switchMap operator which is useful in this case to prevent multiple outstanding searches to be triggered in parallel. The main difference between switchMap and other flattening operators is the cancelling effect. On each emission the previous inner observable (the result of the function you supplied) is cancelled and the new observable is subscribed. You can remember this by the phrase switch to a new observable. Here is the update data.service.ts file which reflected the above changes:

```

import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';

@Injectable()
export class DataService {

  _booksUrl:string = 'http://waelsbookservice.azurewebsites.net/books';

  constructor(private _http: Http) { }

```

```

private handleError(error: any) {
    let errMsg = (error.message) ? error.message : error.status ?
    `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Observable.throw(errMsg);
}

getBooks(): Observable<Ibook[]> {
    return this._http.get(this._booksUrl+"/GetBooks")
        .map((response: Response) => {
            let data: Ibook[] = <Ibook[]>response.json();
            return data;
        })
        .catch(this.handleError);
}

search(terms: Observable<string>) {
    return terms.debounceTime(400)
        .distinctUntilChanged()
        .switchMap(term => this.getBooks(term));
}

getBooks(query?: string): Observable<Ibook[]> {
    return this._http.get(this._booksUrl+"/GetBooks")
        .map((response: Response) => {
            let data: Ibook[] = <Ibook[]>response.json();
            if (query != null && query.length > 0) {
                data = data.filter(
                    data =>
                        data.author.includes(query) ||
                        data.title.includes(query)
                )
            }
            return data;
        })
        .catch(this.handleError);
}
}

```

Next we will need to modify the collection.component.ts file to utilize the newly updated service.

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar } from '@angular/material';
import { DataService } from '../services/data.service';
import { Subject } from 'rxjs/Subject';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>;

  showOperatingHours: boolean;
  openingTime: Date;
  closingTime: Date;
  searchTerm$ = new Subject<string>();

  constructor(private _snackBar: MatSnackBar, private _dataService: DataService) {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
  }

  ngOnInit() {
    this.books = this._dataService.getBooks();
    this._dataService.search(this.searchTerm$)
      .subscribe(books => {
        this.books = books;
      });
  }

  updateMessage(message: string, type: string): void {
    if (message) {
      this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
        duration: 3000
      });
    }
  }

  onRatingUpdate(book: Ibook): void {
    this.updateMessage(book.title, " Rating has been updated");
  }
}
```

Modify the collection.component.html file to display a search box.

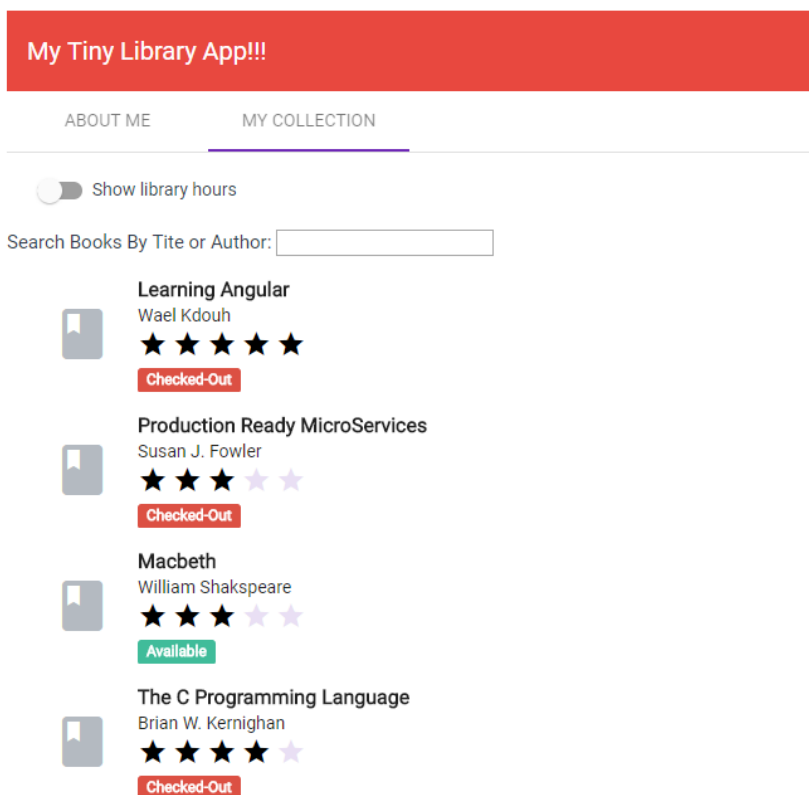
```
<h3>{{pageTitle}}&nbsp;  <mat-slide-toggle class="plr-15" color="primary"
  [(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library
  hours
</mat-slide-toggle>
</h3>
<div [hidden]="!showOperatingHours">
  <mat-card>
    <mat-card-subtitle>
      <strong>Operating Hours</strong>
    </mat-card-subtitle>
    <mat-card-content>{{openingTime | date:'shortTime'}} - {{closingTime |
date:'shortTime'}} (M-F)</mat-card-content>
  </mat-card>
</div>
<div>
  Search Books By Tite or Author:
  <input (keyup)="searchTerm$.next($event.target.value)">
  <mat-list>
    <mat-list-item *ngFor="let book of books">
      <mat-icon mat-list-icon>book</mat-icon>
      <h3 mat-line>
        <strong>{{book.title}}</strong>
      </h3>
      <div>
        <button mat-button (click)="openDialog(book.id)">
          <i class="material-icons">pageview</i>
          Dialog
        </button>
        <button mat-button (click)="openRoute(book.id)">
          <i class="material-icons"> pageview</i>
          Route
        </button>
        <button mat-button (click)="delete(book)">
          <i class="material-icons">delete_forever</i>
          Delete
        </button>
      </div>
      <p mat-line>
        <span>{{book.author}}</span>
      </p>
      <p mat-line>
```

```

        <my-rating [rating]="book.rating" [book]="book"
(ratingClicked)="onRatingUpdate($event)">
        </my-rating>
    </p>
    <p mat-line>
        <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-
success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>
    </p>
    </mat-list-item>
</mat-list>
<div class="text-right add-btn">
    <button mat-raised-button color="primary" (click)="addBook()">
        <i class="material-icons">add_box</i> ADD BOOK</button>
    </div>
</div>

```

You should now have a search button that allows you to dynamically filter the collection when you search by author or title.



Remember to commit lab 7.

## Lab 8 – Routing

When we created the application we added the `--routing` flag in order to generate a file called `app-routing.module.ts` which will contain the routing collection. **In case you forgot to add the `--routing` flag when you first created the application, issue the following command to add the `app-routing.module.ts`:**

**ng generate module app-routing**

In this lab we will populate the routing collection with some routes. Start by modifying the `app-routing.module.ts` file by adding the following routes:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AboutComponent } from '../about/about.component';
import { CollectionComponent } from '../collection/collection.component';

const routes: Routes = [
  {
    path: 'about',
    component: AboutComponent
  },
  {
    path: 'collection',
    component: CollectionComponent
  },
  {
    path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

The first route will be triggered when someone navigates to the about page, while the second route will be triggered when someone navigates to the collection page. The third route will redirect route. A redirect route requires a `pathMatch` property to tell the router how to match a URL to the path of a route. The router throws an error if you don't. In this app, the router should select the route to the `AboutComponent` only when the entire URL matches "", so set the `pathMatch` value to 'full'. Technically, `pathMatch = 'full'` results in a route hit when the



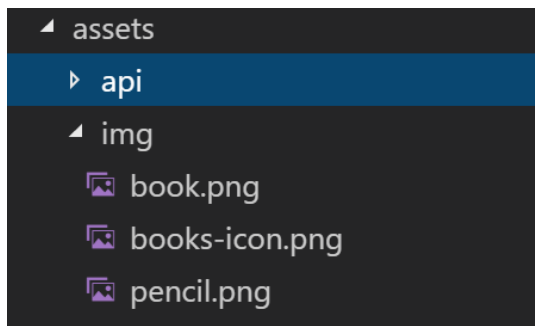
remaining, unmatched segments of the URL match ". In this example, the redirect is in a top level route so the remaining URL and the entire URL are the same thing. The other possible pathMatch value is 'prefix' which tells the router to match the redirect route when the remaining URL begins with the redirect route's prefix path.

Next we will introduce a tabbing system into our application. Create a tabs components by issuing the following command:

### ng generate component tabs

