**Angular Workshop**

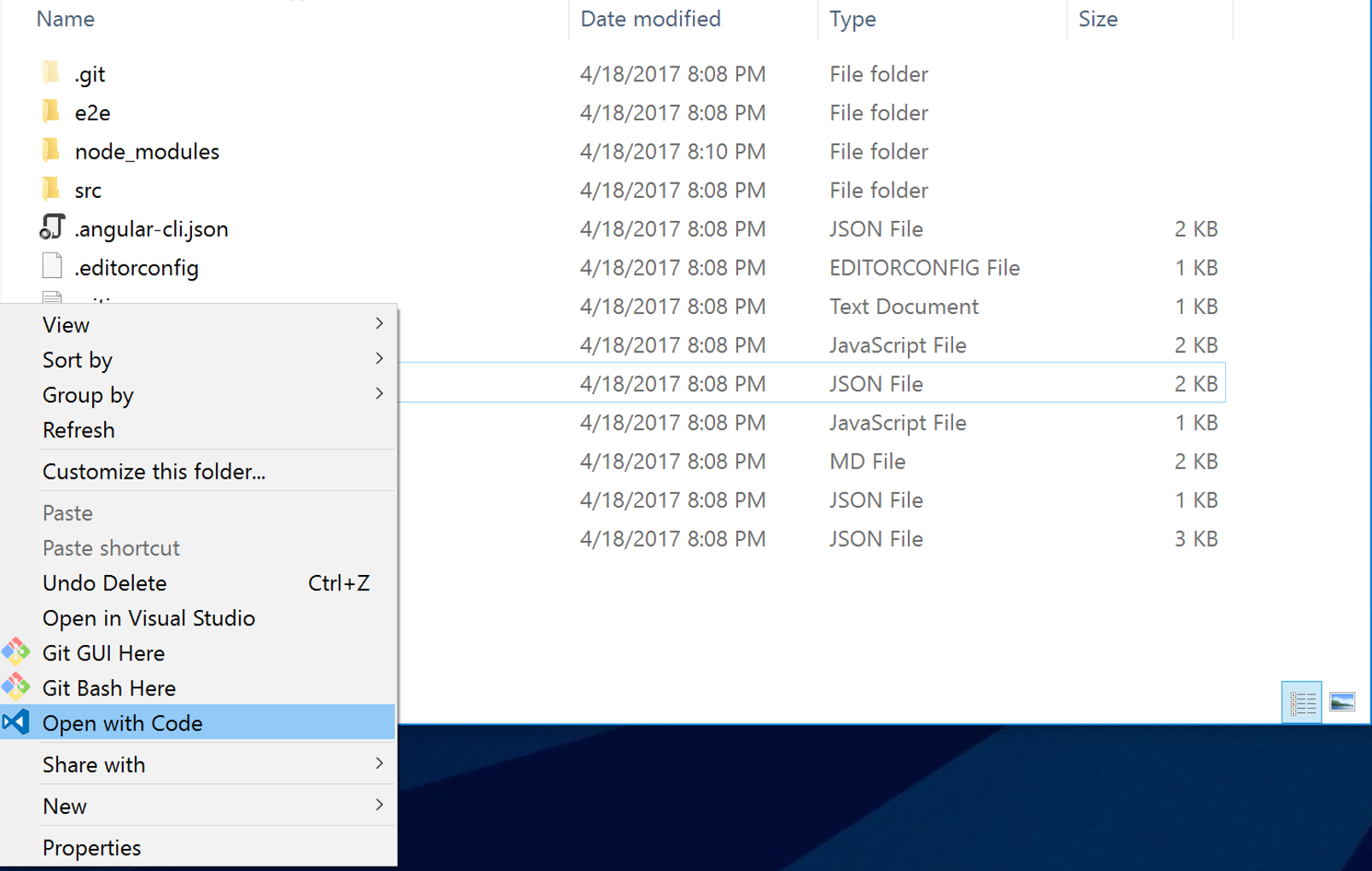
# Lab 1 – Setup: Install angular cli by issuing the following command **npm install -g @angular/cli**

**Note:** npm install -g angular/cli will work but this is a deprecated version

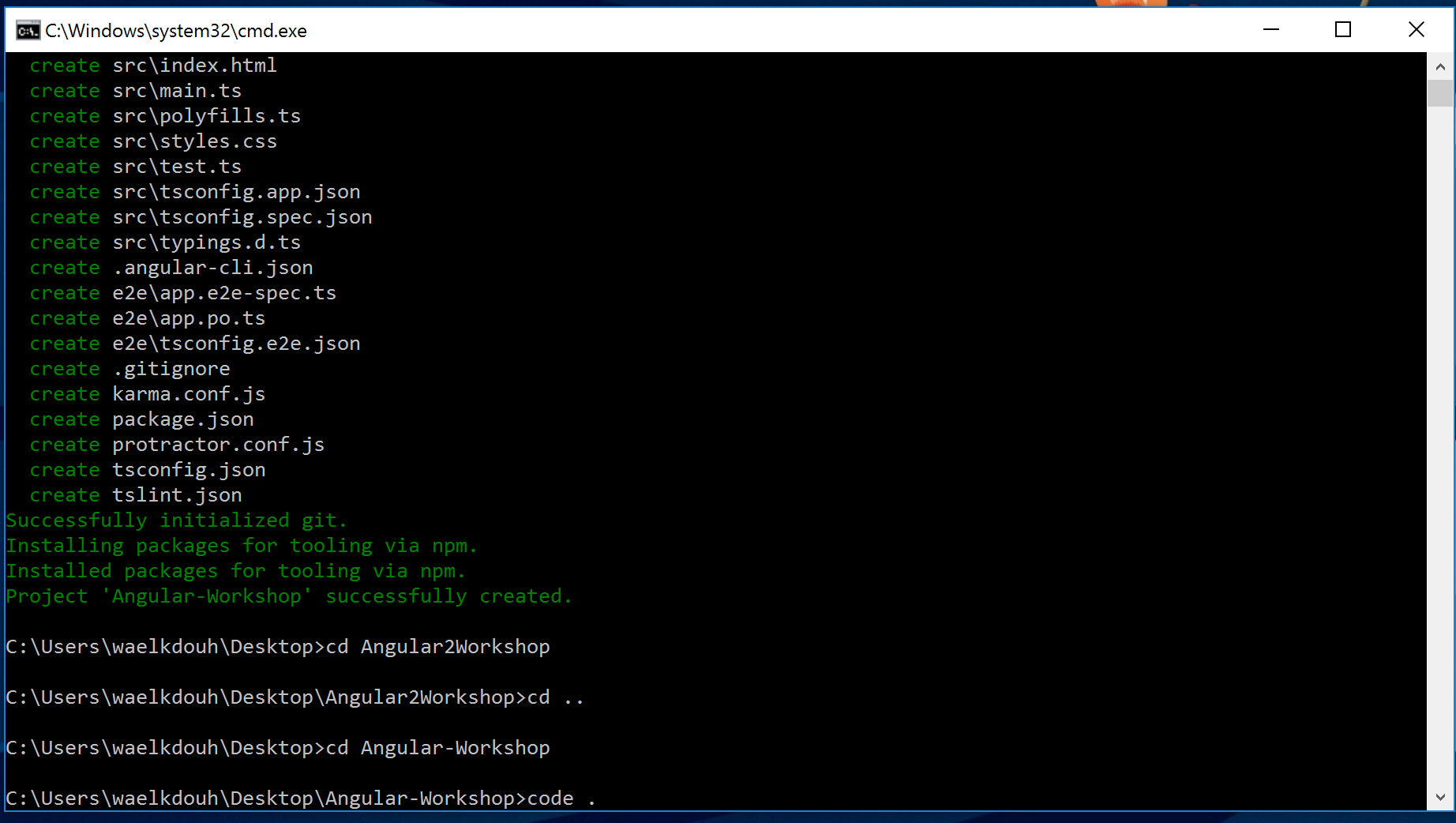
Create a new Angular project using the following command  
**ng new Angular-Workshop**

