SOURCE : <https://www.djamware.com/post/5a0673c880aca7739224ee21/mean-stack-angular-5-crud-web-application-example>

By: by Didin J

The following tools, frameworks, and modules are required for this tutorial:

- Node.js (recommended version)  
- Angular CLI 1.5  
- Angular 5  
- MongoDB  
- Express.js  
- Mongoose.js  
- IDE or Text Editor

We assume that you already installed Node.js and runnable in the Terminal (Linux/Mac) or Node.js command line (Windows). Also, you have installed MongoDB and run Mongo daemon on your machine.

**1. Update Angular CLI and Create Angular 5 Application**

First, we have to update the Angular CLI to the latest version (1.5 when this tutorial was written). Open the terminal or Node command line then go to your projects folder. Type this command for updating Angular CLI.

sudo npm install -g @angular/cli

You can exclude `sudo` when you update or install Angular CLI on Windows/Node command line. Now, type this command to create new Angular 2 application.

ng new mean-angular5

Go to the newly created application folder.

cd ./mean-angular5

Run the Angular 2 application by typing this command.

ng serve

You see the compilation process faster than the previous Angular version.

\*\* NG Live Development Server is listening on localhost:4200, open your browser on <http://localhost:4200/> \*\*

Date: 2017-11-10T23:12:58.186Z                                                - Hash: a8de16d629b34a42bbda

Time: 9459ms

chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]

chunk {main} main.bundle.js (main) 20.6 kB [initial] [rendered]

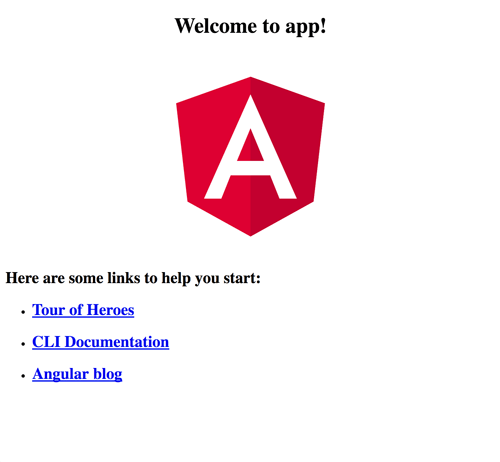
chunk {polyfills} polyfills.bundle.js (polyfills) 553 kB [initial] [rendered]

chunk {styles} styles.bundle.js (styles) 33.8 kB [initial] [rendered]

chunk {vendor} vendor.bundle.js (vendor) 7.03 MB [initial] [rendered]

webpack: Compiled successfully.

Now, open the browser then go to `[http://localhost:4200](http://localhost:4200/)` you should see this page.



**2. Replace Web Server with Express.js**

Close the running Angular app first by press `ctrl+c` then type this command for adding Express.js modules and it dependencies.

npm install --save express body-parser morgan body-parser serve-favicon

Then, add bin folder and www file inside bin folder.

mkdir bin

touch bin/www

Open and edit www file then add this lines of codes.

#!/usr/bin/env node

/\*\*

 \* Module dependencies.

 \*/

var app = require('../app');

var debug = require('debug')('mean-app:server');

var http = require('http');

/\*\*

 \* Get port from environment and store in Express.

 \*/

var port = normalizePort(process.env.PORT || '3000');

app.set('port', port);

/\*\*

 \* Create HTTP server.

 \*/

var server = http.createServer(app);

/\*\*

 \* Listen on provided port, on all network interfaces.

 \*/

server.listen(port);

server.on('error', onError);

server.on('listening', onListening);

/\*\*

 \* Normalize a port into a number, string, or false.

 \*/

function normalizePort(val) {

  var port = parseInt(val, 10);

  if (isNaN(port)) {

    // named pipe

    return val;

  }

  if (port >= 0) {

    // port number

    return port;

  }

  return false;

}

/\*\*

 \* Event listener for HTTP server "error" event.

 \*/

function onError(error) {

  if (error.syscall !== 'listen') {

    throw error;

  }

  var bind = typeof port === 'string'

    ? 'Pipe ' + port

    : 'Port ' + port;

  // handle specific listen errors with friendly messages

  switch (error.code) {

    case 'EACCES':

      console.error(bind + ' requires elevated privileges');

      process.exit(1);

      break;

    case 'EADDRINUSE':

      console.error(bind + ' is already in use');

      process.exit(1);

      break;

    default:

      throw error;

  }

}

/\*\*

 \* Event listener for HTTP server "listening" event.

