ICP7

Library Management System

Mean Stack

ICP Group 4

Name: Venkata Mahesh Mokkapati

Email: vmzwn@umsystem.edu

Partner Name: Sailaja Narra

Partner email: sntnn@umsystem.edu

Partner Repo: https://github.com/UMKC-APL-WebMobileProgramming/ICP7-sailajanarra

My report, video and Source code links:

Report:
https://drive.google.com/file/d/1U627vxKu3r9UmQp2MGyKo0y9Pdh003XX/view?usp=sharing

Video:
https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6e482364-c73c-4d98-8f7d-adbe003a55fb

Source: https://github.com/UMKC-APL-WebMobileProgramming/ICP7-Mahesh68

This task documentation (proper comments), Video, validation and responsiveness has also been handled.

In this task I learnt about connecting to database and full framed application. There is source code given which is having front end components and some node js configuration.

So here on click of one of the list elements, we can edit and also delete the element and store the state using mongo DB.

Database connection:

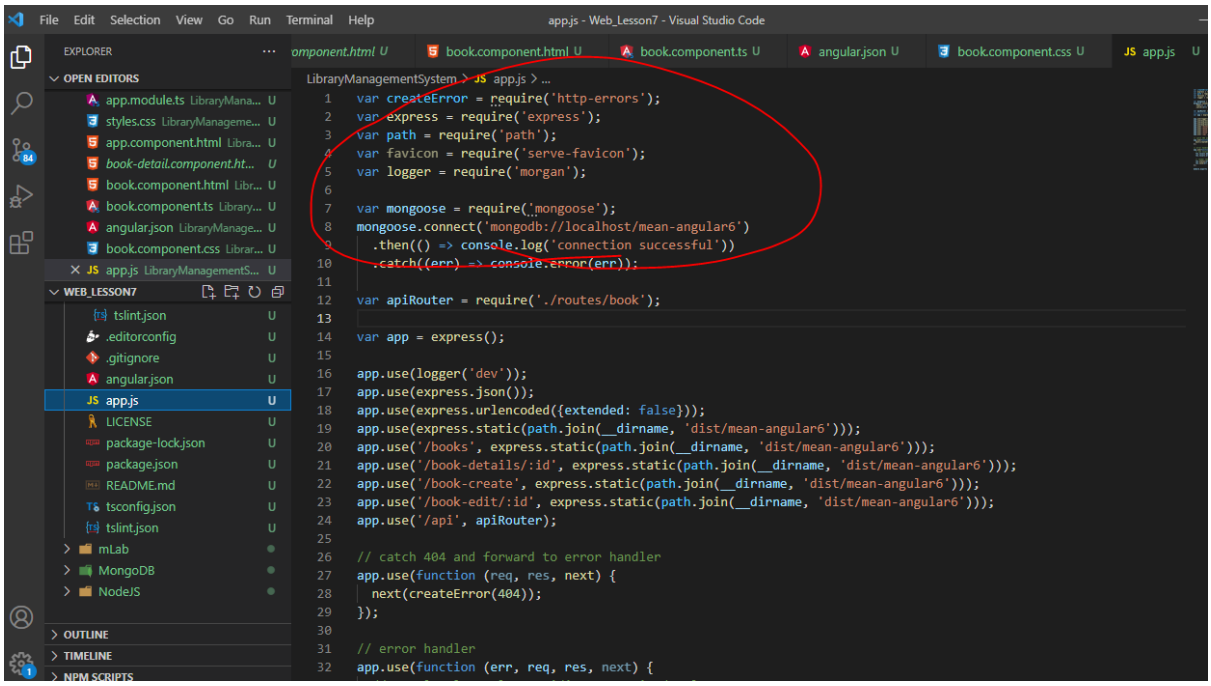
Installed monogo DB locally in my machine and installed node modules after downloading the content from the source.

Created some db path and used that folder for storing DB logs.

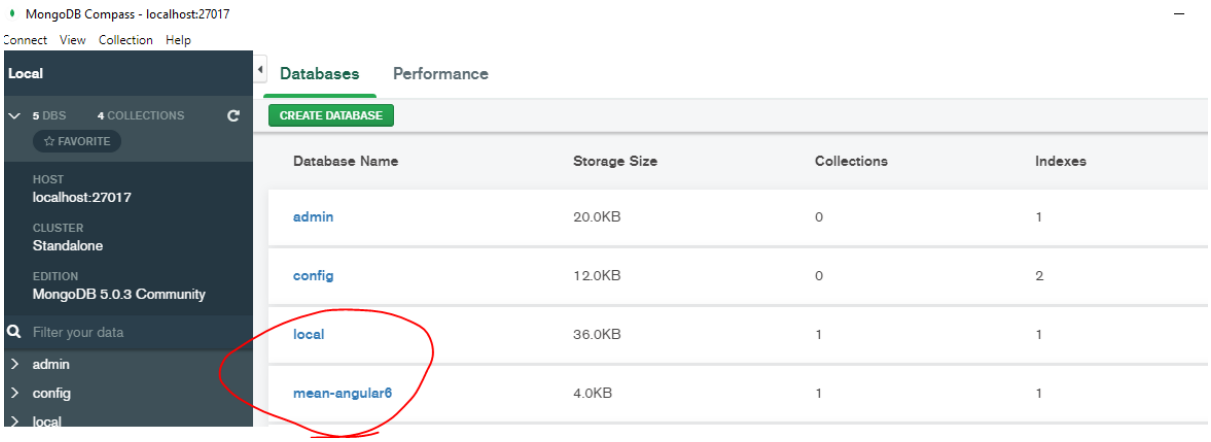
Run mongod.exe file to connect to database. Now open mongodb compass and create a new connection with the database url mentioned in the app.js file

```
mongodb://localhost/mean-angular6
```

