

ASP.NET Core Practice - Library Due Date Tracker Day 2

Deadline: Wednesday. September 23 2020, 09:00AM

[GitHub Classroom Link](#)

Introduction

This assignment is meant to challenge your mastery of ASP.NET Web Application (Model - View - Controller) and how well you are able to use MVC to create a CRUD application. Your goal in this assignment is to create a tool that will help you keep track of all the books you have checked out of the library. This is a cumulative activity. Use your code from ASP.NET Core Assignment - Library Due Date Tracker Day 1 as a starting point.

Requirements

- ☐ Book class (Model) **modified** to serve as a database code-first class (all properties public, not null unless specified otherwise):
 - ☐ int "ID" - int(10) (primary key)
 - ☐ string "Title" - varchar(30)
 - ☐ DateTime "PublicationDate" - date
 - ☐ DateTime "CheckedOutDate" - date
 - ☐ DateTime "DueDate" - date
 - ☐ DateTime "ReturnedDate" - date (nullable)
 - ☐ int "AuthorID" - int(10) (foreign key)
 - ☐ Requisite virtual property for foreign key.
- ☐ **Add** a code-first Author class (Model) (all properties public, not null unless specified otherwise):
 - ☐ int "ID" - int(10) (primary key)
 - ☐ string "Name" - varchar(30)
 - ☐ Requisite virtual property and constructor for foreign key.
- ☐ **Add** a LibraryContext class (Context):
 - ☐ All requisite methods and properties to function as a context.
 - ☐ Database connection string to a database called "mvc_library".
 - ☐ Seed the database with:
 - ☐ 5 "Authors" of your choice.
 - ☐ 3 "Books" by the same Author.
 - ☐ All three books must have a "CheckoutDate" equal to December 25, 2019.
 - ☐ One book must be returned on-time with no extension.
 - ☐ One book must be returned on-time with one extension.
 - ☐ One book must not have been returned at all!
 - ☐ Add migrations and update the database once this and the models are completed.
- ☐ **Add** a scaffolded controller and views for the "Author" model (using "LibraryContext").
 - ☐ Do **NOT** use scaffolding for the "Book" model, continue to use the manually generated Controller and Views from yesterday (ASP.NET Core Assignment - Library Due Date Tracker Day 1).

- ☐ BookController (Controller) class **modified**:
 - ☐ Remove the “Books” property (static list of Books).
 - ☐ Modify “ExtendDueDateForBookByID()” to update a book in the database using Entity Framework.
 - ☐ Modify “DeleteBookByID()” to delete a book from the database using Entity Framework.
 - ☐ Add a “GetBooks()” method to get a list of all books in the database using Entity Framework.
 - ☐ Ensure that the “Author” virtual property is populated before the list is returned (for use on the List view).
 - ☐ Modify “GetBookByID()” to get a specific book from the database.
 - ☐ Ensure that the “Author” virtual property is populated before the object is returned (for use on the Details view).
 - ☐ Modify “CreateBook()” to save books to a database using Entity Framework.
 - ☐ Have “CreateBook()” perform the nulling of “ReturnDate”.
 - ☐ Have “CreateBook()” perform the setting of “DueDate”.
- ☐ Book Create (View) **modified**:
 - ☐ Have a dropdown (select) to select the author.
 - ☐ Populate the dropdown (select) based on the author table.
- ☐ Book List (View) **modified**:
 - ☐ Modify the output to account for the new “Author” model (show Author “Name”).
- ☐ Book Details (View) **modified**:
 - ☐ Modify the output to account for the new “Author” model (show Author “Name”).
- ☐ An Edit view for the Book is **NOT** necessary. The “Extend” / “Delete” buttons on the Details view will suffice.

Challenges (See Rubric for Details)

- ☐ Make it look nice with CSS
- ☐ Have an unexpected feature
- ☐ Modify the Details view to show the user how many days a book is overdue.
- ☐ Which Author has been checked out the longest?
 - ☐ Write a new Action and View that will determine which author’s books have the longest cumulative checked-out time.
 - ☐ This should work with books that haven’t been returned, as well as on books that have been returned.
- ☐ Add validation that the book cannot be returned prior to checkout, and cannot be checked out prior to publishing.
- ☐ Research how to display a list of exceptions on the Create view, and list all fields that do not have data on a failed submit.

Hints

- General Hints:
 - Focus on the requirements first, challenges are extra!
 - This kind of project has been done by many others in the past! Don’t hesitate to use your Google-Fu skills if you don’t know how to implement certain features!
 - Please include source citations in your code and README.md
- Day 1 Hints:
 - If you are struggling with the Book class, look back at other class examples done during C# (Such as the Car and Pen classes during OOP)
 - Look up how the DateTime class works for C#, this will help you easily keep track of dates

- The Book class has properties defined, the BookController : Controller class is where all your data manipulation methods will be contained
- Day 2 Hints:
 - Remember you must use a database to store all information, the information should not change if a session is switched or the page is refreshed
 - Your Book ID should no longer be user defined, but be generated by the database, search up how to use the “auto increment” attribute if you are struggling
 - See the Views/PhoneNumber/Create.cshtml file from today’s repository for an example of select usage.
 - Install:
 - dotnet add package Microsoft.EntityFrameworkCore.Design
 - dotnet add package Pomelo.EntityFrameworkCore.MySql
 - dotnet add package Microsoft.EntityFrameworkCore.SqlServer
 - If your scaffolded views are throwing exceptions, remember to add “services.AddDbContext()” to Startup.cs (see repo).

Citation Guide for Borrowed Code

Whenever you borrow code, the following information must be included:

- Comments to indicate both where the borrowed code begins and ends.
- A source linking to where you found the code (URL, book, example, etc.).
- Your reason for adding the code to your assignment or project instead of writing it out yourself.
- Explain to us how the code is supposed to work, include links to documentation and articles you read to help you understand.
- A small demonstration to prove you understand how the code works.

```

1  const inputArr = [5,1,3,4,2];
2
3  /*Borrowed code for bubbleSort starts*/
4  let bubbleSort = (inputArr) => {
5      let len = inputArr.length;
6      for (let i = 0; i < len; i++) {
7          for (let j = 0; j < len; j++) {
8              if (inputArr[j] > inputArr[j + 1]) {
9                  let tmp = inputArr[j];
10                 inputArr[j] = inputArr[j + 1];
11                 inputArr[j + 1] = tmp;
12             }
13         }
14     }
15     return inputArr;
16 };
17
18 /*Borrowed code from bubbleSort ends*/
19
20 //Source: bubbleSort function obtained from https://medium.com/javascript-algorithms/javascript-algorithms-bubble-sort-3d27f285c3b2
21 //Reason to add: implementing bubble sort can be tedious and bug prone, it would be better to use a proven version than to write my own
22 //How it works: I read the following article to understand how bubble sorts work (http://www.pkirs.utep.edu/CIS3355/Tutorials/chapter9/tutorial19A/bubblesort.htm)
23 //Demonstration of understanding:
24 //Example array: [3,1,2]
25 //Step 1: Compare 3 and 1. Since 1 is smaller, swap places.
26 //Array: [1,3,2]
27 //Step 2: Compare 3 and 2. Since 2 is smaller, swap places.
28 //Array: [1,2,3]
29 //Step 3: Compare 1 and 2. No need to swap.
30 //Array: [1,2,3]
31 //Step 4: Compare 2 and 3. No need to swap.
32 //Array: [1,2,3]
33 //Function complete.
34 console.log(bubbleSort(inputArr));

```