 \*/

function onListening() {

  var addr = server.address();

  var bind = typeof addr === 'string'

    ? 'pipe ' + addr

    : 'port ' + addr.port;

  debug('Listening on ' + bind);

}

To make the server run from bin/www, open and edit "package.json" then replace "start" value.

"scripts": {

  "ng": "ng",

  "start": "ng build && node ./bin/www",

  "build": "ng build",

  "test": "ng test",

  "lint": "ng lint",

  "e2e": "ng e2e"

},

Now, create app.js in the root of project folder.

touch app.js

Open and edit app.js then add all this lines of codes.

var express = require('express');

var path = require('path');

var favicon = require('serve-favicon');

var logger = require('morgan');

var bodyParser = require('body-parser');

var book = require('./routes/book');

var app = express();

app.use(logger('dev'));

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({'extended':'false'}));

app.use(express.static(path.join(\_\_dirname, 'dist')));

app.use('/books', express.static(path.join(\_\_dirname, 'dist')));

app.use('/book', book);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

  var err = new Error('Not Found');

  err.status = 404;

  next(err);

});

// error handler

app.use(function(err, req, res, next) {

  // set locals, only providing error in development

  res.locals.message = err.message;

  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page

  res.status(err.status || 500);

  res.render('error');

});

module.exports = app;

Next, create routes folder then create routes file for the book.

mkdir routes

touch routes/book.js

Open and edit `routes/book.js` file then add this lines of codes.

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

  res.send('Express RESTful API');

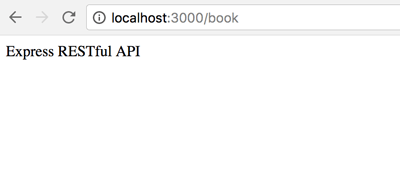
});

module.exports = router;

Now, run the server using this command.

npm start

You will see the previous Angular landing page when you point your browser to `[http://localhost:3000](http://localhost:3000/)`. When you change the address to `<http://localhost:3000/book>` you will see this page.



Now, we have RESTful API with the compiled Angular 5 front end.

**3. Install and Configure Mongoose.js**

We need to access data from MongoDB. For that we will install and configure Mongoose.js. On the terminal type this command after stopping the running Express server.

npm install --save mongoose bluebird

Open and edit `app.js` then add this lines after another variable line.

var mongoose = require('mongoose');

mongoose.Promise = require('bluebird');

mongoose.connect('mongodb://localhost/mean-angular5', { useMongoClient: true, promiseLibrary: require('bluebird') })

  .then(() =>  console.log('connection succesful'))

  .catch((err) => console.error(err));

Now, run MongoDB server on different terminal tab or command line or run from the service.

mongod

Next, you can test the connection to MongoDB run again the Node application and you will see this message on the terminal.

connection succesful

If you are still using built-in Mongoose Promise library, you will get this deprecated warning on the terminal.

(node:42758) DeprecationWarning: Mongoose: mpromise (mongoose's default promise library) is deprecated, plug in your own promise library instead: <http://mongoosejs.com/docs/promises.html>

That's the reason why we added `bluebird` modules and register it as Mongoose Promise library.

**4. Create Mongoose.js Model**

Add a models folder on the root of project folder for hold Mongoose.js model files.

mkdir models

Create new Javascript file that uses for Mongoose.js model. We will create a model of Book collection.

touch models/Book.js

Now, open and edit that file and add Mongoose require.

var mongoose = require('mongoose');

Then add model fields like this.

var BookSchema = new mongoose.Schema({

  isbn: String,

  title: String,

  author: String,

  description: String,

  published\_year: String,

  publisher: String,

  updated\_date: { type: Date, default: Date.now },

});

That Schema will mapping to MongoDB collections called book. If you want to know more about Mongoose Schema Datatypes you can find it [here](http://mongoosejs.com/docs/schematypes.html). Next, export that schema.

module.exports = mongoose.model('Book', BookSchema);

**5. Create Routes for Accessing Book Data via Restful API**

Open and edit again "routes/book.js” then replace all codes with this.

var express = require('express');

var router = express.Router();

var mongoose = require('mongoose');

var Book = require('../models/Book.js');

/\* GET ALL BOOKS \*/

router.get('/', function(req, res, next) {

  Book.find(function (err, products) {

    if (err) return next(err);

    res.json(products);

  });

});

/\* GET SINGLE BOOK BY ID \*/

router.get('/:id', function(req, res, next) {

  Book.findById(req.params.id, function (err, post) {

    if (err) return next(err);

    res.json(post);