Here is the connection in the app.js file

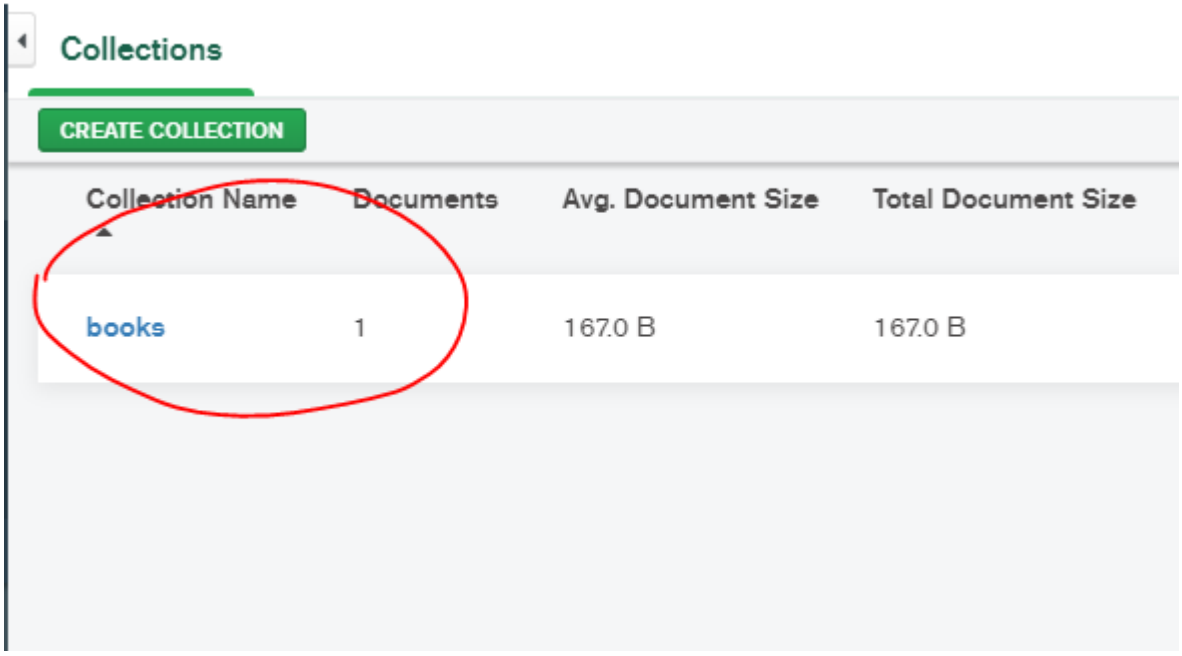


DB connections in the compass

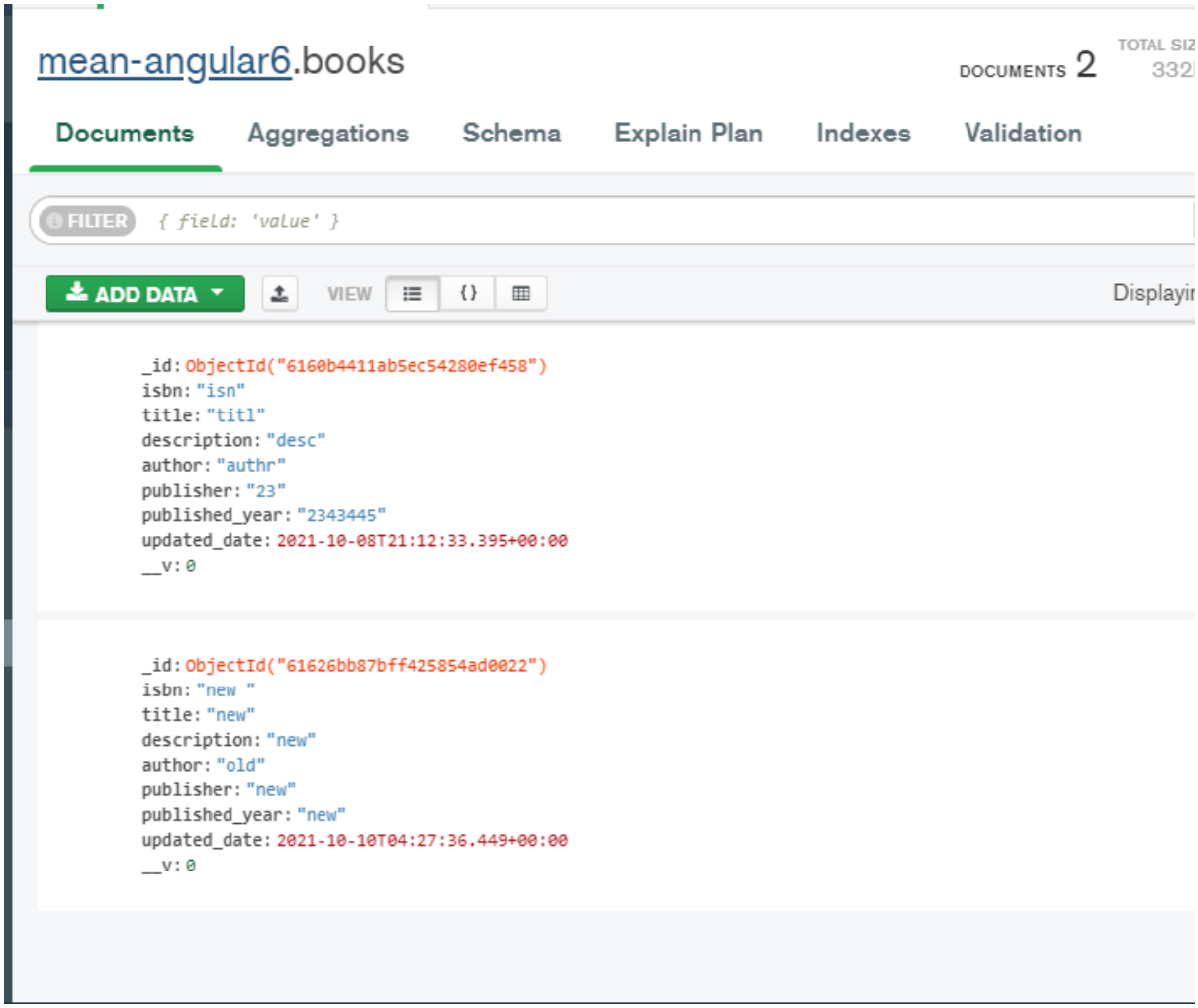


List of dbs are as shown above

Below are the collections in the connected database



Here is the list of documents inside a collection



We can create documents manually here in the compass or else using command prompt. And these will reflect in the front end application.

Now connecting front end to backend(node js)

As the controller in the front end will handle any http requests that are triggered by the user, there is also a controller in the backend that handles the client side requests and give response back to the client.

Here is the step by step connection that’s happening when a request is made from the front end:

Appservice.ts

```
module.ts U   styles.css U   app.component.html U   book.component.html U   book.component.ts U   api.service.ts U
LibraryManagementSystem > src > app > api.service.ts > ...
import { Injectable } from "@angular/core";
import { Observable, throwError } from "rxjs";
import {
  HttpClient,
  HttpErrorResponse,
  HttpHeaders,
} from "@angular/common/http";
import { catchError, map } from "rxjs/operators";

const httpOptions = {
  headers: new HttpHeaders({ "Content-Type": "application/json" }),
};
const apiUrl = "/api";

@Injectable({
  providedIn: "root",
})
export class ApiService {
  constructor(private http: HttpClient) {}

  getBooks(): Observable<any> {
    return this.http
      .get(apiUrl, httpOptions)
      .pipe(map(this.extractData), catchError(this.handleError));
  }

  getBook(id: string): Observable<any> {
    const url = `${apiUrl}/${id}`;
    return this.http
      .get(url, httpOptions)
      .pipe(map(this.extractData), catchError(this.handleError));
```

App.js file which is having same api url:

```
View  Go  Run  Terminal  Help        app.js - Web_Lesson7 - Visual Studio Code
book.component.html U   book.component.ts U   api.service.ts U   angular.json U   book.component.css U   app.js
LibraryManagementSystem > app.js > ...
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/mean-angular6')
  .then(() => console.log('connection successful'))
  .catch((err) => console.error(err));

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

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({extended: false}));
app.use(express.static(path.join(__dirname, 'dist/mean-angular6')));
app.use('/books', express.static(path.join(__dirname, 'dist/mean-angular6')));
app.use('/book-details/:id', express.static(path.join(__dirname, 'dist/mean-angular6')));
app.use('/book-create', express.static(path.join(__dirname, 'dist/mean-angular6')));
app.use('/book-edit/:id', express.static(path.join(__dirname, 'dist/mean-angular6')));
app.use('/api', apiRouter);

// catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});

// error handler
```

So the api router has all the CRUD operations which is book.js

```
var express = require("express");
var router = express.Router();
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", async function (req, res, next) {
  Book.findByIdAndUpdate(req.params.id, req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
```

If the url is “/” and if a get request is reached to server then the server will return this response back to the client.