```
C:\Users\waelkdouh\Desktop\Angular-Workshop>ng generate component tabs
installing component
  create src\app\tabs\tabs.component.css
  create src\app\tabs\tabs.component.html
  create src\app\tabs\tabs.component.spec.ts
  create src\app\tabs\tabs.component.ts
  update src\app\app.module.ts
```

Copy the img folder that was provided to you and paste it under the assets folder. The updated project structure should now look like this as VS Code automatically picks up the newly included folder:



Modify the TabsComponent to include an array of navigation links which will be used later on to populate the different Angular Material tabs.

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-tabs',
  templateUrl: './tabs.component.html',
  styleUrls: ['./tabs.component.css']
})
export class TabsComponent implements OnInit {
```

```

    navLinks:Array<object> = [
    {
      path: 'about',
      label: 'ABOUT ME'
    },
    {
      path: 'collection',
      label: 'MY COLLECTION'
    }
  ];

  constructor() { }

  ngOnInit() {
  }
}

```

Modify the TabsComponent template (html file) to utilize the Angular Material tabs:

```

<nav mat-tab-nav-bar>
  <a mat-tab-link
    *ngFor="let link of navLinks"
    [routerLink]="link.path"
    routerLinkActive #rla="routerLinkActive"
    [active]="rla.isActive">
    {{link.label}}
  </a>
</nav>

<router-outlet></router-outlet>

```

Now that the tabs component has been created we will need to update our AppComponent template to start utilizing the newly introduced tabs component. Modify the AppComponent template to include both a material toolbar as well as the newly introduced TabsComponent:

```

<h1>My Tiny Library App</h1>
<my-collection></my-collection>

```

```

<div class="container">
  <mat-toolbar color="warn">
    <span>{{title}}</span>
    <span class="fill-remaining-space"></span>
    
  </mat-toolbar>
  <app-tabs></app-tabs>
</div>

```

Since we introduced the tabs component, we won't need the selector property on both the AboutComponent and CollectionComponent since we won't include them directly under a template, but instead we will navigate to them.

```

import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'my-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  pageTitle:string = 'About Me';
}

```

```

import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar } from '@angular/material';
import { DataService } from '../services/data.service';

@Component({
  selector: 'my-collection',
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

```

```

pageTitle: string;
books: Array<Ibook>;

showOperatingHours: boolean;
openingTime:Date;
closingTime:Date;

constructor(private _snackBar: MatSnackBar,private _dataService: DataService) {
  this.openingTime = new Date();
  this.openingTime.setHours(10, 0);
  this.closingTime = new Date();
  this.closingTime.setHours(15, 0);
}

ngOnInit() {
  this.books = this._dataService.getBooks();
  this._dataService.search(this.searchTerm$)
    .subscribe(books => {
      this.books = books;
    });
}

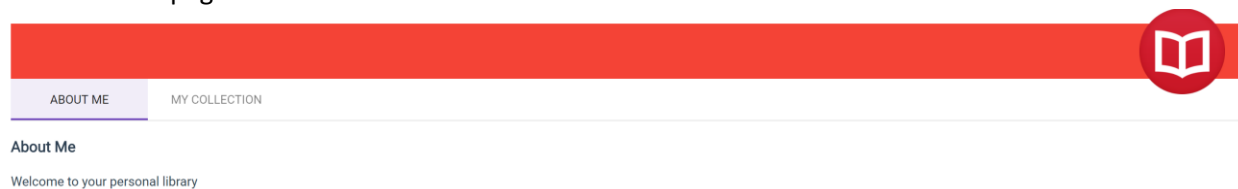
updateMessage(message: string, type: string): void {
  if (message) {
    this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
      duration: 3000
    });
  }
}

onRatingUpdate(book: Ibook): void {
  this.updateMessage(book.title, " Rating has been updated");
}

getBooks(): void {
  this._dataService.getBooks().subscribe(
    books => this.books = books,
    error => this.updateMessage(<any>error, 'ERROR'));
}
}

```

You should now have two tabs with the first tab showing the about page and the second tab showing the collection page.



Remember to commit lab 8.

## Lab 9 – Passing Parameters To a Route and Activating a Route with Code

In this lab we will add the ability to show the details of a book on a separate page. We will implement this feature using two different methods (in a real application you would choose one or the other). The first method will use a modal box and the second method will navigate to a totally new page. Both methods will provide the ability to modify the book rating, but the second method will allow cycling through all the available books without having to go back to the book library. In addition, we will also introduce the ability to delete a book from the book collection page.

Start by adding a new component called book-detail which will be used to display the book details:

### **ng generate component book-detail**

Before we start implementing the BookDetailComponent, we will need to modify the DataService to support fetching a single book, getting the next and previous book, updating a book, as well as deleting a book:

- Add a `getBook` method which would allow fetching a single book. Here is the signature of the new method: `getBook(id: number): Observable<Ibook>`
- Add a `getPreviousBookId` method which would allow fetching a single book. Here is the signature of the new method: `getPreviousBookId(id: number): Observable<number>`
- Add a `getNextBookId` method which would allow fetching a single book. Here is the signature of the new method: `getNextBookId(id: number): Observable<number>`
- Add a `updateBook` method which would allow fetching a single book. Here is the signature of the new method: `updateBook(book: Ibook): Observable<void>`
- Add a `deleteBook` method which would allow fetching a single book. Here is the signature of the new method: `deleteBook(id: number): Observable<void>`
- Add `getNextId(): Observable<number>` method which will be used later on in the lab to generate a unique id when we add a new book

```

import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';

@Injectable()
export class DataService {

  _booksUrl:string = 'http://waelsbookservice.azurewebsites.net/books';

  constructor(private _http: Http) { }

  private handleError(error: any) {
    let errMsg = (error.message) ? error.message : error.status ?
    `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Observable.throw(errMsg);
  }

  search(terms: Observable<string>) {
    return terms.debounceTime(400)
      .distinctUntilChanged()
      .switchMap(term => this.getBooks(term));
  }

  getBooks(query?: string): Observable<Ibook[]> {
    return this._http.get(this._booksUrl+"/GetBooks")
      .map((response: Response) => {
        let data: Ibook[] = <Ibook[]>response.json();
        if (query != null && query.length > 0) {
          data = data.filter(
            data =>
              data.author.includes(query) ||
              data.title.includes(query)
          )
        }
        return data;
      })
      .catch(this.handleError);
  }

  getBook(id: number): Observable<Ibook> {
    return this.getBooks()
      .map((books: Ibook[]) => books.find(b => b.id === id))
      // .do(data => console.log( JSON.stringify(data)))
      .catch(this.handleError);
  }

  getPreviousBookId(id: number): Observable<number> {
    return this.getBooks()
      .map((books: Ibook[]) => {

```

```

        return books[Math.max(0, books.findIndex(b => b.id === id) - 1)].id;
    })
    .catch(this.handleError);
}

getNextBookId(id: number): Observable<number> {
    return this.getBooks()
        .map((books: Ibook[]) => {
            return books[Math.min(books.length - 1, books.findIndex(b => b.id ===
                id) + 1)].id;
        })
        .catch(this.handleError);
}

updateBook(book: Ibook): Observable<void> {
    return this._http.put(this._booksUrl+"/modifybook", book)
        .catch(this.handleError);
}

deleteBook(id: number): Observable<void> {
    return this._http.delete(`${this._booksUrl+"/deletebook"}/${id}`)
        .catch(this.handleError);
}

getNextId(): Observable<number> {
    return this._http.get(this._booksUrl+"/GetNextId")
        .map((response: Response) => {
            let nextId: number = <number>response.json();
            return nextId;
        })
        .catch(this.handleError);
}
}

```

Next you will need to modify the BookDetailComponent to start utilizing the different operations that are now being exposed by the DataService. Mainly, the ability to modify a book rating and navigate between different books. We will also need to add a method that will allow us to return to the book collection page. We won't need the selector property as we will be navigating to this component. Also make sure you add the DataService as a provider as it will be needed for the material dialog to access the data.

```

import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { Subscription } from 'rxjs/Subscription';
import { ActivatedRoute, Router } from '@angular/router';