  });

});

/\* SAVE BOOK \*/

router.post('/', function(req, res, next) {

  Book.create(req.body, function (err, post) {

    if (err) return next(err);

    res.json(post);

  });

});

/\* UPDATE BOOK \*/

router.put('/:id', function(req, res, next) {

  Book.findByIdAndUpdate(req.params.id, req.body, function (err, post) {

    if (err) return next(err);

    res.json(post);

  });

});

/\* DELETE BOOK \*/

router.delete('/:id', function(req, res, next) {

  Book.findByIdAndRemove(req.params.id, req.body, function (err, post) {

    if (err) return next(err);

    res.json(post);

  });

});

module.exports = router;

Run again the Express server then open the other terminal or command line to test the Restful API by type this command.

curl -i -H "Accept: application/json" localhost:3000/book

If that command return response like below then REST API is ready to go.

HTTP/1.1 200 OK

X-Powered-By: Express

Content-Type: application/json; charset=utf-8

Content-Length: 2

ETag: W/"2-l9Fw4VUO7kr8CvBlt4zaMCqXZ0w"

Date: Fri, 10 Nov 2017 23:53:52 GMT

Connection: keep-alive

Now, let's populate Book collection with initial data that sent from RESTful API. Run this command to populate it.

curl -i -X POST -H "Content-Type: application/json" -d '{ "isbn":"123442123, 97885654453443","title":"Learn how to build modern web application with MEAN stack","author": "Didin J.","description":"The comprehensive step by step tutorial on how to build MEAN (MongoDB, Express.js, Angular 5 and Node.js) stack web application from scratch","published\_year":"2017","publisher":"Djamware.com" }' localhost:3000/book

You will see this response to the terminal if success.

HTTP/1.1 200 OK

X-Powered-By: Express

Content-Type: application/json; charset=utf-8

Content-Length: 415

ETag: W/"19f-SB/dEQyffaTjobOBJbvmwCn7WJA"

Date: Fri, 10 Nov 2017 23:58:11 GMT

Connection: keep-alive

{"\_\_v":0,"isbn":"123442123, 97885654453443","title":"Learn how to build modern web application with MEAN stack","author":"Didin J.","description":"The comprehensive step by step tutorial on how to build MEAN (MongoDB, Express.js, Angular 5 and Node.js) stack web application from scratch","published\_year":"2017","publisher":"Djamware.com","\_id":"5a063d123cf0792af12ce45d","updated\_date":"2017-11-10T23:58:10.971Z"}MacBook-Pro:mean-angular5

**6. Create Angular 5 Component for Displaying Book List**

To create Angular 5 Component, simply run this command.

ng g component book

That command will generate all required files for build book component and also automatically added book component to app.module.ts.

create src/app/book/book.component.css (0 bytes)

create src/app/book/book.component.html (23 bytes)

create src/app/book/book.component.spec.ts (614 bytes)

create src/app/book/book.component.ts (321 bytes)

update src/app/app.module.ts (390 bytes)

Before add any functionality to the component, we need to add `HttpClientModule` to `app.module.ts`. Open and edit `src/app/app.module.ts` then add this import.

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

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

Add it to `@NgModule` imports after `BrowserModule`.

imports: [

  BrowserModule,

  FormsModule,

  HttpClientModule

],

Now, we will making a request to Book RESTful API using this Angular `HttpClient` module. Open and edit `src/app/book/book.component.ts` then add this import.

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

Inject `HttpClient` to the constructor.

constructor(private http: HttpClient) { }

Add array variable for holding books data before the constructor.

books: any;

Add a few lines of codes for getting a list of book data from RESTful API inside `ngOnInit` function.

ngOnInit() {

  this.http.get('/book').subscribe(data => {

    this.books = data;

  });

}

Now, we can display the book list on the page. Open and edit `src/app/book/book.component.html` then replace all tags with this lines of HTML tags.

<div class="container">

  <h1>Book List</h1>

  <table class="table">

    <thead>

      <tr>

        <th>Title</th>

        <th>Author</th>

        <th>Action</th>

      </tr>

    </thead>

    <tbody>

      <tr \*ngFor="let book of books">

        <td>{{ book.title }}</td>

        <td>{{ book.author }}</td>

        <td>Show Detail</td>

      </tr>

    </tbody>

  </table>

</div>

That HTML tags include style class from Bootstrap CSS library. Open and edit `src/index.html` then add the Bootstrap CSS and JS library.