Here the logical code for the update and delete methods are same as get and save. Only difference is the http calling methods.

```
  });
});

/* 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", async 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) {
  console.log(req.params.id);
  Book.findByIdAndDelete(req.params.id, (err, response) => {
    if (err) return next(err);
    res.json(response);
  });
});
```

Ln 41, Col 31 Spaces: 2 UTF-8

Here once user clicks on any element URL will also get updated along with the ID of the book that is clicked.

So to do some manipulations for that book we can make use of the params in the URL.

Params is an inbuilt method to both angular and node which gives the values of query parameters in the URL. So navigation and handling URL is the main thing that has to be handled because based on navigation and url change different components are getting loaded and CRUD operations are getting called.

Book create component

Terminal Help book.component.html - Web_Lesson7 - Visual Studio Code

JS book.js U | app.module.ts U | styles.css U | app.component.html U | book.component.html U | book.component.ts U

LibraryManagementSystem > src > app > book > book.component.html > div.example-container.mat-elevation-z8 > table > ng-container

```html
<div class="button-row">
  <a
    [routerLink]="['/book-create']"
    color="primary"
    mat-raised-button
    class="add-book"
  >
    <mat-icon>add</mat-icon><span class="ml-2">Add book</span>
  </a>
</div>
<div class="example-container mat-elevation-z8">
  <table #table [dataSource]="dataSource" mat-table>
    <!--- Note that these columns can be defined in any order.
          The actual rendered columns are set as a property on the row definition" -->

    <!-- Title Column -->
    <ng-container matColumnDef="isbn">
      <th *matHeaderCellDef mat-header-cell>ISBN</th>
      <td *matCellDef="let element" class="isbn-col" mat-cell>
        {{ element.isbn }}
      </td>
    </ng-container>

    <!-- Title Column -->
    <ng-container matColumnDef="title">
      <th *matHeaderCellDef mat-header-cell>Title</th>
      <td *matCellDef="let element" class="isbn-col" mat-cell>
        {{ element.title }}
      </td>
    </ng-container>
```

Run Terminal Help book.component.ts - Web_Lesson7 - Visual Studio Code

JS book.js U | app.module.ts U | styles.css U | app.component.html U | book.component.html U | book.component.ts U

LibraryManagementSystem > src > app > book > book.component.ts > BookComponent > constructor

```typescript
import { Component, OnInit } from "@angular/core";
import { ApiService } from "../api.service";
import { DataSource } from "@angular/cdk/collections";

@Component({
  selector: "app-book",
  templateUrl: "./book.component.html",
  styleUrls: ["./book.component.css"],
})
export class BookComponent implements OnInit {
  books: any;
  displayedColumns = ["isbn", "title", "author"];
  dataSource = new BookDataSource(this.api);

  constructor(private api: ApiService) {}

  ngOnInit() {
    this.api.getBooks().subscribe(
      (res) => {
        console.log(res);
        this.books = res;
      },
      (err) => {
        console.log(err);
      }
    );
  }
}

export class BookDataSource extends DataSource<any> {
  constructor(private api: ApiService) {
```

Book edit component

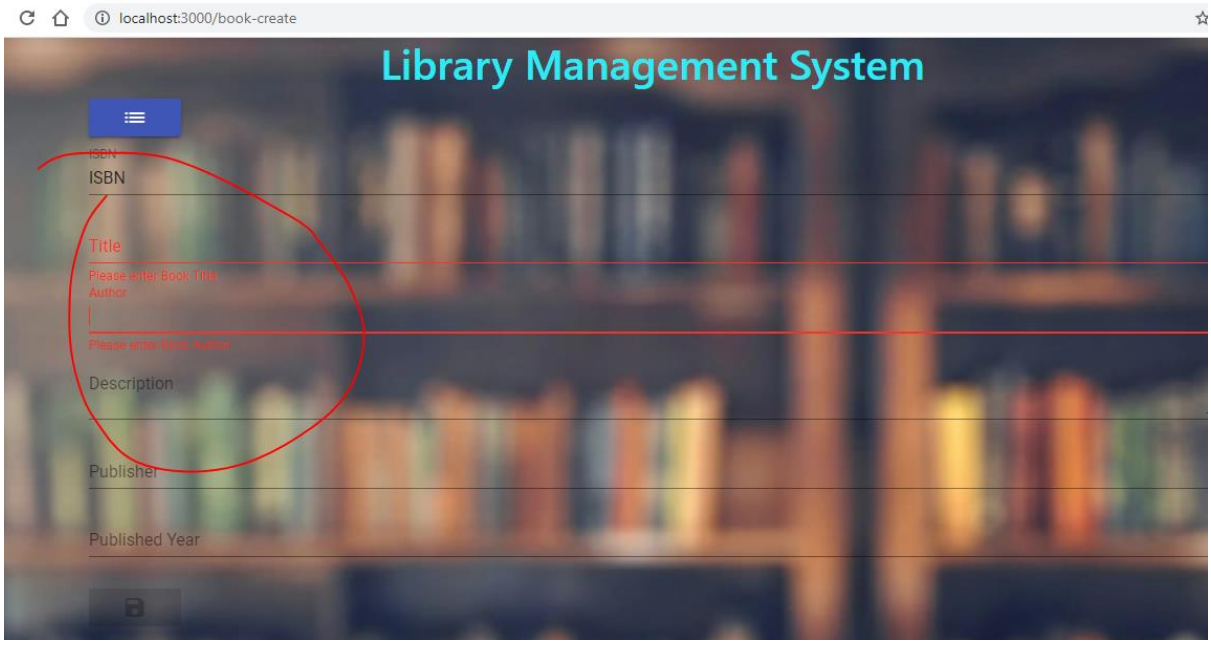
```html
<!-- write code to edit and save book i this component -->
<form [formGroup]="bookForm" (ngSubmit)="onFormSubmit(bookForm.value)">
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="ISBN"
      formControlName="isbn"
      [errorStateMatcher]="matcher"
    />
    <mat-error>
      <span *ngIf="!bookForm.get('isbn').valid && bookForm.get('isbn').touched"
        >Please enter ISBN</span
      >
    </mat-error>
  </mat-form-field>
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="Title"
      formControlName="title"
      [errorStateMatcher]="matcher"
    />
    <mat-error>
      <span
        *ngIf="!bookForm.get('title').valid && bookForm.get('title').touched"
        >Please enter Book Title</span
      >
    </mat-error>
  </mat-form-field>
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="Author"
```

Here the list of form fields are created using form builder which is a template driven form.