Navigate to the newly created project   
**cd Angular-Workshop**

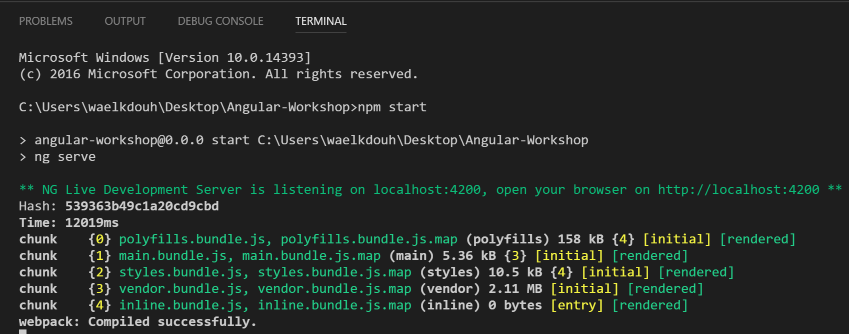
Open the project with vs code. You can either right click inside the folder and open with vs code



or you can type “code .” inside the project directory

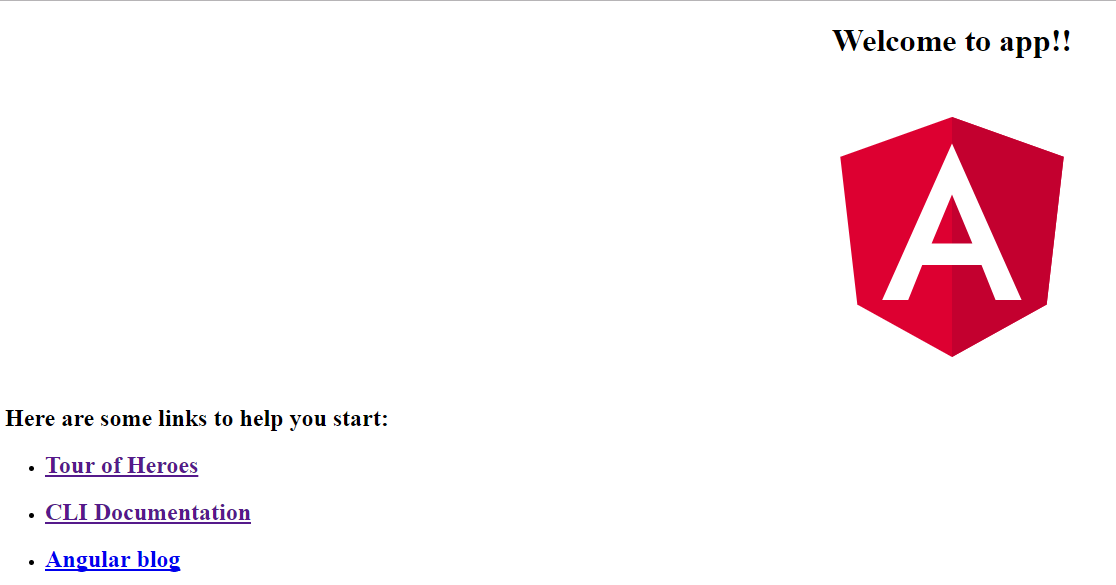


Run the application using the following command inside the vs code command line  
**npm start**



Open a browser and navigate to <http://localhost:4200>

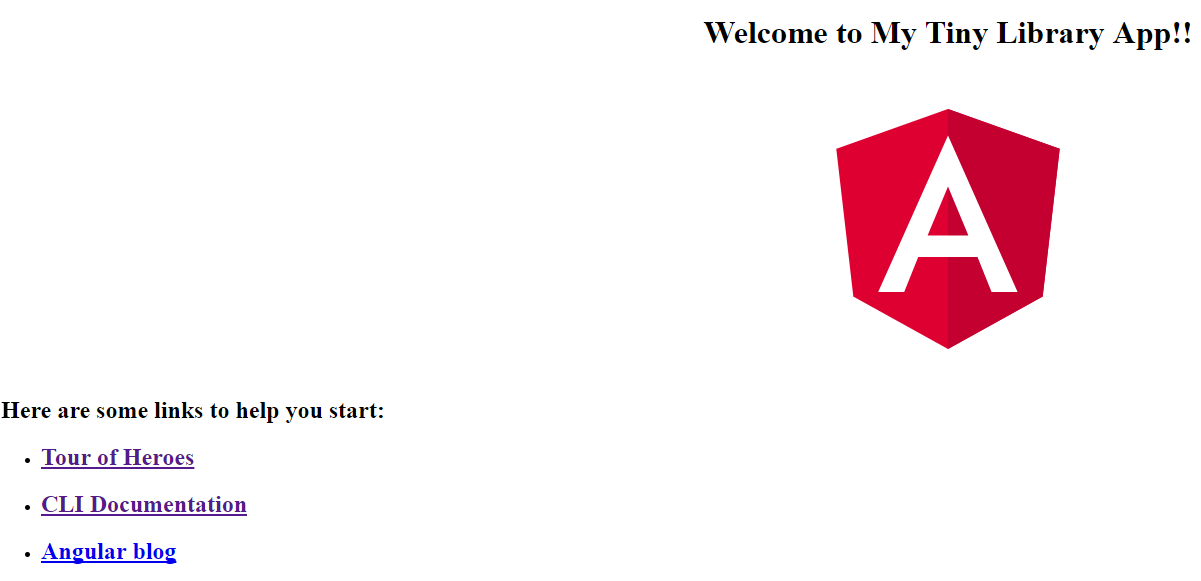
**Tip**: You can click ctrl + hover over the link shown in the command line window to navigate to the url



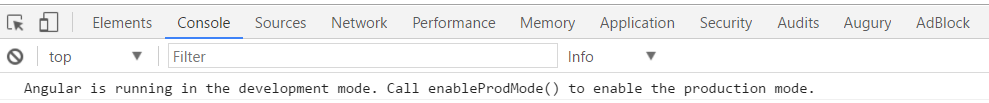
You have now successfully created/run your first Angular application. You can keep the application running as the server will support live page reloading when making any future changes to the code.

Navigate to app.component.ts file which is located under the app folder and modify the title property found under the AppComponent class from “app” to “My Tiny Library App”. Save.

Notice how the browser reflects the changes upon saving the file.