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>MeanAngular5</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <!-- Latest compiled and minified CSS -->

  <link rel="stylesheet" href="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css>" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

  <!-- Optional theme -->

  <link rel="stylesheet" href="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css>" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

</head>

<body>

  <app-root></app-root>

  <!-- Latest compiled and minified JavaScript -->

  <script src="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js>" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

</body>

</html>

**7. Create Angular 5 Routes to Book Component**

To use book component as default landing page, open and edit `src/app/app.module.ts` the add import for Routing.

import { RouterModule, Routes } from '@angular/router';

Create constant router for routing to book component before `@NgModule`.

const appRoutes: Routes = [

  {

    path: 'books',

    component: BookComponent,

    data: { title: 'Book List' }

  },

  { path: '',

    redirectTo: '/books',

    pathMatch: 'full'

  }

];

In @NgModule imports, section adds ROUTES constant, so imports section will be like this.

imports: [

  BrowserModule,

  HttpClientModule,

  RouterModule.forRoot(

    appRoutes,

    { enableTracing: true } // <-- debugging purposes only

  )

],

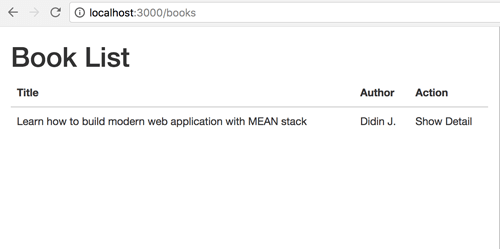
To activate that routes in Angular 5, open and edit `src/app/app.component.html` then replace all codes with this.

<router-outlet></router-outlet>

Now, we have to test our MEAN app with only list page. Build then run the application.

npm start

You should see this page when pointing to `[http://localhost:3000](http://localhost:3000/)` or `<http://localhost:3000/books>`.



**8. Create Angular 5 Component for Displaying Book Detail**

Same as previous section, type this command to generate new component.

ng g component book-detail

Add router to `src/app/app.module.ts` routes constant.

const appRoutes: Routes = [

  {

    path: 'books',

    component: BookComponent,

    data: { title: 'Book List' }

  },

  {

    path: 'book-details/:id',

    component: BookDetailComponent,

    data: { title: 'Book Details' }

  },

  { path: '',

    redirectTo: '/books',

    pathMatch: 'full'

  }

];

Open and edit `src/app/book-detail/book-detail.component.ts`. Replace all codes with this.

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

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

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

@Component({

  selector: 'app-book-detail',

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

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

  encapsulation: ViewEncapsulation.None

})

export class BookDetailComponent implements OnInit {

  book = {};

  constructor(private route: ActivatedRoute, private http: HttpClient) { }

  ngOnInit() {

    this.getBookDetail(this.route.snapshot.params['id']);

  }

  getBookDetail(id) {

    this.http.get('/book/'+id).subscribe(data => {

      this.book = data;

    });

  }

}

Open and edit `src/app/book-detail/book-detail.component.html`. Replace all codes with this.

<div class="container">

  <h1>{{ book.title }}</h1>

  <dl class="list">

    <dt>ISBN</dt>

    <dd>{{ book.isbn }}</dd>

    <dt>Author</dt>

    <dd>{{ book.author }}</dd>

    <dt>Publisher</dt>

    <dd>{{ book.publisher }}</dd>

    <dt>Price</dt>

    <dd>{{ book.price }}</dd>

    <dt>Update Date</dt>

    <dd>{{ book.updated\_at }}</dd>

  </dl>

</div>

**9. Create Angular 5 Component for Add New Book**

To create a component to add new Book, type this command as usual.

ng g component book-create

Add router to `src/app/app.module.ts` routes constant.

const appRoutes: Routes = [

  {

    path: 'books',

    component: BookComponent,

    data: { title: 'Book List' }

  },

  {

    path: 'book-details/:id',

    component: BookDetailComponent,

    data: { title: 'Book Details' }

  },

  {

    path: 'book-create',

    component: BookCreateComponent,

    data: { title: 'Create Book' }

  },

  { path: '',

    redirectTo: '/books',

    pathMatch: 'full'

  }

];

Add 'book-create' link on `src/app/book/book.component.html`.

<h1>Book List

  <a [routerLink]="['/book-create']" class="btn btn-default btn-lg">

    <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>

  </a>

</h1>

Now, open and edit `src/app/book/book-create.component.ts` then replace all with this codes.