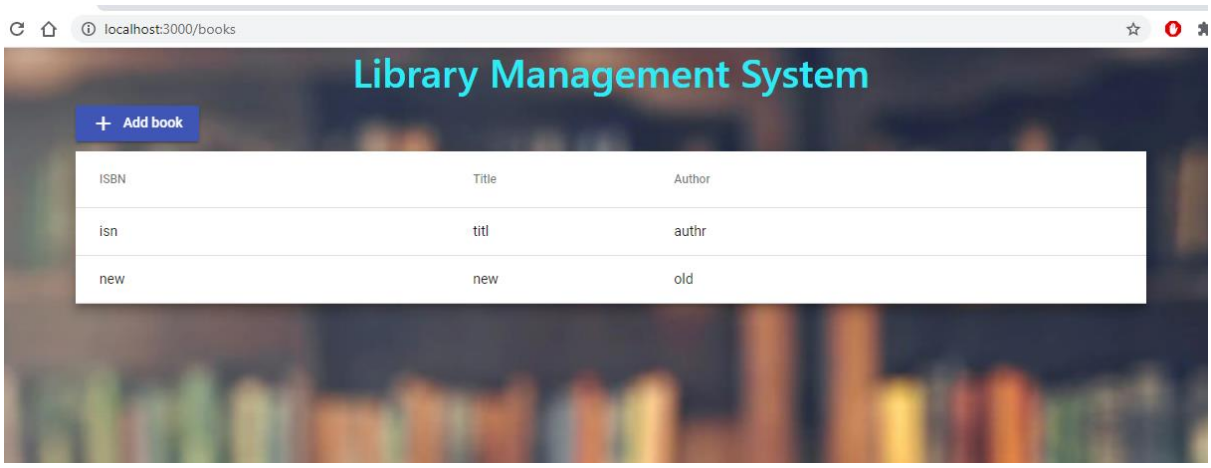
Using mat module handled the input required field errors which is as shown below:

```
<!-- write code to edit and save book i this component -->
<form [formGroup]="bookForm" (ngSubmit)="onFormSubmit(bookForm.value)">
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="ISBN"
      formControlName="isbn"
      [errorStateMatcher]="matcher"
    />
    <mat-error>
      <span *ngIf="!bookForm.get('isbn').valid && bookForm.get('isbn').touched"
        >Please enter ISBN</span
      >
    </mat-error>
  </mat-form-field>
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="Title"
      formControlName="title"
      [errorStateMatcher]="matcher"
    />
    <mat-error>
      <span
        *ngIf="!bookForm.get('title').valid && bookForm.get('title').touched"
        >Please enter Book Title</span
      >
    </mat-error>
  </mat-form-field>
  <mat-form-field class="example-full-width">
    <input
      matInput
      placeholder="Author"
```

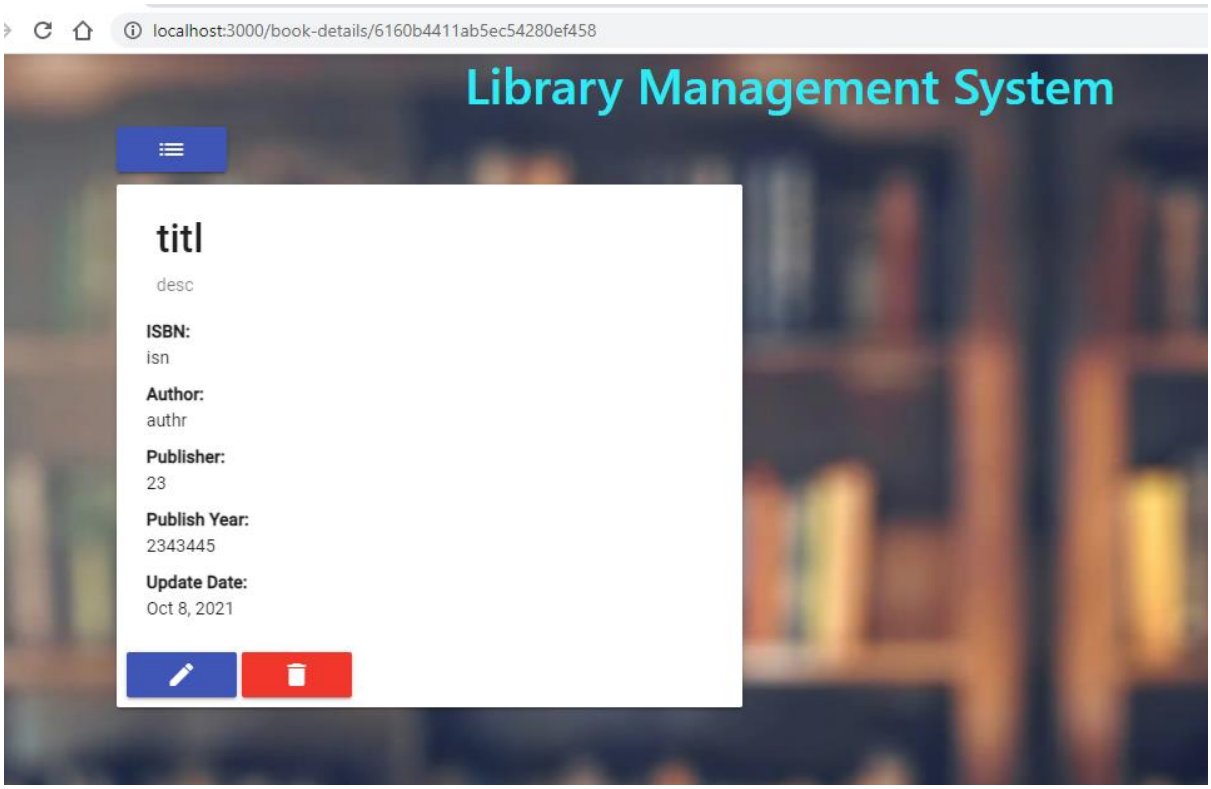
So if any form fields are not specified then there is a mat error displayed to the user



List of books



When user clicks on one of the element, book details component will get loaded and here is the page:



So here are the two buttons for edit and delete that book from the database:

For editing bookedit component is used which loads the form as explained above

```
ngOnInit() {
  this.bookForm = this.formBuilder.group({
    isbn: "",
    title: "",
    description: "",
    author: "",
    publisher: "",
    published_year: "",
  });
  this.api
    .getBook(this.route.snapshot.params["id"])

    .subscribe((data) => {
      console.log(data.isbn);
      this.bookForm = this.formBuilder.group({
        isbn: data.isbn,
        title: data.title,
        description: data.description,
        author: data.author,
        publisher: data.publisher,
        published_year: data.published_year,
      });
    });
}
onFormSubmit(form: NgForm) {
  this.api.updateBook(this.route.snapshot.params["id"], form).subscribe(
    (res) => {
      let id = this.route.snapshot.params["id"];
      this.router.navigate(["/book-details", id]);
```