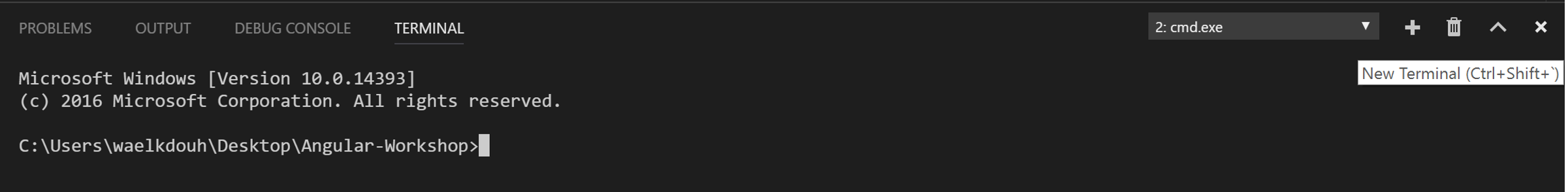


**Tip**: Make sure to always have the F12 tools open under your browser(navigate to the console tab) as Angular provides you with valuable debugging information there. Notice by default the Angular runtime is telling you that you are running in development mode. More on that later.



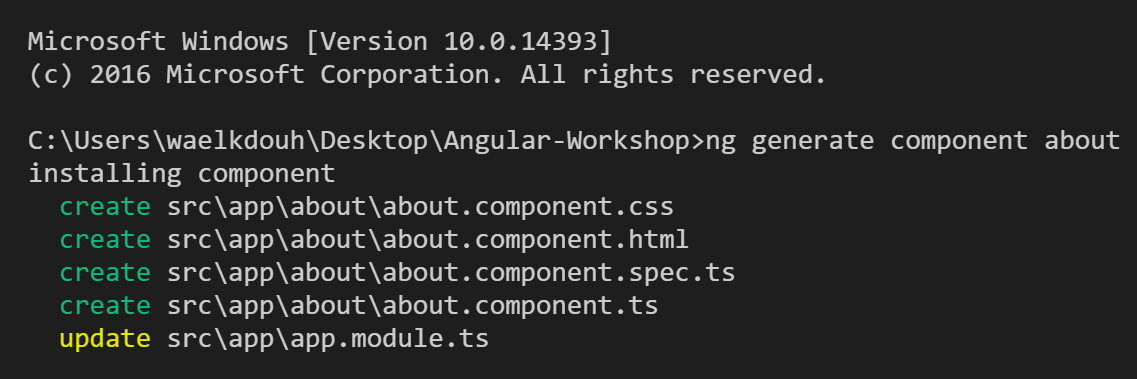
# Lab 2 – Template Containing a Component:

Open a new command window VS Code by clicking on the + icon. This way the first command window will keep the server running for liver reloading to work.



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

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



Notice that the app.module.ts file has been updated to include the newly added component

**Note:** When importing from a path, the ‘**./**’ operator refers to a **directory relative to that of file you are editing**.

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

import { NgModule } from '@angular/core';

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

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

@NgModule({

declarations: [

AppComponent,

AboutComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Modify about.component.ts file to include the following code:

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

@Component({

selector: 'app-about',

templateUrl: './about.component.html',

styleUrls: ['./about.component.css']

})

export class AboutComponent implements OnInit {

pageTitle: string = 'About Me';

constructor() { }

ngOnInit() {

}

}

Modify about.component.html file to include the following code:

<h3>{{pageTitle}}</h3>

<p>Welcome to your personal library </p>

Update the app.component.ts to include the About Component. Save.

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

@Component({

selector: 'app-root',

template: `<h1>{{title}}</h1>

<app-about></app-about>`,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'My Tiny Library App';

}

The browser should now reflect the changes you have made so far.

# Lab 3 – Data Binding:

For this lab we will utilize a framework called [Angular Material](https://material.angular.io/guide/getting-started) 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 inside your project using the following command:

**npm install --save @angular/material @angular/cdk**

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**

Some components (md-slide-toggle, md-slider, mdTooltip) rely on [HammerJS](http://hammerjs.github.io/) for gestures. In order to get the full feature-set of these components, HammerJS must be loaded into the application.

You can add HammerJS to your application via [npm](https://www.npmjs.com/package/hammerjs), a CDN (such as the [Google CDN](https://developers.google.com/speed/libraries/#hammerjs)), or served directly from your app.

To install via npm, use the following command:

**npm install --save hammerjs**

Create a file in the app folder called material-imports.ts:

import { MdCardModule, MdListModule, MdRippleModule, MdIconModule, MdSnackBarModule, MdDialogModule,

MdTabsModule, MdToolbarModule, MdButtonModule, MdInputModule, MdSlideToggle, FocusMonitor } from '@angular/material';

export const MaterialModules = [

MdCardModule,

MdListModule,

MdRippleModule,

MdIconModule,

MdSnackBarModule,

MdDialogModule,

MdTabsModule,

MdToolbarModule,

MdButtonModule,

MdInputModule

];

export const MaterialComponents = [

MdSlideToggle

];

export const MaterialServices = [

FocusMonitor

];

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 'hammerjs';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { FormsModule } from '@angular/forms';

import { MaterialModules, MaterialComponents, MaterialServices } from './material-module';

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

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

import { CollectionComponent } from './collection/collection.component';

@NgModule({

declarations: [

AppComponent,

AboutComponent,

CollectionComponent,

...MaterialComponents

],

imports: [

BrowserModule,

...MaterialModules,

BrowserAnimationsModule,

FormsModule

],

providers: [...MaterialServices],

bootstrap: [AppComponent]

})

export class AppModule { }

Copy the following into your styles.css:

@import '~https://fonts.googleapis.com/icon?family=Material+Icons';

@import '~https://fonts.googleapis.com/css?family=Roboto';

@import '~@angular/material/prebuilt-themes/deeppurple-amber.css';

html {

-webkit-tap-highlight-color: rgba(0, 0, 0, 0);

}

body {

font-family: Roboto, sans-serif;

margin-top: 25px;

font-size: 15px;

line-height: 1.42857143;

color: #2c3e50;

background-color: #ffffff;

}

.fill-remaining-space {

flex: 1 1 auto;

}

.plr-15 {

padding-left: 15px;

padding-right: 15px;

}

.material-icons {

width: 24px;

}

.pull-right {

float: right;

}

.pull-left {

float: left;

}

.text-right {

text-align: right;

}

.text-left {

text-align: left;

}

.full-width {

width: 100%;

}

.container {

margin-right: auto;

margin-left: auto;

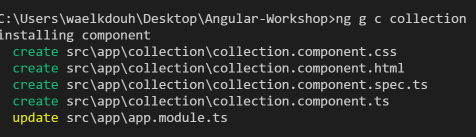
padding-left: 25px;

padding-right: 25px;

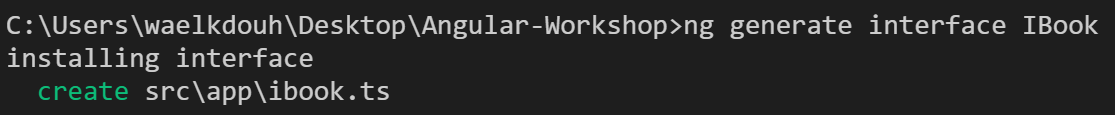
}

Next you want to add a new component called collection by issuing the following command  
**ng generate component collection**

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



Add IBook interface using the following command  
**ng generate interface IBook** or **ng g i IBook**



Modify the IBook interface:

export interface IBook {

id: number;

title: string;

author: string;

isCheckedOut: boolean;

rating: number;

}

Update the collection component (collection.component.ts) to include:

* pageTitle: string, shown in the panel header
* books: Array<IBook>, shown in the table rows
* showOperatingHours: boolean, for message visibility and button text
* a method to toggle whether to show operating hours