```

```

import { DataService } from '../services/data.service';
import { MatSnackBar } from '@angular/material';

@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
  providers: [DataService]
})
export class BookDetailComponent implements OnInit {

  bookId: number;

  book: Ibook;

  sub: Subscription;

  constructor(private _route: ActivatedRoute, private _router: Router, private
_dataService: DataService, private _snackBar: MatSnackBar)
  {}

  ngOnInit(): void {
    if (!this.bookId) {
      this.sub = this._route.params.subscribe(
        params => {
          let id = +params['id'];
          this.getBook(id);
        }
      );
      return;
    }
    this.getBook(this.bookId);
  }

  ngOnDestroy(): void {
    if (this.sub) {
      this.sub.unsubscribe();
    }
  }

  getBook(id: number): void {
    this._dataService.getBook(id).subscribe(
      book => this.book = book,
      error => this.updateMessage(<any>error, 'Error'));
  }

  onRatingUpdate(book: Ibook): void {
    this.updateBook(book);
  }

  updateMessage(message:string, type:string, actionText:string = 'DISMISS') {
    if (message) {
      this._snackBar.open(`${type}: ${message}`, actionText, {

```



```

        duration: 3000
    });
}
}

return(): void {
    this._router.navigate(['/collection']);
}

updateBook(book: Ibook): void {
    this._dataService.updateBook(book)
        .subscribe(
            books => {
                this._snackBar.open(`${book.title} has been updated!`, 'DISMISS', {
                    duration: 3000
                });
            },
            error => this.updateMessage(<any>error, 'ERROR'));
}

previous(): void {
    this._dataService
        .getPreviousBookId(this.book.id)
        .subscribe((bookId) => this._router.navigate(['/collection', bookId]));
}

next(): void {
    this._dataService
        .getNextBookId(this.book.id)
        .subscribe((bookId) => this._router.navigate(['/collection', bookId]));
}
}

```

Now modify the BookDetailComponent template to start utilizing the newly introduced capabilities. Notice that we are utilizing the angular \*ngIf directive to show the close button under different conditions than the other three buttons (previous, next, return). This will ensure that the close button only shows up for the dialog window whereas the other three buttons will show when navigating to a new page.

```

<div *ngIf="book">
    <mat-card>
        <mat-card-header>
            <mat-card-title><h4>{{book.title}}</h4></mat-card-title>
            <mat-card-subtitle>{{book.author}}</mat-card-subtitle>
            
        </mat-card-header>
        <mat-card-content>
            <div>

```

```

        <label><strong>Title:</strong></label>
        <span>{{book.title}}</span>
    </div>
    <div>
        <label><strong>Author:</strong></label>
        <span>{{book.author}}</span>
    </div>
    <div>
        <label><strong>Checked Out?</strong></label>
        <span>{{book.isCheckedOut ? 'Yes' : 'No'}}</span>
    </div>
    <div>
        <label><strong>Rating:</strong></label>
        <my-rating [rating]="book.rating" [book]="book"
            (ratingClicked)="onRatingUpdate($event)"></my-rating>
    </div>
</mat-card-content>
<mat-card-actions>
    <div class="text-right">
        <button mat-button mat-dialog-close *ngIf="bookId">
            <i class="material-icons">close</i>
            CLOSE
        </button>
        <button mat-button (click)="previous()" *ngIf="!bookId">
            <i class="material-icons">keyboard_arrow_left</i>
            PREVIOUS
        </button>
        <button mat-button (click)="next()" *ngIf="!bookId">NEXT
            <i class="material-icons">keyboard_arrow_right</i>
        </button>
        <button mat-button (click)="return()" *ngIf="!bookId">
            <i class="material-icons">keyboard_backspace</i>
            RETURN
        </button>
    </div>
</mat-card-actions>
</mat-card>
</div>

```

Modify book-detail.component.css as follows:

```

.mat-card {
    margin-top: 15px;
}
.mat-card-avatar {
    width: 64px;
    height: 64px;
}
.mat-card-header {
    margin-bottom: 10px;
}
.mat-card-header h4 {
    margin-bottom: 0;
}

```

```
margin-top: 5px;  
font-size: 18px;  
}
```

At this point the new book `BookDetailComponent` is ready to be utilized. In order to start utilizing it we will need to modify the `CollectionComponent` template to have the two different navigation options. We will also need to add the delete button to enable deleting books.

```

import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar, MatDialog } from '@angular/material';
import { DataService } from '../services/data.service';
import { Router } from '@angular/router';
import { BookDetailComponent } from '../book-detail/book-detail.component';

@Component({
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>;

  showOperatingHours: boolean;
  openingTime: Date;
  closingTime: Date;

  constructor(private _snackBar: MatSnackBar, private _dataService:
    DataService, private _dialog: MatDialog, private _router: Router)
  {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
  }

  ngOnInit() {
    this.books = this._dataService.getBooks();
    this._dataService.search(this.searchTerm$)
      .subscribe(books => {
        this.books = books;
      });
  }

  updateMessage(message: string, type: string): void {
    if (message) {
      this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
        duration: 3000
      });
    }
  }

  onRatingUpdate(book: Ibook): void {
    this.updateBook(book);
    this.updateMessage(book.title, " Rating has been updated");
  }

  updateBook(book: Ibook): void {
    this._dataService.updateBook(book)
      .subscribe(
        () => {

```

```

        this._snackBar.open(`${book.title}" has been updated!`, 'DISMISS', {
            duration: 3000
        });
    }, error => this.updateMessage(<any>error, 'ERROR'));
}

openDialog(bookId: number): void {
    let config = {width: '650px', height: '400x', position: {top: '50px'}};
    let dialogRef = this._dialog.open(BookDetailComponent, config);
    dialogRef.componentInstance.bookId = bookId;
    dialogRef.afterClosed().subscribe(res => {
        this.getBooks();
    });
}

openRoute(bookId: number): void {
    this._router.navigate(['/collection', bookId]);
}

delete(book: Ibook) {
    this._dataService
        .deleteBook(book.id)
        .subscribe(() => {
            this.getBooks()
            this._snackBar.open(`${book.title}" has been deleted!`,
                'DISMISS', {
                    duration: 3000
                });
        });
    }, error => this.updateMessage(<any>error, 'ERROR'));
}

getBooks(): void {
    this._dataService.getBooks().subscribe(
        books => this.books = books,
        error => this.updateMessage(<any>error, 'ERROR'));
}
}

```

Modify the CollectionComponent template to include the new buttons:

```

<h3>{{pageTitle}}&nbsp;<mat-slide-toggle class="plr-15" color="primary"
    [(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library
    hours</mat-slide-toggle></h3>
<div [hidden]="!showOperatingHours">
    <mat-card>
        <mat-card-subtitle><strong>Operating Hours</strong></mat-card-subtitle>
        <mat-card-content>{{openingTime | date:'shortTime'}} - {{closingTime |
    date:'shortTime'}} (M-F)</mat-card-content>
    </mat-card>
</div>
<div>
    Search Books By Title or Author:
    <input (keyup)="searchTerm$.next($event.target.value)">

```

```

<mat-list>
  <mat-list-item *ngFor="let book of books">
    <mat-icon mat-list-icon>book</mat-icon>
    <h3 mat-line><strong>{{book.title}}</strong></h3>
    <div>
      <button mat-button (click)="openDialog(book.id)">
        <i class="material-icons">pageview</i>
        Dialog
      </button>
      <button mat-button (click)="openRoute(book.id)">
        <i class="material-icons">pageview</i>
        Route
      </button>
      <button mat-button (click)="delete(book)">
        <i class="material-icons">delete_forever</i>
        Delete
      </button>
    </div>
    <p mat-line>
      <span>{{book.author}}</span>
    </p>
    <p mat-line>
      <my-rating [rating]="book.rating" [book]="book"
(ratingClicked)="onRatingUpdate($event)">
    </my-rating>
    </p>
    <p mat-line>
      <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-
success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>
    </p>
    </mat-list-item>
  </mat-list>
</div>

```

Finally, head to the Modify the AppRoutingModule and modify it to include a route to navigate a specific page. **Save.**