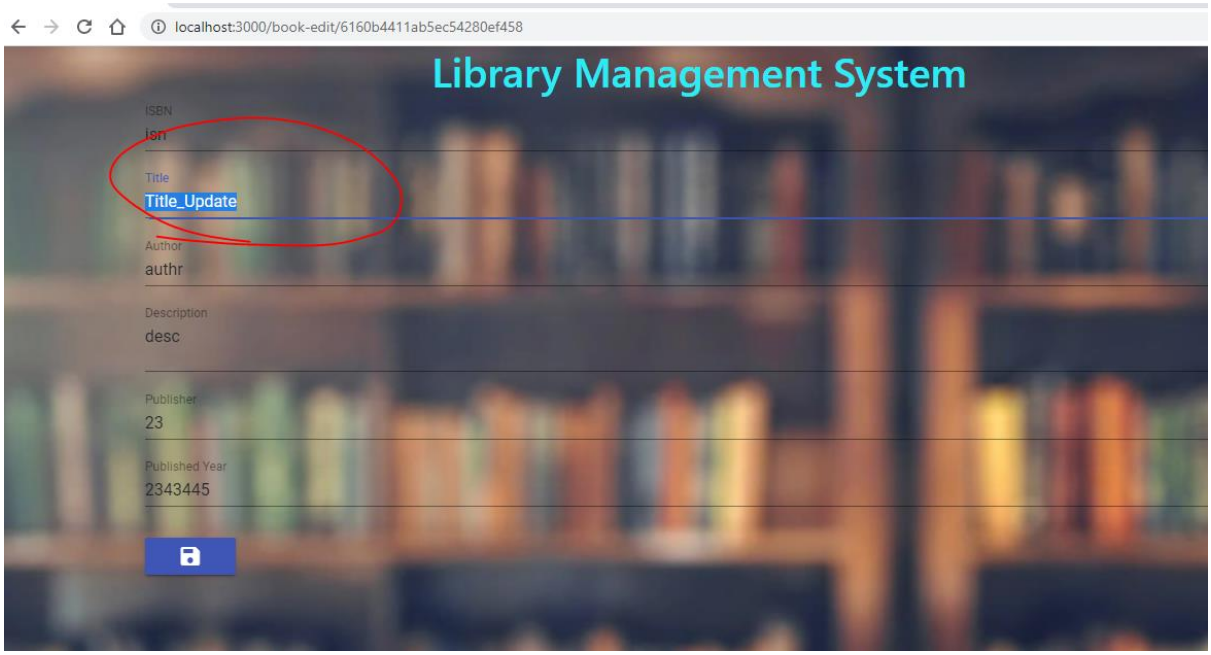
Put request in the node server will get called after successful submission of the book edit.

```
  });

  /* UPDATE BOOK */
  router.put("/:id", async function (req, res, next) {
    Book.findByIdAndUpdate(req.params.id, req.body, function (err, post) {
      if (err) return next(err);
      res.json(post);
    });
  });

  /* DELETE BOOK */
```

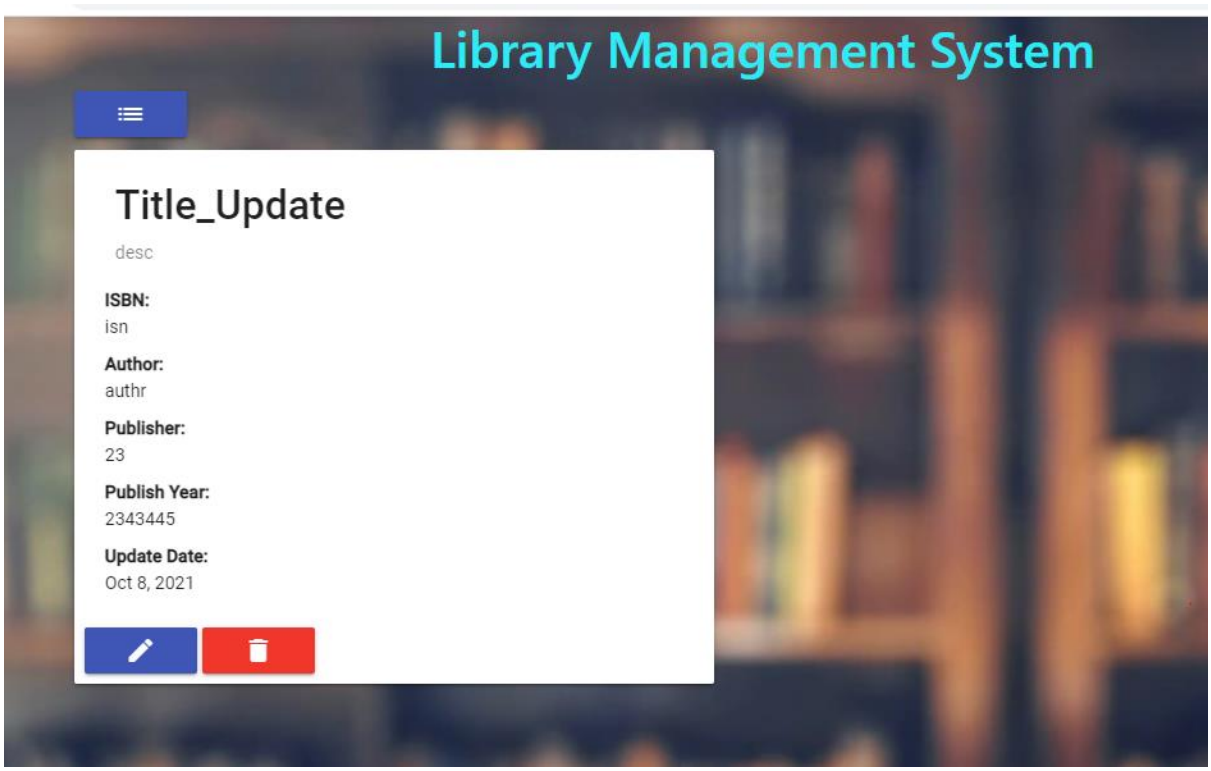
Here async and await methods are used to make the javascript ajax call synchronous . Here the response will be a promise so by the time we get response this line of code will get executed and resolved.

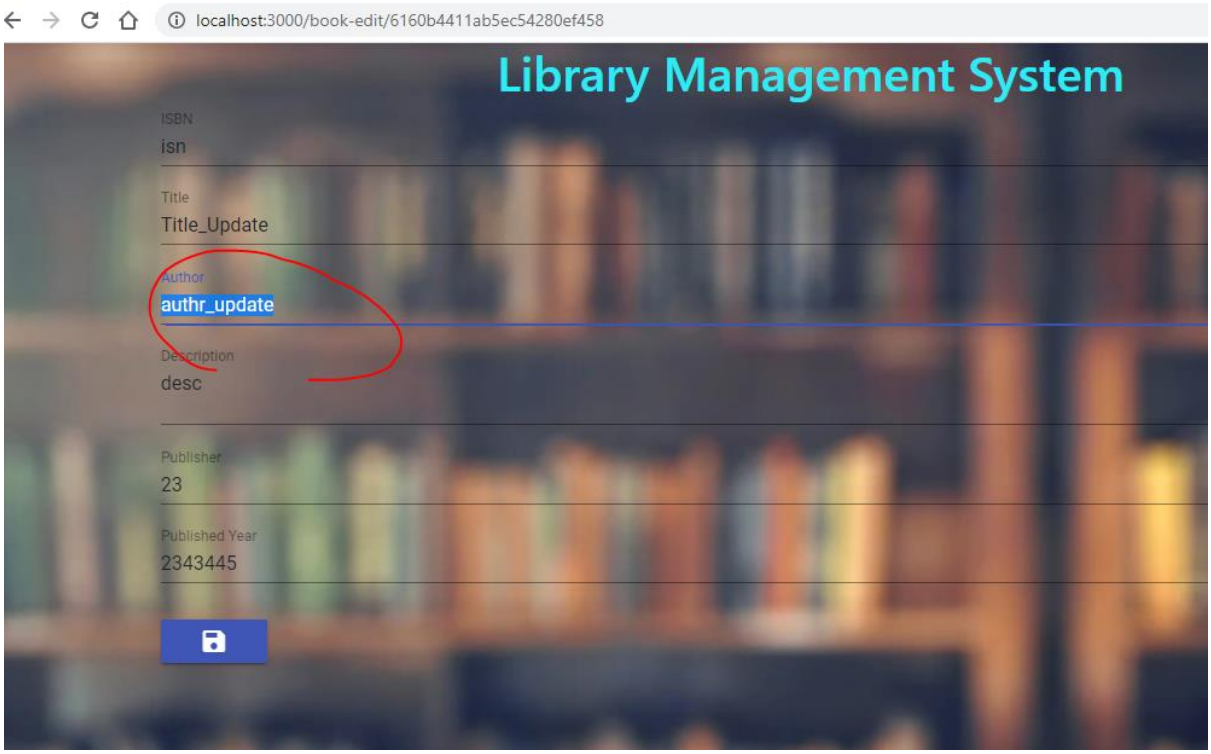
Hence using await function this method will wait for the promise to get resolved which means successful updation to DB and fetch the response from the backend.