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

import { IBook } from '../ibook';

@Component({

selector: 'app-collection',

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public 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

}

];

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor() {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

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

<h3>{{pageTitle}}&nbsp;

<md-slide-toggle class="plr-15" color="primary" [(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library hours

</md-slide-toggle>

</h3>

<div [hidden]="!showOperatingHours">

<md-card>

<md-card-subtitle><strong>Operating Hours</strong></md-card-subtitle>

<md-card-content>{{startTime}} - {{endTime}} (M-F)</md-card-content>

</md-card>

</div>

<div>

<md-list>

<md-list-item \*ngFor="let book of books">

<md-icon md-list-icon>book</md-icon>

<h3 md-line><strong>{{book.title}}</strong></h3>

<p md-line>

<span>{{book.author}}</span>

</p>

<p md-line>

{{book.rating}}

</p>

<p md-line>

<span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>

</p>

</md-list-item>

</md-list>

</div>

Modify the collection.component.css to include the following code:

.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.ts file to include the newly added collection component

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

@Component({

selector: 'app-root',

template: `<h1>{{title}}</h1>

<app-collection></app-collection>`,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'My Tiny Library App';

}

# Lab 4 – Pipes:

Add a new pipe called RatingCategory  
**ng generate pipe RatingCategory** or **ng g p RatingCategory**

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: number): string {

if (value <= 2) {

return 'Poor';

}

if (value <= 4) {

return 'Fine';

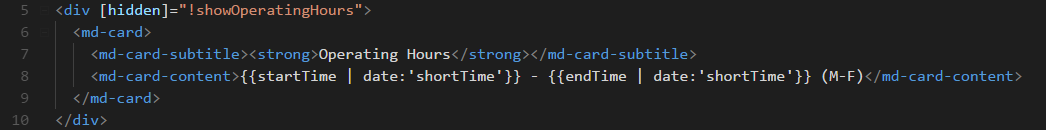
}

return 'Excellent';

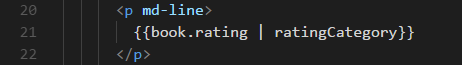
}

}

Update the collection Template to apply the date pipe customized to display ‘shortTime’ to the startTime and endTime.



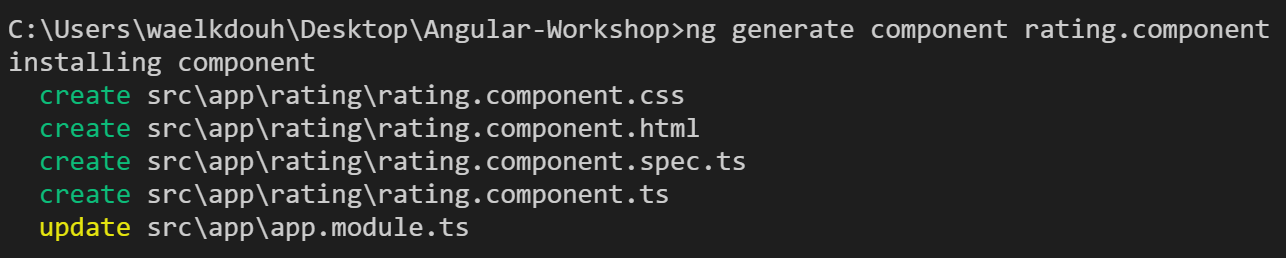
Finally, update collection Template to use Rating Category pipe created above. Save.



# Lab 5 – Communication between Parent and Child Components:

Add a Rating Component by issuing the following command  
**ng generate component rating** or **ng g c rating**

This will generate a new component in a new folder called rating:



Modify the rating.component.ts file to include:

* @Input() for rating, @Input() for the book, @Output() for the click EventEmitter
* ngOnInit() to log a message to the console with the value of the rating
* ngOnChanges() to log a message to the console with the value of the rating
* A method to emit the updated book rating via @Output() EventEmitter when the user changes the rating.

import { Component, OnInit, OnChanges, Input, Output, EventEmitter } from

'@angular/core';

import { IBook } from '../ibook';

@Component({

selector: 'app-rating',

templateUrl: './rating.component.html',

styleUrls: ['./rating.component.css']

})

export class RatingComponent implements OnInit, OnChanges {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

@Input() rating: number;

@Input() book: IBook;

@Output() ratingClicked: EventEmitter<IBook> = new EventEmitter<IBook>();

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor() { }

ngOnInit(): void {

// console.log("ngOnInit called for: " + this.rating.toString());

}

ngOnChanges(): void {

// console.log("The rating was just set to: " + this.rating.toString());

}

click(rating: number): void {

this.book.rating = rating;

this.ratingClicked.emit(this.book);

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

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>

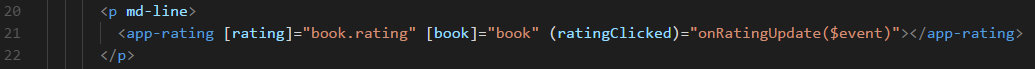
<a (click)="click(3)"><i class="material-icons {{rating >= 3 ? 'active' : ''}}">star\_rate</i></a>

<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. Also add a message box that confirms the rate change:



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 { MdSnackBar } from '@angular/material';

@Component({

selector: 'app-collection',

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public 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

}

];

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor(private \_snackBar: MdSnackBar) {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

}

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');

}

// << ------------- >>

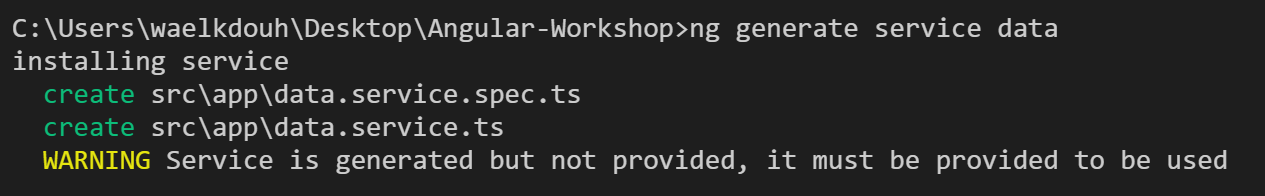
// << METHODS - END >>

// << ------------- >>

}

# Lab 6 – Build a Service:

Generate a new service using the following command  
**ng generate service data** or **ng g s data**



Add one method to Data Service named getBooks:

* This method will return an array of IBooks
* Move the array of books from the collection.component.ts file into getBooks() method as the data doesn’t belong under the component
* Decorate the Data Service with @Injectable and add the required import

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

import { Injectable } from '@angular/core';

import { IBook } from './ibook';

@Injectable()

export class DataService {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

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

}

];

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify app.component.ts to include the data service as a provider as it will be utilized by the collection component (Note: you could have also added the provider directly under the collection controller if you know it will only be utilized there). Save.