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AboutComponent } from './about/about.component';
import { CollectionComponent } from './collection/collection.component';
import { BookDetailComponent } from './book-detail/book-detail.component';

const routes: Routes = [

{
  path: 'about',
  component: AboutComponent
},
{
  path: 'collection',
  component: CollectionComponent
},

```

```

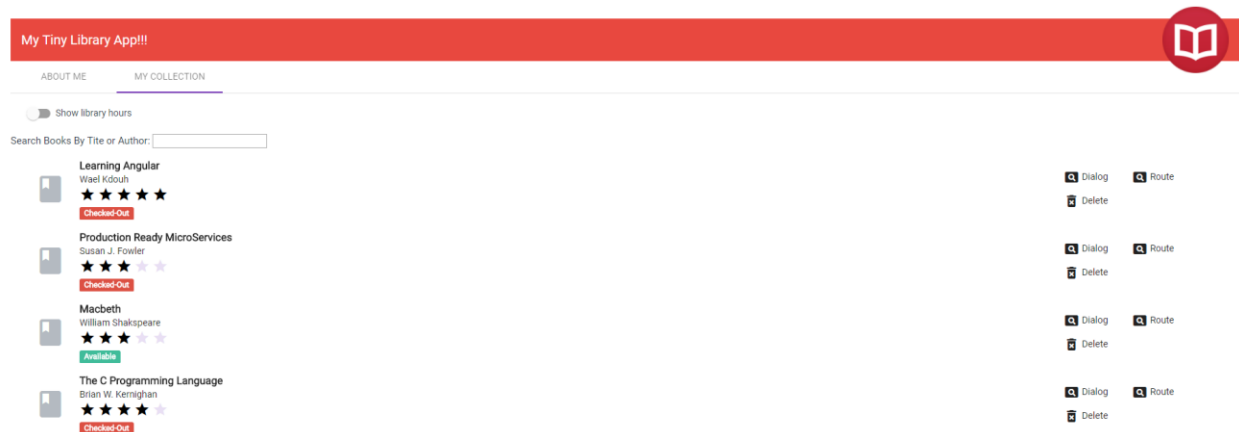
    {
      path: 'collection/:id',
      component: BookDetailComponent
    },

    {
      path: '',
      redirectTo: '/about',
      pathMatch: 'full'
    }
  ];

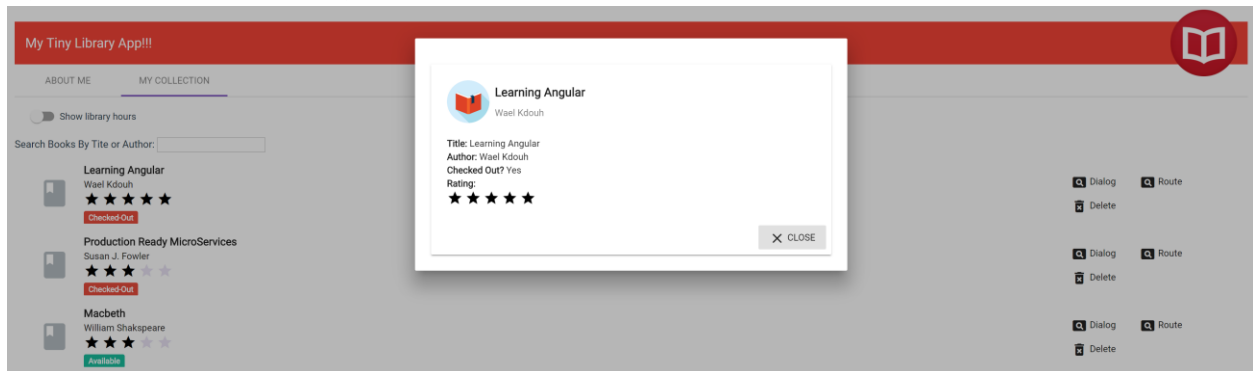
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

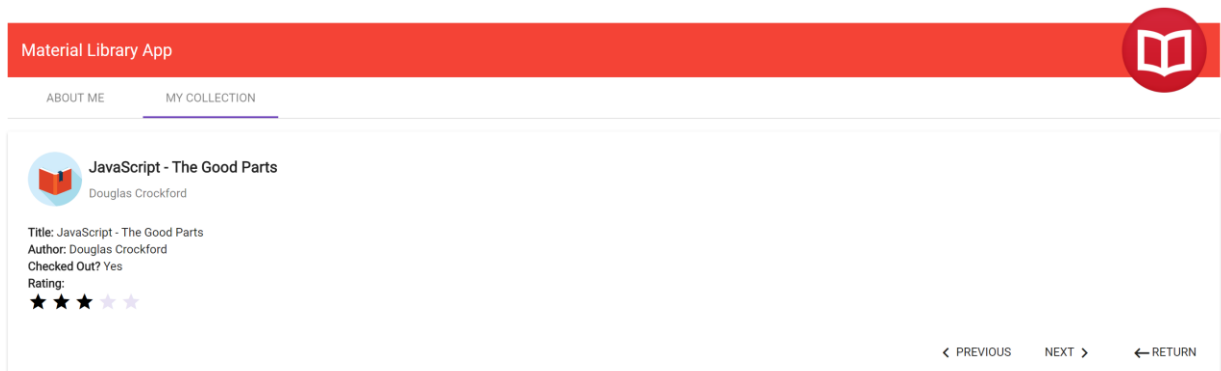
The book collection page should now provide you with three buttons which allow you to navigate to the book details using two different methods in addition to a delete button.



The modal dialog should only show the close button. You should also be able to update the book rating.



Navigating to a new page should show three buttons. The previous and next buttons should allow you to cycle through the different books, whereas the return button should allow you to return to the book collection page. You should also be able to update the book rating.



Remember to commit lab 9.



## Lab 10 – Protecting Routes with Guards

In this lab we will introduce a route guard which allows us to prevent navigating to an invalid book directly using the url. For example, entering `http://localhost:4200/collection/-1` is invalid and thus we will redirect to the book collection page upon entering such an invalid url.

Start by adding a book guard service using the following command:

### **ng generate service guards/book-guard**

Add a `canActivate` method to the the `BookGuardService` which has the following signature `canActivate(route: ActivatedRouteSnapshot)`

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, Router } from '@angular/router';
import { DataService } from '../services/data.service';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of';

@Injectable()
export class BookGuardService {

  constructor(private _router: Router, private _dataService: DataService) { }

  canActivate(route: ActivatedRouteSnapshot) {
    // parse the book id from the route
    let id = +route.url[1].path;
    if (isNaN(id)) {
      // start a new navigation to redirect to list page
      this._router.navigate(['/collection']);
      // abort current navigation
      return false;
    }

    return this._dataService.canActivate(id).map(result => {
      if (result) {
        return true;
      }
      this._router.navigate(['/collection']);
      return Observable.of(false);
    }).catch(() => {
      this._router.navigate(['/collection']);
      return Observable.of(false);
    });
  }
}
```

Modify the DataService to include a canActivate method which will call a backend service that checks whether the book id is valid before attempting to navigate to the book in question.

- Add canActivate (id): Observable<boolean> to the DataService

```
import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';

@Injectable()
export class DataService {

  _booksUrl:string = 'http://waelsbookservice.azurewebsites.net/books';

  constructor(private _http: Http) { }

  private handleError(error: any) {
    let errMsg = (error.message) ? error.message : error.status ?
    `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Observable.throw(errMsg);
  }

  search(terms: Observable<string>) {
    return terms.debounceTime(400)
      .distinctUntilChanged()
      .switchMap(term => this.getBooks(term));
  }

  getBooks(query?: string): Observable<Ibook[]> {
    return this._http.get(this._booksUrl+"/GetBooks")
      .map((response: Response) => {
        let data: Ibook[] = <Ibook[]>response.json();
        if (query != null && query.length > 0) {
          data = data.filter(
            data =>
              data.author.includes(query) ||
              data.title.includes(query)
          )
        }
        return data;
      })
      .catch(this.handleError);
  }
}
```