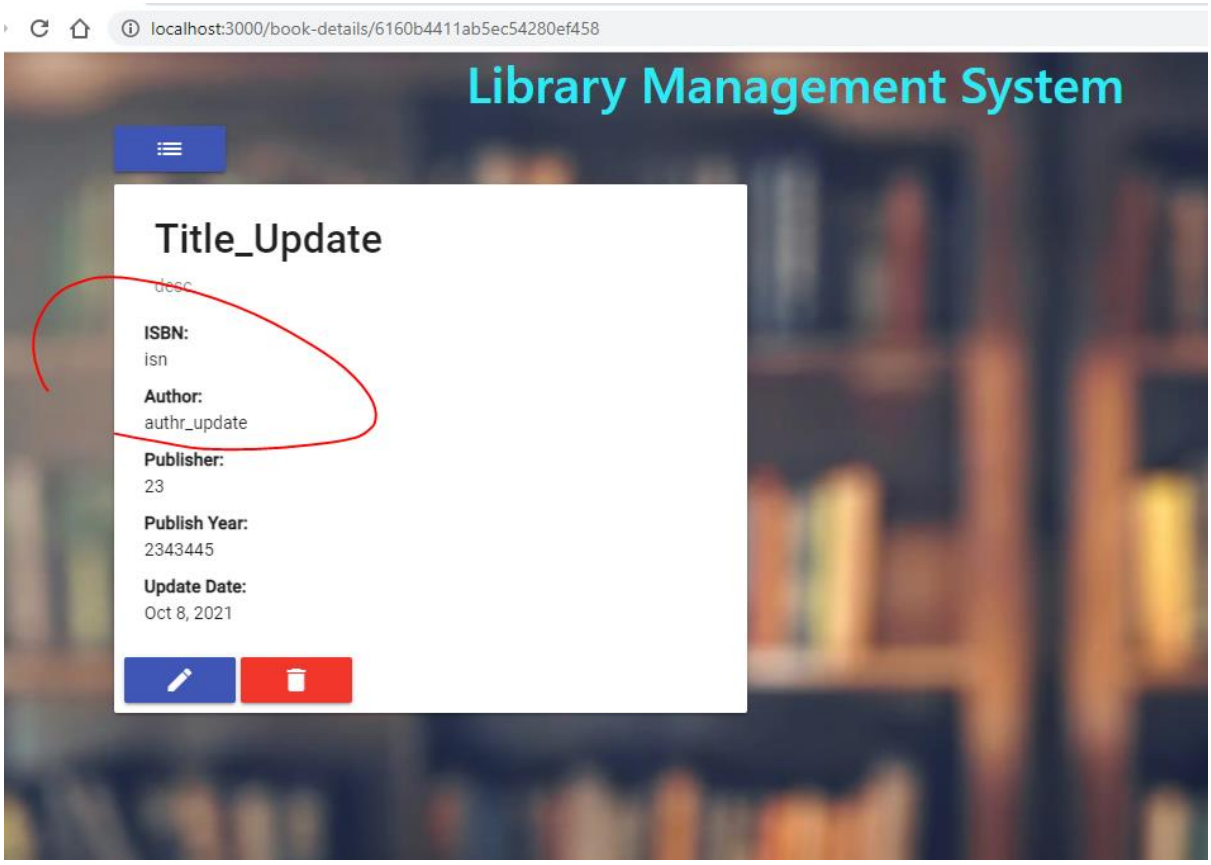
Here updated the title in the book and saved.

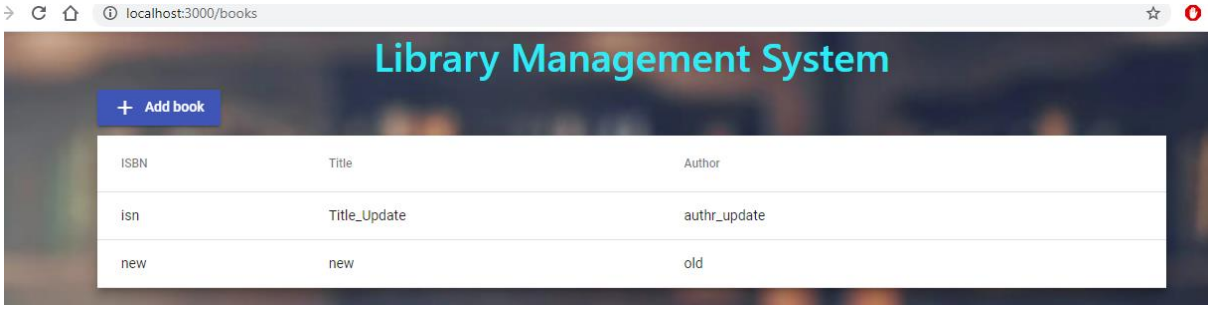


On successful update to DB, front end will get navigated to book details page with the updated values



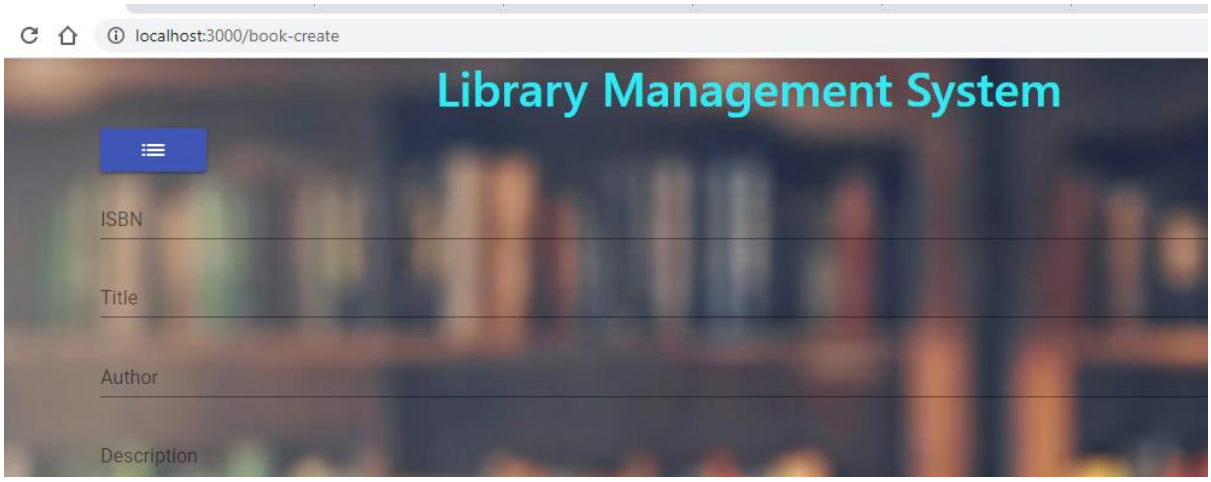

localhost:3000/book-edit/6160b4411ab5ec54280ef458
Library Management System
ISBN
isn
Title
Title_Update
Author
authr_update
Description
desc
Publisher
23
Published Year
2343445