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

import { DataService } from './data.service';

@Component({

selector: 'app-root',

template: `<h1>{{title}}</h1>

<app-collection></app-collection>`,

styleUrls: ['./app.component.css'],

providers: [DataService]

})

export class AppComponent {

title = 'My Tiny Library App';

}

Modify the collection.component.ts file:

* Constructor should now expect an instance of DataService via dependency injection
* ngOnInit should call getBooks() method in Data Service and assign the result to its books property
* Add the required import for the Data Service

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

import { IBook } from '../ibook';

import { MdSnackBar } from '@angular/material';

import { DataService } from '../data.service';

@Component({

selector: 'app-collection',

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public books: Array<IBook>;

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor(private \_snackBar: MdSnackBar, private \_dataService: DataService) {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

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');

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

# Lab 7 – Http Service:

Start by creating a folder called api under assets folder. Create a new file called books.json and the following code to it:

[

{

"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",

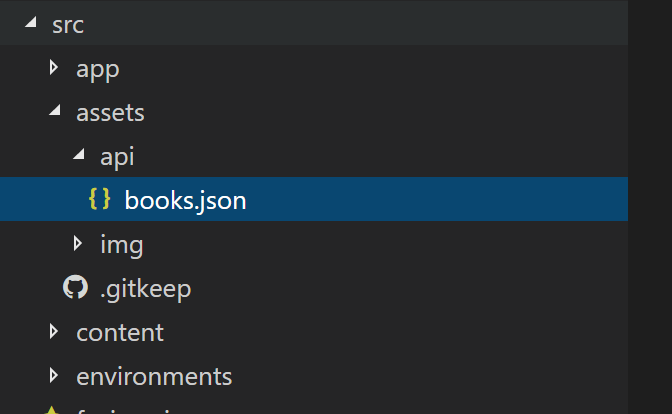
"isCheckedOut": false,

"rating": 5

}

]

As you can see above we moved the data from the data.service.ts file as it doesn’t belong there. Here is the new structure after adding the books.json file:



Create a file called rxjs-operators.ts under the app folder and add the following code:

// Statics

import 'rxjs/add/observable/throw';

// Operators

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

import 'rxjs/add/operator/do';

Modify app.component.ts to have access to those RxJS methods by adding an import at the top of the file:

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

import { DataService } from './data.service';

import './rxjs-operators';

@Component({

selector: 'app-root',

template: `<h1>My Tiny Library App</h1>

<my-collection></my-collection>

`,

providers: [DataService]

})

export class AppComponent {

}

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

* Add a constructor that expects an instance of Http service
* Modify getBooks() to return Observable<IBook[]>
* getBooks() calls \_http.get() and then map() the response
* Add required import for Http and Response from @angular/http
* Add required import for Observable from rxjs/Observable

import { Injectable } from '@angular/core';

import { Http, Response } from '@angular/http';

import { Observable } from 'rxjs/Observable';

import { IBook } from './ibook';

@Injectable()

export class DataService {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

private \_booksUrl = 'assets/api/books.json';

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(private \_http: Http) { }

getBooks(): Observable<IBook[]> {

return this.\_http.get(this.\_booksUrl)

.map((response: Response) => {

let data: IBook[] = <IBook[]>response.json();

localStorage.setItem('books', JSON.stringify(data));

return data;

})

.catch(this.handleError);

}

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);

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

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

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

import { NgModule } from '@angular/core';

import { MaterialModule } 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';

import { CollectionComponent } from './collection/collection.component';

import { RatingCategoryPipe } from './rating-category.pipe';

import { RatingComponent } from './rating/rating.component';

import { HttpModule } from "@angular/http";

@NgModule({

declarations: [

AppComponent,

AboutComponent,

CollectionComponent,

RatingCategoryPipe,

RatingComponent

],

imports: [

BrowserModule,

FormsModule,

MaterialModule,

BrowserAnimationsModule,

HttpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Modify the collection.component.ts file to include a getbooks() method which returns an Observable. Save.

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

import { IBook } from '../ibook';

import { MdSnackBar } from '@angular/material';

import { DataService } from '../data.service';

@Component({

selector: 'app-collection',

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public books: Array<IBook>;

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor(private \_snackBar: MdSnackBar, private \_dataService: DataService) {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

this.getBooks();

}

getBooks(): void {

this.\_dataService.getBooks()

.subscribe(books => this.books = books,

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

}

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');

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

# Lab 8 – Routing:

Add app.routing.ts to the app folder

import { ModuleWithProviders } 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'

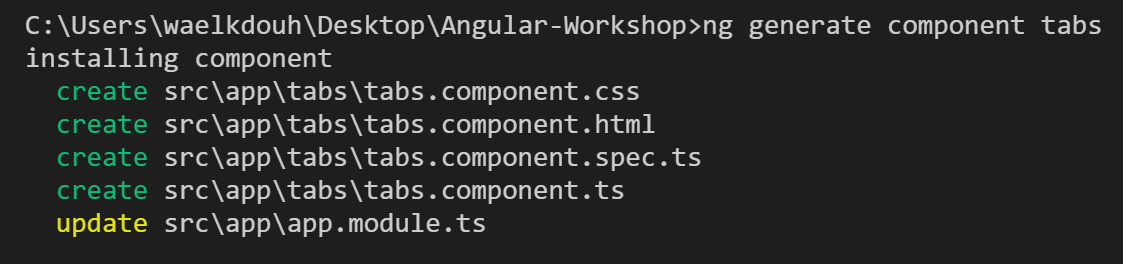
}

];

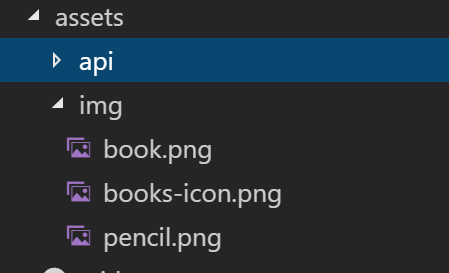
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);

Add a tabs components which will be used to create a navigation menu.

**ng generate component tabs** or **ng g c tabs**



Include the img folder provided to you in the lab assets zip file. The updated project structure should now look like this:



Modify the tabs.component.ts to include the following code:

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

@Component({

selector: 'app-tabs',

templateUrl: './tabs.component.html',

styleUrls: ['./tabs.component.css']

})

export class TabsComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

navLinks: Array<object> = [

{

path: 'about',

label: 'ABOUT ME'

},

{

path: 'collection',

label: 'MY COLLECTION'

}

];

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor() { }

ngOnInit(): void {

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify the tabs.component.html to include the following code:

<nav md-tab-nav-bar>

<a md-tab-link

\*ngFor="let link of navLinks"

[hidden]="!link.path"

[routerLink]="link.path"

routerLinkActive #rla="routerLinkActive"

[active]="rla.isActive">

{{link.label}}

</a>

</nav>

<router-outlet></router-outlet>

Modify app.component.html to include a menu to navigate between the collection page and the about page:

<div class="container">

<md-toolbar color="warn">

<span>{{title}}</span>

<span class="fill-remaining-space"></span>

<img src="assets/img/books-icon.png"

class="center-block"

style="max-height:100px" />

</md-toolbar>

<app-tabs></app-tabs>

</div>

Moidify app.component.ts to point to the updated app.component.html which now included the newly created tabs component:

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

import { DataService } from './data.service';

import './rxjs-operators';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css'],

providers: [DataService]

})

export class AppComponent {

title = 'Material Library App';

}

Modify app.module.ts:

* import { routing } from './app.routing';
* include routing in the imports property of @NgModule

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

import { NgModule } from '@angular/core';

import 'hammerjs';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { FormsModule } from '@angular/forms';

import { MaterialModules, MaterialComponents, MaterialServices } from './material-imports';

import { HttpModule } from '@angular/http';

import { routing } from './app.routing';

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

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

import { CollectionComponent } from './collection/collection.component';

import { RatingCategoryPipe } from './rating-category.pipe';

import { RatingComponent } from './rating/rating.component';

import { TabsComponent } from './tabs/tabs.component';

@NgModule({

declarations: [

AppComponent,

AboutComponent,

CollectionComponent,

...MaterialComponents,

RatingCategoryPipe,

RatingComponent,

TabsComponent

],

imports: [

BrowserModule,

...MaterialModules,

BrowserAnimationsModule,

FormsModule,

HttpModule,

routing

],

providers: [...MaterialServices],

bootstrap: [AppComponent]

})

export class AppModule { }

Remove the selector from the about.component.ts and collection.component.ts files as the collection page will now be navigated to using the tabs instead on including it as part of the app.component.ts file. Save.