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

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

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

@Component({

  selector: 'app-book-create',

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

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

  encapsulation: ViewEncapsulation.None

})

export class BookCreateComponent implements OnInit {

  book = {};

  constructor(private http: HttpClient, private router: Router) { }

  ngOnInit() {

  }

  saveBook() {

    this.http.post('/book', this.book)

      .subscribe(res => {

          let id = res['\_id'];

          this.router.navigate(['/book-details', id]);

        }, (err) => {

          console.log(err);

        }

      );

  }

}

Modify `src/app/book-create/book-create.component.html`, replace all with this HTML tags.

<div class="container">

  <h1>Add New Book</h1>

  <div class="row">

    <div class="col-md-6">

      <form (ngSubmit)="saveBook()" #bookForm="ngForm">

        <div class="form-group">

          <label for="name">ISBN</label>

          <input type="text" class="form-control" [(ngModel)]="book.isbn" name="isbn" required>

        </div>

        <div class="form-group">

          <label for="name">Title</label>

          <input type="text" class="form-control" [(ngModel)]="book.title" name="title" required>

        </div>

        <div class="form-group">

          <label for="name">Author</label>

          <input type="text" class="form-control" [(ngModel)]="book.author" name="author" required>

        </div>

        <div class="form-group">

          <label for="name">Published Year</label>

          <input type="number" class="form-control" [(ngModel)]="book.published\_year" name="published\_year" required>

        </div>

        <div class="form-group">

          <label for="name">Publisher</label>

          <input type="text" class="form-control" [(ngModel)]="book.publisher" name="publisher" required>

        </div>

        <div class="form-group">

          <button type="submit" class="btn btn-success" [disabled]="!bookForm.form.valid">Save</button>

        </div>

      </form>

    </div>

  </div>

</div>

**10. Create Angular 5 Component for Edit Book**

As usual, we will generate component for edit book. Type this command for doing that.

ng g component book-edit

Add route in `src/app/app.module.ts` so, it looks like this.

const appRoutes: Routes = [

  {

    path: 'books',

    component: BookComponent,

    data: { title: 'Book List' }

  },

  {

    path: 'book-details/:id',

    component: BookDetailComponent,

    data: { title: 'Book Details' }

  },

  {

    path: 'book-create',

    component: BookCreateComponent,

    data: { title: 'Create Book' }

  },

  {

    path: 'book-edit/:id',

    component: BookEditComponent,

    data: { title: 'Edit Book' }

  },

  { path: '',

    redirectTo: '/books',

    pathMatch: 'full'

  }

];

Open and edit again `src/app/book-details/book-details.component.html` and add edit routeLink in the last line.

<div class="row">

  <div class="col-md-12">

    <a [routerLink]="['/book-edit', book.\_id]" class="btn btn-success">EDIT</a>

  </div>

</div>

Now, open and edit `src/app/book-edit/book-edit.component.ts` then replace all codes with this.

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

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

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

@Component({

  selector: 'app-book-edit',

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

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

  encapsulation: ViewEncapsulation.None

})

export class BookEditComponent implements OnInit {

  book = {};

  constructor(private http: HttpClient, private router: Router, private route: ActivatedRoute) { }

  ngOnInit() {

    this.getBook(this.route.snapshot.params['id']);

  }

  getBook(id) {

    this.http.get('/book/'+id).subscribe(data => {

      this.book = data;

    });

  }

  updateBook(id, data) {

    this.http.put('/book/'+id, data)

      .subscribe(res => {

          let id = res['\_id'];

          this.router.navigate(['/book-details', id]);

        }, (err) => {

          console.log(err);

        }

      );

  }

}

Open and edit `src/app/book-edit/book-edit.component.html` then replace all codes with this.

<div class="container">

  <h1>Edit Book</h1>

  <div class="row">

    <div class="col-md-6">

      <form (ngSubmit)="updateBook(book.\_id)" #bookForm="ngForm">

        <div class="form-group">

          <label for="name">ISBN</label>

          <input type="text" class="form-control" [(ngModel)]="book.isbn" name="isbn" required>

        </div>

        <div class="form-group">

          <label for="name">Title</label>

          <input type="text" class="form-control" [(ngModel)]="book.title" name="title" required>

        </div>

        <div class="form-group">

          <label for="name">Author</label>

          <input type="text" class="form-control" [(ngModel)]="book.author" name="author" required>

        </div>

        <div class="form-group">

          <label for="name">Published Year</label>

          <input type="number" class="form-control" [(ngModel)]="book.published\_year" name="published\_year" required>

        </div>

        <div class="form-group">

          <label for="name">Publisher</label>

          <input type="text" class="form-control" [(ngModel)]="book.publisher" name="publisher" required>

        </div>

        <div class="form-group">

          <button type="submit" class="btn btn-success" [disabled]="!bookForm.form.valid">Update</button>

        </div>

      </form>

    </div>

  </div>

</div>

**11. Create Delete Function on Book-Detail Component**

Open and edit `src/app/book-detail/book-detail`.component.ts then add `Router` module to `@angular/router`.