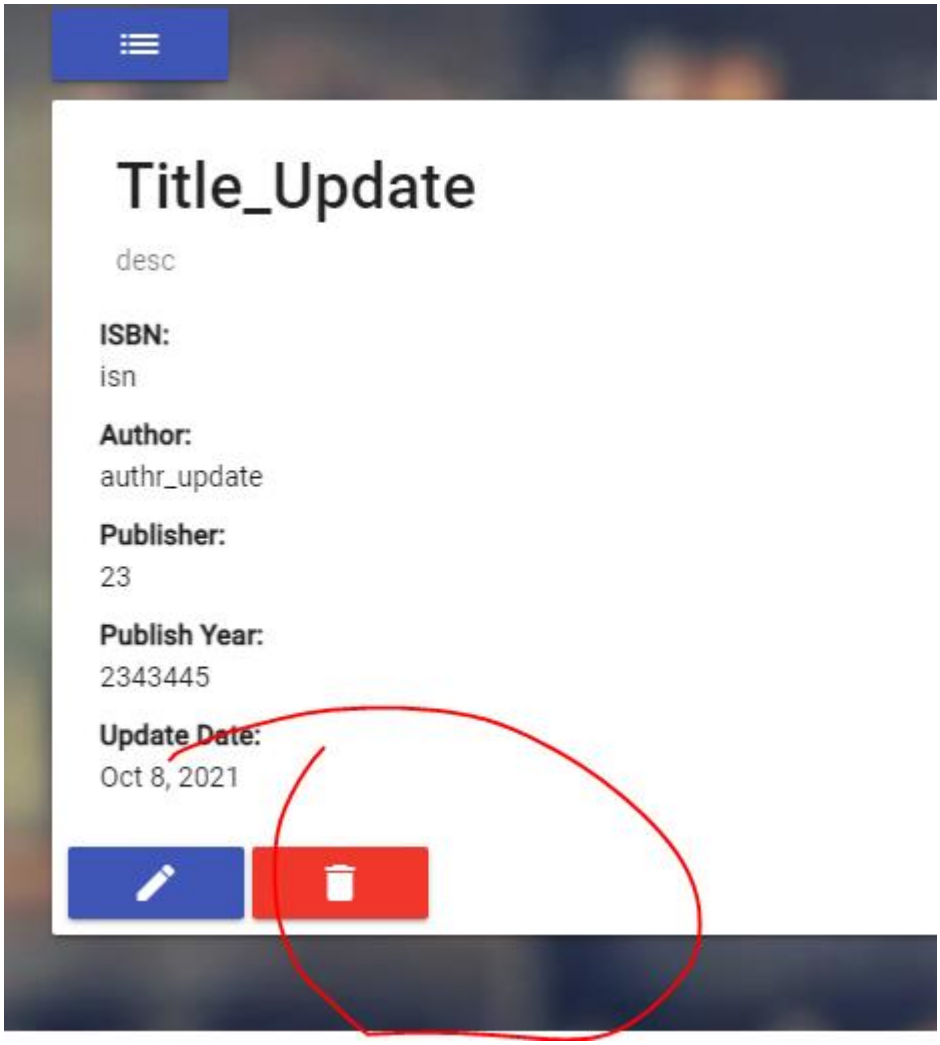

localhost:3000/book-details/6160b4411ab5ec54280ef458
Library Management System
Title_Update
desc
ISBN:
isn
Author:
authr_update
Publisher:
23
Publish Year:
2343445
Update Date:
Oct 8, 2021



If user want to create a new book, this add book button will serve the purpose.

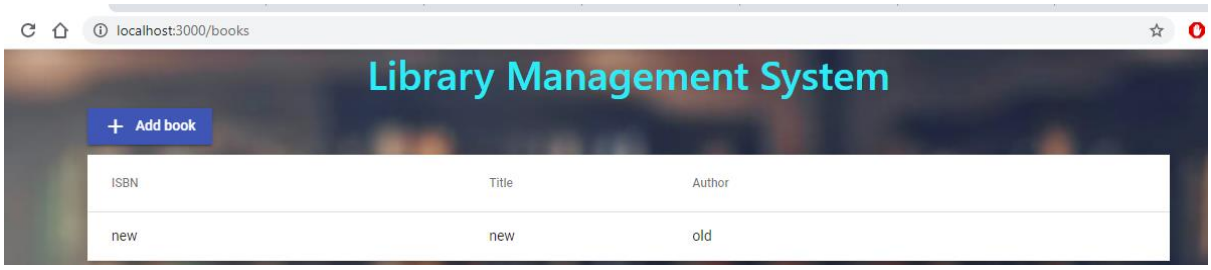
On click of add button we will get the details of the book with the form fields editable



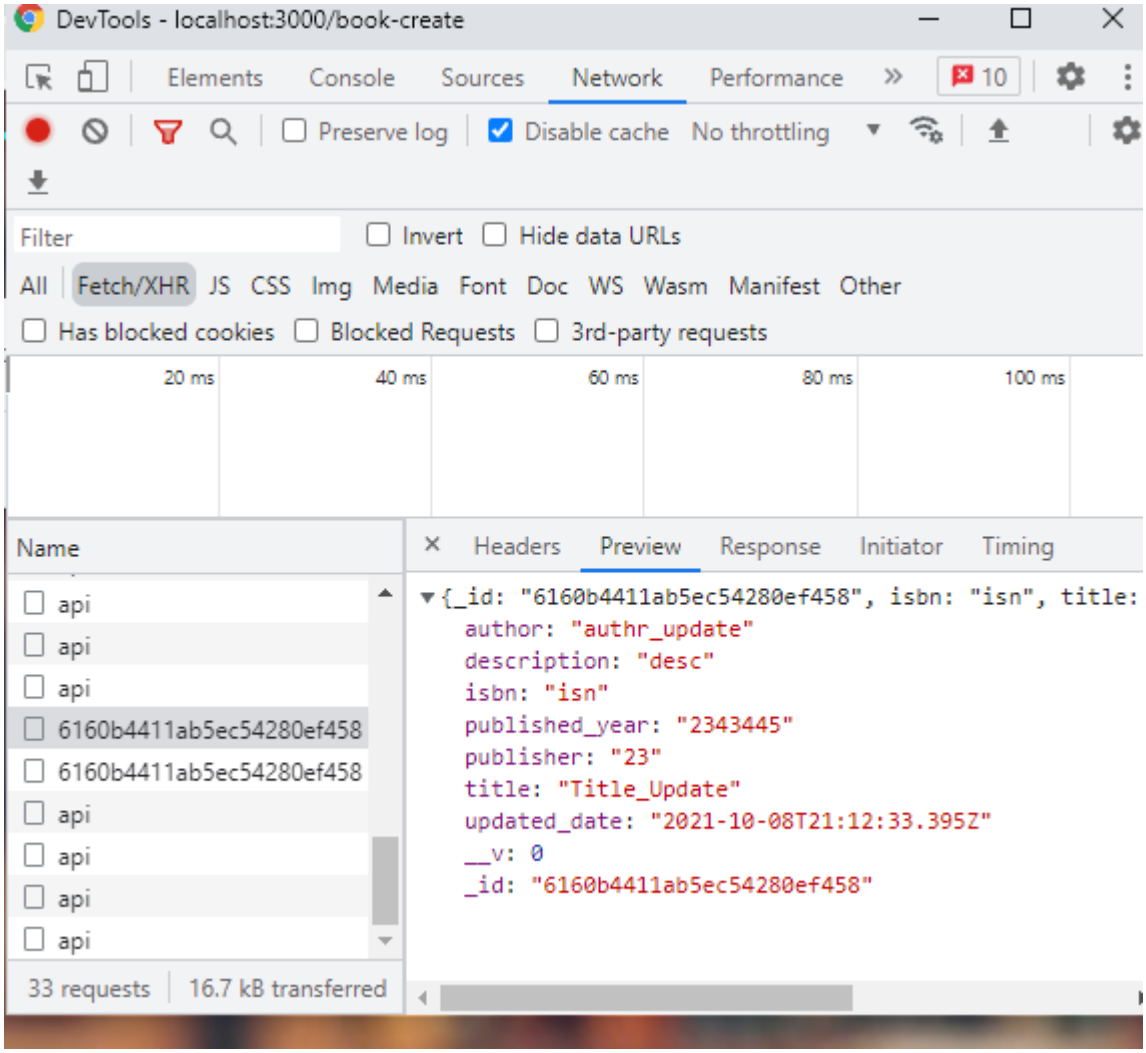


On click of delete button, delete method will get called and in the DB using findOneAndRemove method will get called and removed from the mongo database.