```

getBook(id: number): Observable<Ibook> {
    return this.getBooks()
        .map((books: Ibook[]) => books.find(b => b.id === id))
        // .do(data => console.log( JSON.stringify(data)))
        .catch(this.handleError);
}

getPreviousBookId(id: number): Observable<number> {
    return this.getBooks()
        .map((books: Ibook[]) => {
            return books[Math.max(0, books.findIndex(b => b.id === id) - 1)].id;
        })
        .catch(this.handleError);
}

getNextBookId(id: number): Observable<number> {
    return this.getBooks()
        .map((books: Ibook[]) => {
            return books[Math.min(books.length - 1, books.findIndex(b => b.id ===
            id) + 1)].id;
        })
        .catch(this.handleError);
}

updateBook(book: Ibook): Observable<void> {
    return this._http.put(this._booksUrl+"/modifybook", book)
        .catch(this.handleError);
}

deleteBook(id: number): Observable<void> {
    return this._http.delete(`${this._booksUrl+"/deletebook"}/${id}`)
        .catch(this.handleError);
}

addBook(book: Ibook): Observable<void> {
    return this._http.post(this._booksUrl+"/addbook", book)
        .catch(this.handleError);
}

getNextId(): Observable<number> {
    return this._http.get(this._booksUrl+"/GetNextId")
        .map((response: Response) => {
            let nextId: number = <number>response.json();
            return nextId;
        })
        .catch(this.handleError);
}

```

```

    canActivate(id): Observable<boolean> {
      return this._http.get(`${this._booksUrl+"/canactivate"}/${id}`)
        .map((response: Response) => {
          let canActivate: boolean = <boolean>response.json();
          return canActivate;
        })
        .catch(this.handleError);
    }
  }
}

```

In order to start utilizing the newly introduced book guard we have to apply to one of the defined routes. Modify the AppRoutingModuleModule to include a guard on the book detail route to prevent routing to an invalid book.

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AboutComponent } from './about/about.component';
import { CollectionComponent } from './collection/collection.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { BookGuardService } from './guards/book-guard.service';

const routes: Routes = [
  {
    path: 'about',
    component: AboutComponent
  },
  {
    path: 'collection',
    component: CollectionComponent
  },
  {
    path: 'collection/:id',
    canActivate: [BookGuardService],
    component: BookDetailComponent
  },
  {
    path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Finally modify AppModule to include the BookGuardService. **Save.**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { FormsModule } from '@angular/forms';
import { MatListModule, MatCardModule, MatSlideToggleModule, MatDialogModule,
  MatIconModule, MatInputModule, MatSnackBarModule, MatTabsModule,
  MatButtonModule, MatLineModule, MatToolbarModule } from '@angular/material';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { CollectionComponent } from './collection/collection.component';
import { RatingCategoryPipe } from './pipes/rating-category.pipe';
import { RatingComponent } from './rating/rating.component';
import { HttpModule } from '@angular/http';
import { TabsComponent } from './tabs/tabs.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { BookGuardService } from './guards/book-guard.service';

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    CollectionComponent,
    RatingCategoryPipe,
    RatingComponent,
    TabsComponent,
    BookDetailComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
    MatToolbarModule,
    BrowserAnimationsModule,
    HttpModule
  ],
  providers: [BookGuardService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

If you attempt to navigate to <http://localhost:4200/collection/-1> you should get rerouted back to the book collection page.

Remember to commit lab 10.

## Lab 11 – Adding Forms

So far we have mainly managed existing data through the application, but haven't enabled adding to the existing data. In this lab we will provide the user with the ability to add a new book. Along the way we will demonstrate the built in feature in Angular that allows for easy validation for forms.

Start by adding a NewBookComponent using the following command:

**ng generate component new-book**

Modify the new-book.component.ts file:

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatDialogRef } from '@angular/material';

@Component({
  selector: 'app-new-book',
  templateUrl: './new-book.component.html',
  styleUrls: ['./new-book.component.css']
})
export class NewBookComponent implements OnInit {

  book:Ibook;

  constructor(private _dialogRef: MatDialogRef<NewBookComponent>) { }

  ngOnInit() {
    this.book = {
      id: 0,
      title: '',
      author: '',
      isCheckedOut: false,
      rating: 0
    }
  }

  cancel(): void {
    this._dialogRef.close();
  }

  save(): void{
    this._dialogRef.close(this.book);
  }
}
```

Modify the new-book.component.css file:

```
.mat-card-avatar {  
  width: 64px;  
  height: 64px;  
  -webkit-border-radius: 0;  
  -moz-border-radius: 0;  
  border-radius: 0;  
}  
.mat-card-header {  
  margin-bottom: 25px;  
}  
.mat-card-header h4 {  
  margin-bottom: 0;  
  margin-top: 15px;  
  font-size: 24px;  
}  
.mat-input-container {  
  width: 100%;  
  line-height: 2.23;  
  margin-bottom: 15px;  
}  
p.rating-label {  
  color: rgba(0,0,0,.38);  
}  
.mat-slide-toggle {  
  height: 45px;  
  line-height: 45px;  
  margin: 10px 0;  
}
```



Modify the new-book.component.html file:

```
<form #newBookForm="ngForm">
  <mat-card>
    <mat-card-header>
      <mat-card-title><h4>Add New Book</h4></mat-card-title>
      
    </mat-card-header>
    <mat-card-content>
      <mat-input-container>
        <input matInput placeholder="Book Title" [(ngModel)]="book.title"
name="title" required />
      </mat-input-container>
      <mat-input-container>
        <input matInput placeholder="Author" [(ngModel)]="book.author"
name="author" required />
      </mat-input-container>
      <p class="rating-label">Rating</p>
      <my-rating [rating]="book.rating" [book]="book">
      </my-rating>
      <mat-slide-toggle color="primary" [(ngModel)]="book.isCheckedOut"
name="checkedOut">Checked out?</mat-slide-toggle>
    </mat-card-content>
    <mat-card-actions>
      <div class="text-right">
        <button type="submit" mat-button (click)="save()" color="warn"
[disabled]="newBookForm.form.invalid"><i class="material-icons">save</i>SAVE</button>
        <button type="reset" mat-button (click)="cancel()"><i class="material-
icons">cancel</i>CANCEL</button>
      </div>
    </mat-card-actions>
  </mat-card>
</form>
```

Modify AppModule to include the NewBookComponent as an entryComponent as shown below. You can read more about entry components [here](#).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { FormsModule } from '@angular/forms';
import { MatListModule, MatCardModule, MatSlideToggleModule, MatDialogModule,
  MatIconModule, MatInputModule, MatSnackBarModule, MatTabsModule,
  MatButtonModule, MatLineModule, MatToolbarModule } from '@angular/material';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { CollectionComponent } from './collection/collection.component';
import { RatingCategoryPipe } from './pipes/rating-category.pipe';
```

```

import { RatingComponent } from './rating/rating.component';
import { HttpModule } from '@angular/http';
import { TabsComponent } from './tabs/tabs.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { BookGuardService } from './guards/book-guard.service';
import { NewBookComponent } from './new-book/new-book.component';

@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    CollectionComponent,
    RatingCategoryPipe,
    RatingComponent,
    TabsComponent,
    BookDetailComponent,
    NewBookComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
    MatToolbarModule,
    BrowserAnimationsModule,
    HttpModule
  ],
  providers: [BookGuardService],
  bootstrap: [AppComponent],
  entryComponents: [
    NewBookComponent
  ]
})
export class AppModule { }

```

Modify CollectionComponent template to include a new book button:

```

<h3>{{pageTitle}}&nbsp;<mat-slide-toggle class="plr-15" color="primary"
[(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library
hours</mat-slide-toggle></h3>
<div [hidden]="!showOperatingHours">
  <mat-card>
    <mat-card-subtitle><strong>Operating Hours</strong></mat-card-subtitle>

```