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

In this lab we will show the details of a book on a separate page. We will demonstrate that using two different methods. The first is using a modal box and the second is navigating to a totally new page (in reality you would choose one or the other, but we are including both to demonstrate passing parameters as well activating a route with code).

Add a book-detail component as follows:   
**ng generate component book-detail** or **ng g c book-detail**

Modify data.service.ts to include a getBook() method which would allow fetching a single book. Also we will start preparing our application to persisting changes using HTML5 local storage (in reality you would persist to the database). Here is the updated data.service.ts file:

import { Injectable } from '@angular/core';

import { Http, Response } from '@angular/http';

import { Observable } from 'rxjs/Observable';

import { IBook } from './ibook';

@Injectable()

export class DataService {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

private \_booksUrl = 'assets/api/books.json';

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(private \_http: Http) { }

getBooks(): Observable<IBook[]> {

let localBooks = localStorage.getItem('books');

if (localBooks) {

return Observable.create(observer => {

observer.next(JSON.parse(localBooks));

});

}

return this.\_http.get(this.\_booksUrl)

.map((response: Response) => {

let data: IBook[] = <IBook[]>response.json();

localStorage.setItem('books', JSON.stringify(data));

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);

}

updateBook(book: IBook): Observable<IBook[]> {

const local: string = localStorage.getItem('books');

if (!local) return Observable.throw('Local storage not found.');

let localBooks: IBook[] = JSON.parse(local);

localBooks = localBooks.map(b => {

if (b.id === book.id) {

return Object.assign(b, book);

}

return b;

});

localStorage.setItem('books', JSON.stringify(localBooks));

return Observable.create(observer => {

observer.next(localBooks);

});

}

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);

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify book-detail.component.ts file

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

import { Router, ActivatedRoute } from '@angular/router';

import { Subscription } from 'rxjs/Subscription';

import { IBook } from '../ibook';

import { DataService } from '../data.service';

import { MdSnackBar } from '@angular/material';

@Component({

templateUrl: './book-detail.component.html',

styleUrls: ['./book-detail.component.css'],

providers: [DataService]

})

export class BookDetailComponent implements OnInit, OnDestroy {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

bookId: number;

book: IBook;

sub: Subscription;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(

private \_route: ActivatedRoute,

private \_router: Router,

private \_dataService: DataService,

private \_snackBar: MdSnackBar) {}

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'));

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify book-detail.component.html:

<div \*ngIf="book">

<md-card>

<md-card-header>

<md-card-title><h4>{{book.title}}</h4></md-card-title>

<md-card-subtitle>{{book.author}}</md-card-subtitle>

<img md-card-avatar src="assets/img/book.png" />

</md-card-header>

<md-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>

<app-rating [rating]="book.rating" [book]="book" (ratingClicked)="onRatingUpdate($event)"></app-rating>

</div>

</md-card-content>

<md-card-actions>

<div class="text-right">

<button md-button md-dialog-close \*ngIf="bookId"><i class="material-icons">close</i>CLOSE</button>

<button md-button (click)="return()" \*ngIf="!bookId"><i class="material-icons">keyboard\_arrow\_left</i>RETURN</button>

</div>

</md-card-actions>

</md-card>

</div>

Modify book-detail.component.css

.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;

}

Modify the collection.component.ts to enable users to navigate to the book details page. Also modify the onRatingUpdate method to utilize the newly updated data service which is now persisting on the local storage:

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

import { IBook } from '../ibook';

import { MdSnackBar, MdDialog } from '@angular/material';

import { DataService } from '../data.service';

import { BookDetailComponent } from '../book-detail/book-detail.component';

import { Router } from '@angular/router';

@Component({

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public books: Array<IBook>;

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor(private \_snackBar: MdSnackBar, private \_dataService: DataService,

private \_dialog: MdDialog, private \_router: Router) {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

this.getBooks();

}

getBooks(): void {

this.\_dataService.getBooks()

.subscribe(books => this.books = books,

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

}

updateBook(book: IBook): void {

this.\_dataService.updateBook(book)

.subscribe(books => {

this.books = books;

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]);

}

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");

}

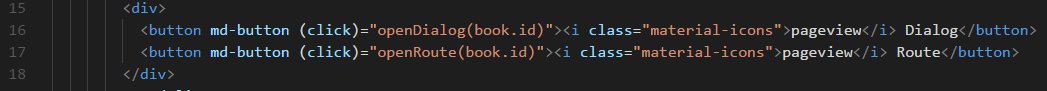
// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify the collection.component.html to enable users to navigate to the book details page:



Modify the app.routing.ts to include the individual book-detail path. Save.

Note: At this point modifying the book rating should now persist across browser refreshes.

import { ModuleWithProviders } 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'

}

];

export const routing: ModuleWithProviders = RouterModule.forRoot(routes);

# Lab 10 – Protecting Routes with Guards:

Add a book guard service using the following command:  
**ng generate service BookGuard** or **ng g s BookGuard**

Modify book-guard.service.ts

import { Injectable } from '@angular/core';

import { ActivatedRouteSnapshot, CanActivate, Router } from '@angular/router';

@Injectable()

export class BookGuardService implements CanActivate {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(private \_router: Router) {

}

canActivate(route: ActivatedRouteSnapshot): boolean {

let id = +route.url[1].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;

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify app.routing.ts to to apply the guard to the book detail route:

import { ModuleWithProviders } 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 './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'

}

];

export const routing: ModuleWithProviders = RouterModule.forRoot(routes);

Modify app.module.ts to include the BookDetailGuard provider. Save.

Note: If you attempt to navigate to http://localhost:4200/collection/-1 you will be rerouted to the main collection page.