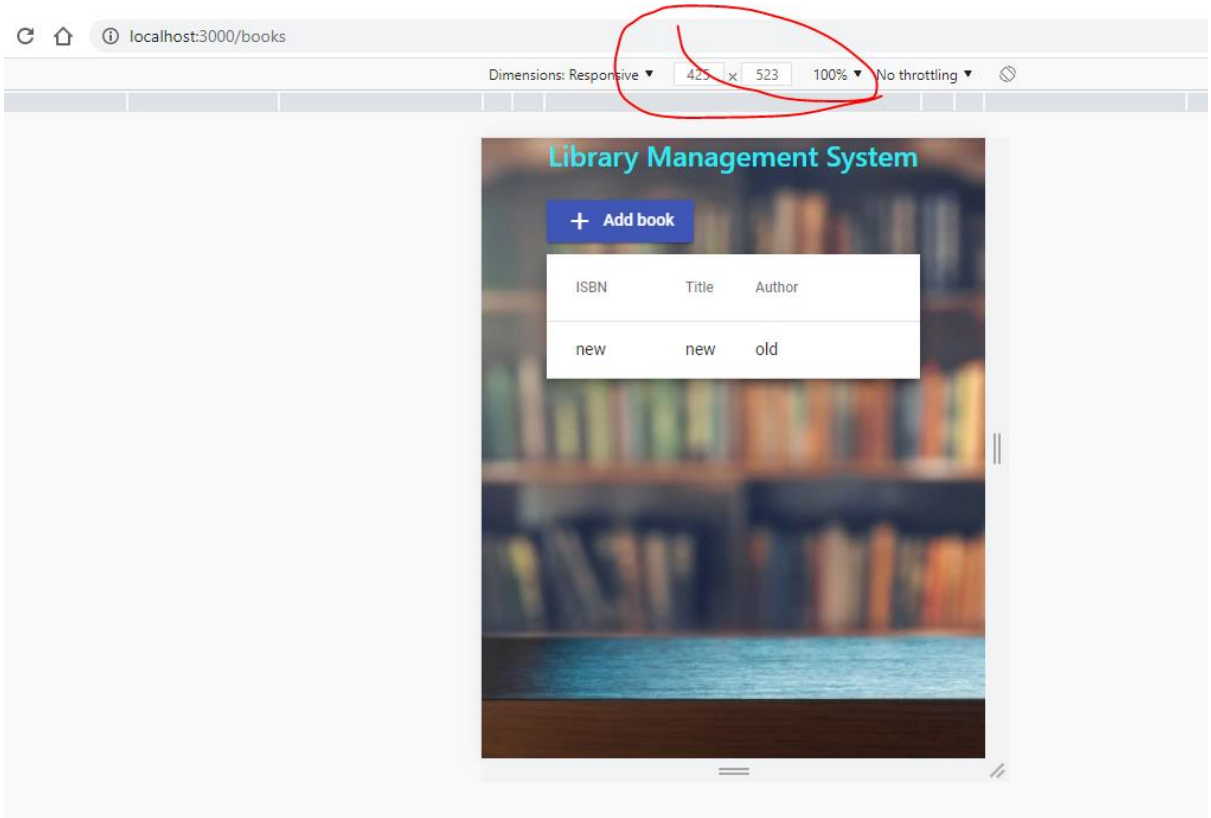
On successful removal from the DB, page will get navigated to the book component an updated books list will be shown.

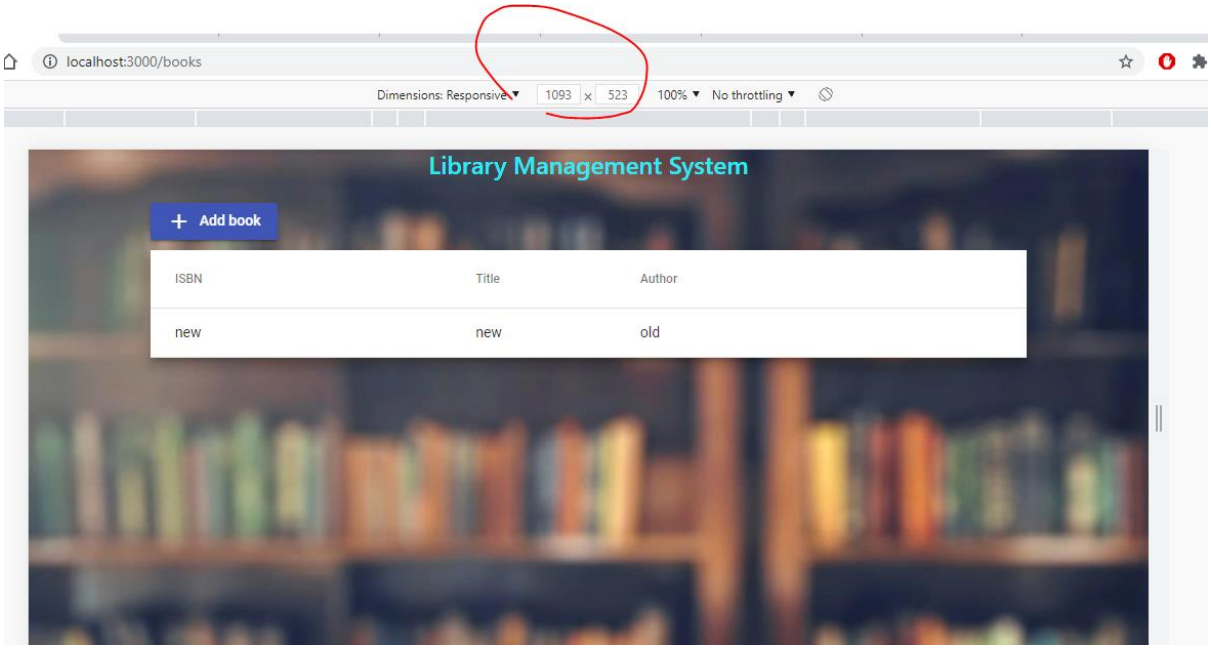


Here are responses in the network tab



Responsive page:


localhost:3000/books
Dimensions: Responsive
425
523
100%
No throttling
Library Management System
Add book
ISBN
Title
Author
new
new
old


localhost:3000/books
Dimensions: Responsive
1093
523
100%
No throttling
Library Management System
Add book
ISBN
Title
Author
new
new
old