```

    <mat-card-content>{{openingTime | date:'shortTime'}} - {{closingTime |
date:'shortTime'}} (M-F)</mat-card-content>
  </mat-card>
</div>
<div>
  Search Books By Tite or Author:
  <input (keyup)="searchTerm$.next($event.target.value)">
  <mat-list>
    <mat-list-item *ngFor="let book of books">
      <mat-icon mat-list-icon>book</mat-icon>
      <h3 mat-line><strong>{{book.title}}</strong></h3>
      <div>
        <button mat-button (click)="openDialog(book.id)">
          <i class="material-icons">pageview</i>
          Dialog
        </button>
        <button mat-button (click)="openRoute(book.id)">
          <i class="material-icons"> pageview</i>
          Route
        </button>
        <button mat-button (click)="delete(book)">
          <i class="material-icons">delete_forever</i>
          Delete
        </button>
      </div>
      <p mat-line>
        <span>{{book.author}}</span>
      </p>
      <p mat-line>
        <my-rating [rating]="book.rating" [book]="book"
(ratingClicked)="onRatingUpdate($event)">
        </my-rating>
      </p>
      <p mat-line>
        <span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-
success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>
      </p>
    </mat-list-item>
  </mat-list>
  <div class="text-right add-btn">
    <button mat-raised-button color="primary" (click)="addBook()"><i
class="material-icons">add_box</i> ADD BOOK</button>
  </div>
</div>

```

Modify the CollectionComponent to include the addBook() method:

```
import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar, MatDialog } from '@angular/material';
import { DataService } from '../services/data.service';
import { Router } from '@angular/router';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { NewBookComponent } from '../new-book/new-book.component';

@Component({
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>;
  showOperatingHours: boolean;
  openingTime: Date;
  closingTime: Date;
  searchTerm$ = new Subject<string>();

  constructor(private _snackBar: MatSnackBar, private _dataService:
    DataService, private _dialog: MatDialog, private _router: Router) {
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
  }

  ngOnInit() {
    this.books = this._dataService.getBooks();
    this._dataService.search(this.searchTerm$)
      .subscribe(books => {
        this.books = books;
      });
  }

  updateMessage(message: string, type: string): void {
    if (message) {
      this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
        duration: 3000
      });
    }
  }

  onRatingUpdate(book: Ibook): void {
    this.updateBook(book);
    this.updateMessage(book.title, " Rating has been updated");
  }

  updateBook(book: Ibook): void {
    this._dataService.updateBook(book)
      .subscribe(
```

```

    () => {
      this._snackBar.open(`${book.title}" has been updated!`, 'DISMISS', {
        duration: 3000
      });
    }, error => this.updateMessage(<any>error, 'ERROR'));
  }

  openDialog(bookId: number): void {
    let config = {width: '650px', height: '400x', position: {top: '50px'}};
    let dialogRef = this._dialog.open(BookDetailComponent, config);
    dialogRef.componentInstance.bookId = bookId;
    dialogRef.afterClosed().subscribe(res => {
      this.getBooks();
    });
  }

  openRoute(bookId: number): void {
    this._router.navigate(['/collection', bookId]);
  }

  delete(book: Ibook) {
    this._dataService
      .deleteBook(book.id)
      .subscribe(() => {
        this.getBooks()
        this._snackBar.open(`${book.title}" has been deleted!`,
          'DISMISS', {
            duration: 3000
          });
      });
    }, error => this.updateMessage(<any>error, 'ERROR'));
  }

  getBooks(): void {
    this._dataService.getBooks().subscribe(
      books => this.books = books,
      error => this.updateMessage(<any>error, 'ERROR'));
  }

  addBook(): void {
    let config = {width: '650px', height: '650px', position: {top: '50px'},
      disableClose: true};
    let dialogRef = this._dialog.open(NewBookComponent, config);
    dialogRef.afterClosed().subscribe(newBook => {
      if (newBook) {
        this._dataService.getNextId().subscribe(
          (id) =>
          {
            newBook.id = id;
            this._dataService.addBook(newBook)
              .subscribe(
                () =>
                {
                  this.getBooks()
                  this._snackBar.open('Book added!',
                    'DISMISS', {

```

```

        duration: 3000
    });
},
error => this.updateMessage(<any>error, 'ERROR'));
});
}
});
}
}
}

```

Finally, modify the DataService to include an addBook() method with the following signature: addBook(book: Ibook): Observable<void>. **Save.**

```

import { Injectable } from '@angular/core';
import { Ibook } from '../ibook';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/debounceTime';
import 'rxjs/add/operator/distinctUntilChanged';
import 'rxjs/add/operator/switchMap';

@Injectable()
export class DataService {

    _booksUrl: string = 'http://waelsbookservice.azurewebsites.net/books';

    constructor(private _http: Http) { }

    private handleError(error: any) {
        let errMsg = (error.message) ? error.message : error.status ?
        `${error.status} - ${error.statusText}` : 'Server error';
        console.error(errMsg);
        return Observable.throw(errMsg);
    }

    search(terms: Observable<string>) {
        return terms.debounceTime(400)
            .distinctUntilChanged()
            .switchMap(term => this.getBooks(term));
    }

    getBooks(query?: string): Observable<Ibook[]> {
        return this._http.get(this._booksUrl+"/GetBooks")
            .map((response: Response) => {
                let data: Ibook[] = <Ibook[]>response.json();
                if (query != null && query.length > 0) {
                    data = data.filter(
                        data =>
                            data.author.includes(query) ||
                            data.title.includes(query)
                    );
                }
            });
    }
}

```

```

        )
    }
    return data;
})
.catch(this.handleError);
}

getBook(id: number): Observable<Ibook> {
    return this.getBooks()
        .map((books: Ibook[]) => books.find(b => b.id === id))
        // .do(data => console.log( JSON.stringify(data)))
        .catch(this.handleError);
}

getPreviousBookId(id: number): Observable<number> {
    return this.getBooks()
        .map((books: Ibook[]) => {
            return books[Math.max(0, books.findIndex(b => b.id === id) - 1)].id;
        })
        .catch(this.handleError);
}

getNextBookId(id: number): Observable<number> {
    return this.getBooks()
        .map((books: Ibook[]) => {
            return books[Math.min(books.length - 1, books.findIndex(b => b.id ===
            id) + 1)].id;
        })
        .catch(this.handleError);
}

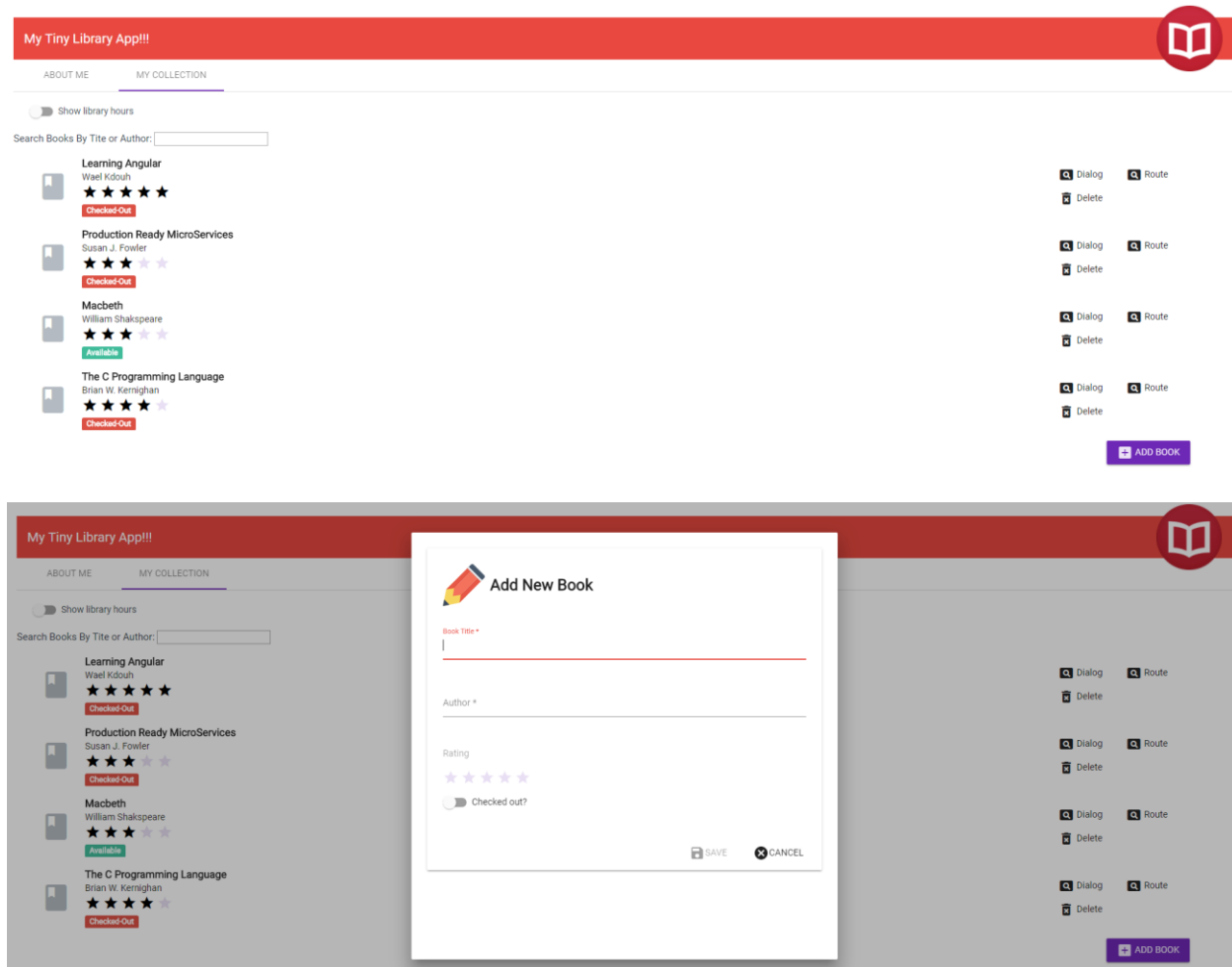
updateBook(book: Ibook): Observable<void> {
    return this._http.put(this._booksUrl+"/modifybook", book)
        .catch(this.handleError);
}