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

import { NgModule } from '@angular/core';

import 'hammerjs';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { FormsModule } from '@angular/forms';

import { MaterialModules, MaterialComponents, MaterialServices } from './material-imports';

import { HttpModule } from '@angular/http';

import { routing } from './app.routing';

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

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

import { CollectionComponent } from './collection/collection.component';

import { RatingCategoryPipe } from './rating-category.pipe';

import { RatingComponent } from './rating/rating.component';

import { TabsComponent } from './tabs/tabs.component';

import { BookDetailComponent } from './book-detail/book-detail.component';

import { BookGuardService } from './book-guard.service';

@NgModule({

declarations: [

AppComponent,

AboutComponent,

CollectionComponent,

...MaterialComponents,

RatingCategoryPipe,

RatingComponent,

TabsComponent,

BookDetailComponent

],

imports: [

BrowserModule,

...MaterialModules,

BrowserAnimationsModule,

FormsModule,

HttpModule,

routing

],

providers: [...MaterialServices,

BookGuardService],

bootstrap: [AppComponent]

})

export class AppModule { }

# Lab 11a [Required] – Adding Forms (Template-Driven):

Add a new-book component using the following command:  
**ng generate component NewBook** or **ng g c NewBook**

Modify the new-book.component.ts file:

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

import { IBook } from '../ibook';

import { MdDialogRef } from '@angular/material';

@Component({

selector: 'app-new-book',

templateUrl: './new-book.component.html',

styleUrls: ['./new-book.component.css']

})

export class NewBookComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

book: IBook;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(private \_dialogRef: MdDialogRef<NewBookComponent>) { }

ngOnInit() {

this.book = {

id: 0,

title: '',

author: '',

isCheckedOut: false,

rating: 0

};

}

cancel(): void {

this.\_dialogRef.close();

}

save(): void {

this.\_dialogRef.close(this.book);

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

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">

<md-card>

<md-card-header>

<md-card-title><h4>Add New Book</h4></md-card-title>

<img md-card-avatar src="assets/img/pencil.png" />

</md-card-header>

<md-card-content>

<md-input-container>

<input mdInput placeholder="Book Title" [(ngModel)]="book.title" name="title" required />

</md-input-container>

<md-input-container>

<input mdInput placeholder="Author" [(ngModel)]="book.author" name="author" required />

</md-input-container>

<p class="rating-label">Rating</p>

<app-rating [rating]="book.rating" [book]="book"></app-rating>

<md-slide-toggle color="primary" [(ngModel)]="book.isCheckedOut" name="checkedOut">Checked out?</md-slide-toggle>

</md-card-content>

<md-card-actions>

<div class="text-right">

<button type="submit " md-button (click)="save()" color="warn" [disabled]="newBookForm.form.invalid"><i class="material-icons">save</i>SAVE</button>

<button type="reset" md-button (click)="cancel()"><i class="material-icons">cancel</i>CANCEL</button>

</div>

</md-card-actions>

</md-card>

</form>

Modify app.module.ts to include the NewBookComponent as an entryComponent as shown below:

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

import { NgModule } from '@angular/core';

import { MaterialModule } 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';

import { CollectionComponent } from './collection/collection.component';

import { RatingCategoryPipe } from './rating-category.pipe';

import { RatingComponent } from './rating/rating.component';

import { HttpModule } from "@angular/http";

import { routing } from "app/app.routing";

import { TabsComponent } from './tabs/tabs.component';

import { RouterModule } from "@angular/router";

import { BookDetailComponent } from './book-detail/book-detail.component';

import { BookGuardService } from "app/book-guard.service";

import { NewBookComponent } from './new-book/new-book.component';

@NgModule({

declarations: [

AppComponent,

AboutComponent,

CollectionComponent,

RatingCategoryPipe,

RatingComponent,

TabsComponent,

BookDetailComponent,

NewBookComponent

],

imports: [

BrowserModule,

FormsModule,

MaterialModule,

BrowserAnimationsModule,

HttpModule,

RouterModule,

routing

],

entryComponents: [

NewBookComponent

],

providers: [BookGuardService],

bootstrap: [AppComponent]

})

export class AppModule { }

At this point you will need to restart your application, as we have made NewBookComponent something that should be loaded on compile time.

Next, modify collection.component.html to include a button to create a new book:

<h3>{{pageTitle}}&nbsp;<md-slide-toggle class="plr-15" color="primary"

[(ngModel)]="showOperatingHours">{{showOperatingHours ? 'Hide' : 'Show'}} library

hours</md-slide-toggle></h3>

<div [hidden]="!showOperatingHours">

<md-card>

<md-card-subtitle><strong>Operating Hours</strong></md-card-subtitle>

<md-card-content>{{startTime | date:'shortTime'}} - {{endTime | date:'shortTime'}} (M-F)</md-card-content>

</md-card>

</div>

<div>

<md-list>

<md-list-item \*ngFor="let book of books">

<md-icon md-list-icon>book</md-icon>

<h3 md-line><strong>{{book.title}}</strong></h3>

<div>

<button md-button (click)="openDialog(book.id)"><i class="material-icons">pageview</i> Dialog</button>

<button md-button (click)="openRoute(book.id)"><i class="material-icons">pageview</i> Route</button>

</div>

<p md-line>

<span>{{book.author}}</span>

</p>

<p md-line>

<app-rating [rating]="book.rating" [book]="book" (ratingClicked)="onRatingUpdate($event)"></app-rating>

</p>

<p md-line>

<span [class]="book.isCheckedOut ? 'chip chip-danger' : 'chip chip-success'">{{book.isCheckedOut ? 'Checked-Out' : 'Available'}}</span>

</p>

</md-list-item>

</md-list>

<div class="text-right add-btn">

<button md-raised-button color="primary" (click)="addBook()"><i class="material-icons">add\_box</i> ADD BOOK</button>

</div>

</div>

Modify the collection.component.ts to include the addBook() method:

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

import { IBook } from '../ibook';

import { MdSnackBar, MdDialog } from '@angular/material';

import { DataService } from '../data.service';

import { BookDetailComponent } from '../book-detail/book-detail.component';

import { Router } from '@angular/router';

import { NewBookComponent } from '../new-book/new-book.component';

@Component({

templateUrl: './collection.component.html',

styleUrls: ['./collection.component.css']

})

export class CollectionComponent implements OnInit {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

pageTitle: string = 'Books';

public books: Array<IBook>;

startTime: Date;

endTime: Date;

showOperatingHours: boolean = false;

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << ---------------- >>

// << METHODS - START >>

// << ---------------- >>

constructor(private \_snackBar: MdSnackBar, private \_dataService: DataService,

private \_dialog: MdDialog, private \_router: Router) {

this.startTime = new Date();

this.startTime.setHours(10, 0);

this.endTime = new Date();

this.endTime.setHours(15, 0);

}

ngOnInit(): void {

this.getBooks();

}

getBooks(): void {

this.\_dataService.getBooks()

.subscribe(books => this.books = books,

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

}

addBook(): void {

let config = {width: '650px', height: '650x', position: {top: '50px'}, disableClose: true};

let dialogRef = this.\_dialog.open(NewBookComponent, config);

dialogRef.afterClosed().subscribe(newBook => {

if (newBook) {

newBook.id = this.books.length + 1;

this.\_dataService.addBook(newBook)

.subscribe(books => this.books = books,

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

}

});

}

updateBook(book: IBook): void {

this.\_dataService.updateBook(book)

.subscribe(books => {

this.books = books;

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]);

}

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');

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

Modify the data.service.ts file to include the addBook() service. Save.

import { Injectable } from '@angular/core';

import { Http, Response } from '@angular/http';

import { Observable } from 'rxjs/Observable';

import { IBook } from './ibook';

@Injectable()

export class DataService {

// << ------------------ >>

// << PROPERTIES - START >>

// << ------------------ >>

private \_booksUrl = 'assets/api/books.json';

// << ---------------- >>

// << PROPERTIES - END >>

// << ---------------- >>

// << --------------- >>

// << METHODS - START >>

// << --------------- >>

constructor(private \_http: Http) { }

getBooks(): Observable<IBook[]> {

let localBooks = localStorage.getItem('books');

if (localBooks) {

return Observable.create(observer => {

observer.next(JSON.parse(localBooks));

});

}

return this.\_http.get(this.\_booksUrl)