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

Inject `Router` in the constructor params.

constructor(private router: Router, private route: ActivatedRoute, private http: HttpClient) { }

Add this function for delete book.

deleteBook(id) {

  this.http.delete('/book/'+id)

    .subscribe(res => {

        this.router.navigate(['/books']);

      }, (err) => {

        console.log(err);

      }

    );

}

Add delete button in `src/app/book-detail/book-detail.component.html` on the right of Edit routerLink.

<div class="row">

  <div class="col-md-12">

    <a [routerLink]="['/book-edit', book.\_id]" class="btn btn-success">EDIT</a>

    <button class="btn btn-danger" type="button" (click)="deleteBook(book.\_id)">DELETE</button>

  </div>

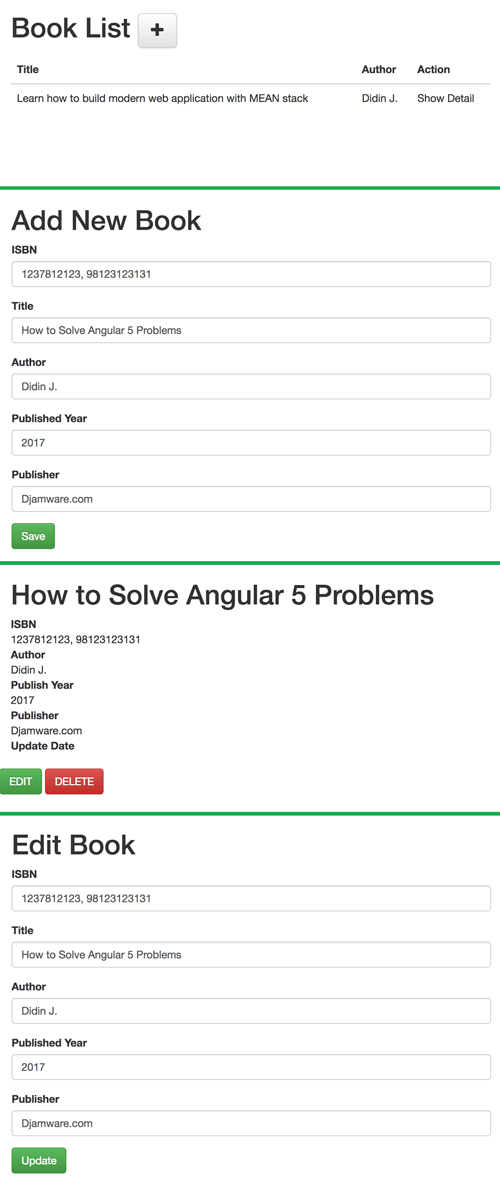
</div>

**12. Run and Test the MEAN Stack (Angular 5) CRUD Application**

Now, it's a time for testing the MEAN Stack (Angular 5) CRUD Application.

npm start

And here we are.



If you need the full working source code, you can find it on our [GitHub](https://github.com/didinj/mean-stack-angular5-crud.git).

That just the basic. If you need more deep learning about MEAN Stack, Angular, and Node.js, you can find the following books:

Düzeltmeler

Well written, thank you so much! The book edit function and the updated date didn't work correctly for me so I made the following changes:

1. book-detail.component.html   
<dd>{{ book.updated\_at }}</dd> ==> <dd>{{ book.updated\_date | date:'short' }}</dd>

2. book-edit.component.ts  
book = {}; ==> book: any = {};

and

updateBook(id, data) {  
this.http.put('/book/'+id, data).subscribe(...

becomes

updateBook(id) {   
this.book.updated\_date = [Date.now](http://disq.us/url?url=http%3A%2F%2FDate.now%3A9-PYgn9UhWHSNRM_AZu4qimtlFk&cuid=4805569)();   
this.http.put('/book/'+id, [this.book](http://disq.us/url?url=http%3A%2F%2Fthis.book%3AWZtwdAAXIJB8NpwOxfiObltQkrk&cuid=4805569)).subscribe(...