deleteBook(id: number): Observable<void> {
    return this._http.delete(`${this._booksUrl+"/deletebook"}/${id}`)
        .catch(this.handleError);
}

addBook(book: Ibook): Observable<void> {
    return this._http.post(this._booksUrl+"/addbook", book)
        .catch(this.handleError);
}
}

```

You should now have a button on the lower right corner of the book collection page that allows you to add a new book. In addition, you should now have a modal window that allows you to add a new book along with the necessary validations.



Remember to commit lab 11.



## Lab 12 – Lazy Loading

In this lab you will enable lazy loading the book collection tab in order to avoid loading the whole application at once (aka eager loading). This will ensure better performance especially when loading the application using a slow connection or when you are dealing with a huge application.

Start by modifying the collection component route under AppRoutingModuleModule file to be lazy loaded. You can achieve this by replacing the “component” property with the “loadChildren” property which will point to the collection module which will be lazy loaded (the new collection module will be added later in this lab). Also, make sure you remove the collection/:id path from this file as it will be moved to the collection-routing.module.ts file which will be added later in this lab.

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AboutComponent } from 'app/about/about.component';
import { CollectionComponent } from 'app/collection/collection.component';
import { BookGuardService } from 'app/guards/book-guard.service';
import { BookDetailComponent } from 'app/book-detail/book-detail.component';

const routes: Routes = [
  {
    path: 'about',
    component: AboutComponent
  },
  {
    path: 'collection',
    component: CollectionComponent
    loadChildren: 'app/collection/collection.module#CollectionModule'
  },
  {
    path: 'collection/:id',
    canActivate: [BookGuardService],
    component: BookDetailComponent
  },
  {
    path: '',
    redirectTo: '/about',
    pathMatch: 'full'
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Now that you have modified the main routing table to lazy load the collection module, you will need to add the new collection module itself in addition to its associated routing table. Execute the following command which will create both the new collection module as well as its associated routing table:

**ng generate module collection --routing**

```
create src/app/collection/collection-routing.module.ts (253 bytes)
create src/app/collection/collection.module.ts (295 bytes)
```

You will need to remove all of the pieces that are related to the collection page from the app.module.ts file into the newly created collection.module.ts. Here is the newly created collection.module.ts file:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { CollectionRoutingModule } from './collection-routing.module';

import { CollectionComponent } from './collection.component';
import { RatingCategoryPipe } from '../pipes/rating-category.pipe';
import { RatingComponent } from '../rating/rating.component';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { NewBookComponent } from '../new-book/new-book.component';
import { DataService } from '../services/data.service';
import { MatListModule, MatTabsModule, MatSnackBarModule,
    MatDialogModule, MatCardModule, MatIconModule,
    MatSlideToggleModule, MatButtonModule, MatLineModule,
    MatInputModule, MatToolbarModule } from '@angular/material';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { BookGuardService } from '../guards/book-guard.service';

@NgModule({
  imports: [
    CommonModule,
    CollectionRoutingModule,
    FormsModule,
    HttpModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
```

```

    MatToolbarModule
  ],
  entryComponents: [
    NewBookComponent
  ],
  declarations: [
    CollectionComponent,
    RatingComponent,
    BookDetailComponent,
    NewBookComponent,
    RatingCategoryPipe
  ],
  providers: [
    BookGuardService,
    DataService
  ]
})
export class CollectionModule { }

```

Modify the newly created collection-routing.module.ts file to include the different routes pertaining to the book collection.

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { CollectionComponent } from '../collection.component';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { BookGuardService } from '../guards/book-guard.service';

const routes: Routes = [
  {
    path: '', component: CollectionComponent
  },
  {
    path: ':id',
    canActivate: [BookGuardService],
    component: BookDetailComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class CollectionRoutingModule { }

```

Modify the AppModule to exclude the collection component as well as all the other related components (rating component, book detail component, etc.).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { FormsModule } from '@angular/forms';
import { MatListModule, MatCardModule, MatSlideToggleModule, MatDialogModule,
  MatIconModule, MatInputModule, MatSnackBarModule, MatTabsModule,
  MatButtonModule, MatLineModule, MatToolbarModule } from '@angular/material';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { CollectionComponent } from './collection/collection.component';
import { RatingCategoryPipe } from './pipes/rating-category.pipe';
import { RatingComponent } from './rating/rating.component';
import { HttpModule } from '@angular/http';
import { TabsComponent } from './tabs/tabs.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { BookGuardService } from './guards/book-guard.service';
import { NewBookComponent } from './new-book/new-book.component';

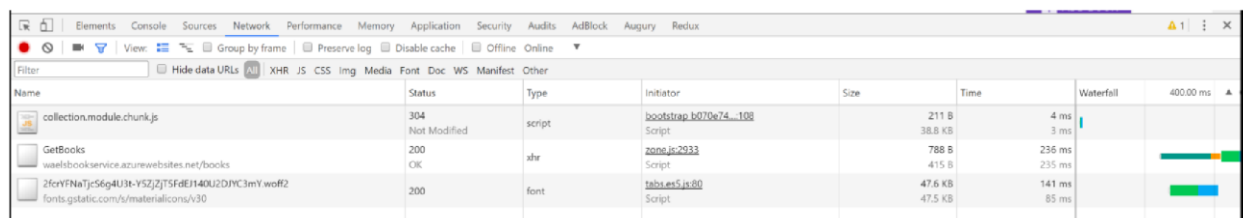
@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    CollectionComponent,
    RatingCategoryPipe,
    RatingComponent,
    TabsComponent,
    BookDetailComponent,
    NewBookComponent
  ],
```

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule,  
  MatListModule,  
  MatTabsModule,  
  MatSnackBarModule,  
  MatDialogModule,  
  MatCardModule,  
  MatIconModule,  
  MatSlideToggleModule,  
  MatButtonModule,  
  MatLineModule,  
  MatInputModule,  
  MatToolbarModule,  
  BrowserAnimationsModule,  
  HttpClientModule  
],  
providers: [BookGuardService],  
bootstrap: [AppComponent],  
entryComponents: [  
  NewBookComponent  
]  
})  
export class AppModule { }
```

Finally, modify the BookGuardService to look for the first parameter in the route instead of the second as the id is now the first parameter.

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router } from '@angular/router';
@Injectable()
export class BookGuardService implements CanActivate {
  constructor(private _router: Router) {
  }
  canActivate(route: ActivatedRouteSnapshot): boolean {
    let id = +route.url[0].path;
    if (isNaN(id) || id < 1) {
      // start a new navigation to redirect to list page
      this._router.navigate(['/collection']);
      // abort current navigation
      return false;
    };
    return true;
  }
}
```

In order to test the lazy loading feature, make sure you have the F12 tools open and ensure that clicking on the book collection tab loads the following files:



The screenshot shows the Chrome DevTools Network tab with the following resources loaded:

Name	Status	Type	Initiator	Size	Time	Waterfall	400.00 ms
collection.module.chunk.js	304 Not Modified	script	bootstrap.b070e74...108	211 B	4 ms		
GetBooks	200 OK	xhr	zone.js:2933	788 B	236 ms		
2fc9fNaTj56g4U3k-Y5Z2jT5f8Ej140U2DjYC3mY_wor2	200 OK	font	tabu.es.js:00	47.6 KB	141 ms		
fonts.gstatic.com/s/materialicons/v30		font		47.5 KB	85 ms		

Remember to commit lab 12.