.map((response: Response) => {

let data: IBook[] = <IBook[]>response.json();

localStorage.setItem('books', JSON.stringify(data));

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);

}

addBook(book: IBook): Observable<IBook[]> {

const local:string = localStorage.getItem('books');

if (!local) return Observable.throw('Local storage not found.');

let localBooks:IBook[] = JSON.parse(local);

localBooks.push(book);

localStorage.setItem('books', JSON.stringify(localBooks));

return Observable.create(observer => {

observer.next(localBooks);

});

}

updateBook(book: IBook): Observable<IBook[]> {

const local: string = localStorage.getItem('books');

if (!local) return Observable.throw('Local storage not found.');

let localBooks: IBook[] = JSON.parse(local);

localBooks = localBooks.map(b => {

if (b.id === book.id) {

return Object.assign(b, book);

}

return b;

});

localStorage.setItem('books', JSON.stringify(localBooks));

return Observable.create(observer => {

observer.next(localBooks);

});

}

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);

}

// << ------------- >>

// << METHODS - END >>

// << ------------- >>

}

# Lab 11b [Optional] – Adding Forms (Model-Driven / Reactive)

In app.module.ts, add ReactiveFormsModule to your import statements and also add it to the imports property of your module decorator.

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

imports: [

BrowserModule,

FormsModule,

MaterialModule,

BrowserAnimationsModule,

HttpModule,

RouterModule,

routing,

ReactiveFormsModule

],

Create a new file called ‘new-book-constants.ts’ in the new-book directory.

import { Injectable } from "@angular/core";

@Injectable()

export class NewBookConstants {

titleMinLength:number = 3;

authorMinLength:number = 5;

titleErrorRequired:string = 'The Title field is required.';

titleErrorMinLength: string = 'The Title field must have at least ' + *this*.titleMinLength + ' characters.';

authorErrorRequired:string = 'The Author field is required.';

authorErrorMinLength:string = 'The Author field must have at least ' + *this*.authorMinLength + ' characters.';

authorErrorFullName:string = 'The Author field must have a first and last name.';

ratingErrorRequired:string = 'The Rating input is required.'

}

This contains all of the constant error messages we will display in the Reactive Form when Validators are not satisfied.

Update new-book.component.ts to import FormBuilder, FormGroup, Validators, and FormControl from ‘@angular/forms’. Also, import the NewBookConstants service we defined above. Add it to your providers. This will allow us to access the consolidated constants we have defined in the service class without cluttering this model. Update the constructor to scaffold rForm, our FormGroup variable, which encapsulates the FormControls for each property of our Reactive Form. Add the addPost method to act on the Reactive Form properties once all Validators are satisfied.

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

import { IBook } from '../ibook';

import { MdDialogRef } from '@angular/material';

import { FormBuilder, FormGroup, Validators, FormControl } from '@angular/forms';

import { NewBookConstants } from "./new-book-constants";

@Component({

selector: 'app-new-book',

templateUrl: './new-book.component.html',

styleUrls: ['./new-book.component.css'],

providers: [NewBookConstants]

})

export class NewBookComponent implements OnInit {

book:IBook;

rForm:FormGroup;

constructor(private *\_fb*:FormBuilder, private *\_dialogRef*: MdDialogRef<NewBookComponent>, public *newBookConstants*:NewBookConstants) {

*this*.rForm = \_fb.group({

'title': [null, Validators.compose([Validators.required, Validators.minLength(3)])],

'author': [null, Validators.compose([Validators.required, Validators.minLength(5)])],

'rating': [null, Validators.required],

'isCheckedOut': [null],

'validate' : ''

});

}

ngOnInit() {

*this*.book = {

id: 0,

title: '',

author: '',

isCheckedOut: false,

rating: 0

}

}

cancel(): void {

*this*.rForm.reset();

*this*.\_dialogRef.close();

}

save(): void{

*this*.\_dialogRef.close(*this*.book);

}

onRatingClicked(*book*:IBook):void {

*this*.rForm.controls['rating'].setValue(*this*.book.rating);

}

addPost(*post*) {

*this*.book.title = post.title;

*this*.book.author = post.author;

*this*.book.rating = post.rating;

*this*.book.isCheckedOut = post.isCheckedOut;

}

}

Update new-book.component.html to reflect the changes to the component class:

<form [formGroup]="rForm" (ngSubmit)="addPost(rForm.value)">

<md-card>

<md-card-header>

<md-card-title><h4>Add New Book</h4></md-card-title>

<img md-card-avatar src="assets/img/pencil.png" />

</md-card-header>

*<!-- Shows the reactive form controls in the modal -->*

<div \*ngIf="rForm.value">

<p>Form value: {{ rForm.value | json }}</p>

<p>Form status: {{ rForm.status | json }}</p>

<p>Form pristine: {{ rForm.pristine | json }}</p>

<p>Form untouched: {{ rForm.untouched | json }}</p>

<p>Title: {{ rForm.controls['title']?.errors | json }}</p>

<p>Author: {{ rForm.controls['author'].errors | json }}</p>

<p>Rating: {{ rForm.controls['rating'].errors | json }}</p>

</div>

<md-card-content>

<md-input-container>

<label>Title

<div \*ngIf="!rForm.controls['title'].valid && (rForm.controls['title'].dirty || rForm.controls['title'].touched)">

<span class="chip chip-danger" \*ngIf="rForm.controls['title'].errors.minlength">

{{ newBookConstants.titleErrorMinLength }}

</span>

<span class="chip chip-danger" \*ngIf="rForm.controls['title'].errors?.required">

{{ newBookConstants.titleErrorRequired }}

</span>

</div>

<input mdInput formControlName="title" class="form-control" required/>

</label>

</md-input-container>

<md-input-container>

<label>Author

<div \*ngIf="!rForm.controls['author'].valid && (rForm.controls['author'].dirty || rForm.controls['author'].touched)">

<span class="chip chip-danger" \*ngIf="rForm.controls['author'].errors.minlength">

{{ newBookConstants.authorErrorMinLength }}

</span>

<span class="chip chip-danger" \*ngIf="rForm.controls['author'].errors?.required">

{{ newBookConstants.authorErrorRequired }}

</span>

</div>

<span class="chip chip-danger" \*ngIf="rForm.controls['author'].valid && rForm.controls['author'].value.indexOf(' ') == -1">

{{ newBookConstants.authorErrorFullName }}

</span>

<input mdInput formControlName="author"/>

</label>

</md-input-container>

<p class="rating-label">Rating </p>

<div \*ngIf="!rForm.controls['title'].valid && (rForm.controls['title'].dirty || rForm.controls['title'].touched)">

<span class="chip chip-danger" \*ngIf="rForm.controls['title'].errors?.required">

{{ newBookConstants.ratingErrorRequired }}

</span>

</div>

<my-rating [book]="book" (ratingClicked)="onRatingClicked($event)">

</my-rating>

<md-slide-toggle color="primary" formControlName="isCheckedOut" name="checkedOut">Checked out?</md-slide-toggle>

</md-card-content>

<md-card-actions>

<div class="text-right">

<button type="submit " md-button (click)="save()" color="warn" [disabled]="!rForm.valid"><i class="material-icons">save</i>SAVE</button>

<button type="reset" md-button (click)="cancel()"><i class="material-icons">cancel</i>CANCEL</button>

</div>

</md-card-actions>

</md-card>

</form>