Congratulations, you should now have a fully functional application which allows you to navigate between tabs, add new books, delete books, search books, navigate the detailed descriptions of the different books, cycle through the different available books, and lazy load the book collection tab.

## Lab 13 – Route Resolves

If you are working on this lab then this means you are now officially an Angular ninja as this is meant to be a bonus lab. At this point you have a fully functional application, but one thing you may have noticed is that the book collection tab renders couple seconds before the data gets the chance to be fetched (typically from a restful backend service). As a matter of fact In this lab we will introduce route resolves which allows us to pre-fetch the data a component needs before it is initialized.

Start by creating a new file under the collection folder called collection.resolver.ts which will hold the code for our collection route resolver.

```
import { Resolve } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import { Ibook } from '../ibook';
import { DataService } from '../services/data.service';
import { Injectable } from '@angular/core';

@Injectable()
export class CollectionResolver implements Resolve<Ibook[]> {
  constructor(private _dataService: DataService) {}

  /**
   * resolve() is the method we have to implement for the Resolve interface.
   * The router will call this method when the users visits the route.
   * We can return Promises, Observables or any other value here.
   * When it's a Promise or Observable, the Angular Router waits for
   * the result and then displays the page (which is what we want).
   */
  resolve(): Observable<Ibook[]> {
    return this._dataService.getBooks();
  }
}
```



Modify the collection route to utilize the newly introduced route resolver. Basically, we will modify the collection route to resolve the data before it allows navigation to that route. This is achieved by adding the resolve property to the route. Notice that inside the resolve object we are exposing a property called books which will include the collection of books generated by the CollectionResolver class. The books collection will later be accessible from inside the collection component. Here is the updated collection-routing.module.ts file:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { CollectionComponent } from './collection.component';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { BookGuardService } from '../guards/book-guard.service';
import { CollectionResolver } from './collection.resolver';

const routes: Routes = [
  {
    path: '', component: CollectionComponent,
    resolve: { books: CollectionResolver }
  },
  {
    path: ':id',
    canActivate: [BookGuardService],
    component: BookDetailComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class CollectionRoutingModule { }
```

Now that we have the route resolver in place its time to start utilizing it inside our collection component. The route resolver will be provided to the collection component using dependency injection. Thus, we will need to add the CollectionResolver to the list of providers under collection.module.ts.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { CollectionRoutingModule } from './collection-routing.module';
import { CollectionComponent } from './collection.component';
import { RatingCategoryPipe } from '../pipes/rating-category.pipe';
import { RatingComponent } from '../rating/rating.component';
import { NewBookComponent } from '../new-book/new-book.component';
import { DataService } from '../services/data.service';
import { MatListModule, MatTabsModule, MatSnackBarModule,
    MatDialogModule, MatCardModule, MatIconModule,
    MatSlideToggleModule, MatButtonModule, MatLineModule,
    MatInputModule, MatToolbarModule } from '@angular/material';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { BookGuardService } from '../guards/book-guard.service';
import { CollectionResolver } from './collection.resolver';
```

```
@NgModule({
  imports: [
    CommonModule,
    CollectionRoutingModule,
    FormsModule,
    HttpClientModule,
    MatListModule,
    MatTabsModule,
    MatSnackBarModule,
    MatDialogModule,
    MatCardModule,
    MatIconModule,
    MatSlideToggleModule,
    MatButtonModule,
    MatLineModule,
    MatInputModule,
    MatToolbarModule
  ],
  entryComponents: [
    NewBookComponent
  ],
  declarations: [
    CollectionComponent,
    RatingComponent,
    BookDetailComponent,
    NewBookComponent,
    RatingCategoryPipe
```

```

    ],
    providers: [
        BookGuardService,
        DataService,
        CollectionResolver
    ]
})
export class CollectionModule { }

```

Finally, we will need to modify the collection component to start utilizing the newly introduced route resolver. Modify the component to access the books collection which was generated by the resolver instead of going directly to the data service. Note that the route resolver is still going through the data service to fetch the data, but it ensures that the page is not rendered until that data is fetched. **Save.**

```

import { Component, OnInit } from '@angular/core';
import { Ibook } from '../ibook';
import { MatSnackBar, MatDialog } from '@angular/material';
import { DataService } from '../services/data.service';
import { Router, ActivatedRoute } from '@angular/router';
import { BookDetailComponent } from '../book-detail/book-detail.component';
import { NewBookComponent } from '../new-book/new-book.component';
import { Subject } from 'rxjs/Subject';

@Component({
  templateUrl: './collection.component.html',
  styleUrls: ['./collection.component.css']
})
export class CollectionComponent implements OnInit {

  pageTitle: string;
  books: Array<Ibook>;
  searchTerm$ = new Subject<string>();

  showOperatingHours: boolean;
  openingTime: Date;
  closingTime: Date;

```

```

constructor(private _snackBar: MatSnackBar,private _dataService:
    DataService,private _dialog: MatDialog, private _router: Router,private
    _route: ActivatedRoute)

{
    this.openingTime = new Date();
    this.openingTime.setHours(10, 0);
    this.closingTime = new Date();
    this.closingTime.setHours(15, 0);
}

ngOnInit() {
    this.books = this._dataService.getBooks();
    this.books= this._route.snapshot.data['books'];
    this._dataService.search(this.searchTerm$)
        .subscribe(books => {
            this.books = books;
        });
}

updateMessage(message: string, type: string): void {
    if (message) {
        this._snackBar.open(`${type}: ${message}`, 'DISMISS', {
            duration: 3000
        });
    }
}

onRatingUpdate(book: Ibook): void {
    this.updateBook(book);
    this.updateMessage(book.title, " Rating has been updated");
}

updateBook(book: Ibook): void {
    this._dataService.updateBook(book)
        .subscribe(
            () => {
                this._snackBar.open(`${book.title} has been updated!`, 'DISMISS', {
                    duration: 3000
                });
            },error => this.updateMessage(<any>error, 'ERROR'));
}

openDialog(bookId:number): void {
    let config = {width: '650px', height: '400x', position: {top: '50px'}};
    let dialogRef = this._dialog.open(BookDetailComponent, config);
    dialogRef.componentInstance.bookId = bookId;
    dialogRef.afterClosed().subscribe(res => {

```

```

        this.getBooks();
    });
}

openRoute(bookId: number): void {
    this._router.navigate(['/collection', bookId]);
}

delete(book: Ibook) {
    this._dataService
        .deleteBook(book.id)
        .subscribe(() => {
            this.getBooks()
            this._snackBar.open(`${book.title}" has been deleted!`,
                'DISMISS', {
                    duration: 3000
                });
        }, error => this.updateMessage(<any>error, 'ERROR'));
}

getBooks(): void {
    this._dataService.getBooks().subscribe(
        books => this.books = books,
        error => this.updateMessage(<any>error, 'ERROR'));
}

addBook(): void {
    let config = {width: '650px', height: '650px', position: {top: '50px'},
        disableClose: true};
    let dialogRef = this._dialog.open(NewBookComponent, config);
    dialogRef.afterClosed().subscribe(newBook => {
        if (newBook) {
            this._dataService.getNextId().subscribe(
                (id) =>
                {
                    newBook.id = id;
                    this._dataService.addBook(newBook)
                    .subscribe(
                        () =>
                        {
                            this.getBooks()
                            this._snackBar.open(`Book added!`,
                                'DISMISS', {
                                    duration: 3000
                                });
                        });
                }
            );
        }
    });
}

```

```
    },  
    error => this.updateMessage(<any>error, 'ERROR'));  
  });  
}  
});  
}  
  
}
```

You should now notice that clicking on the book collection tab does not switch to that tab until the data is fetched. You may need to stop the development server and rerun it to notice that difference.