C# Review/OOP Practice - Entity Framework Code First Migration Updates

Deadline: Monday, September 21st 2020, 09:00 AM

GitHub Classroom Repository

Introduction

This practice is designed to continue to help you familiarize yourself with the fundamentals of using Entity Framework to generate and update a database. This scenario is for when you would like to build your server and database both using C#. For this practice we will be expanding upon the Entity Framework Practice - Code First Migration practice by adding more columns to our "Shelves" table and adding more tables as well.

Requirements

- Use Entity Framework Practice - Code First Migration practice as a starting point for this practice.
- (Optional) - Change "Shelves" to "Shelf" in all cases where it should be singular.
- Modify the "Shelves" model:
    - A table called "Shelves" with:
        - An int primary key called ID.
        - A varchar of length 50 called "Name".
        - A foreign key to ShelfMaterial(ID) called "ShelfMaterialID".
        - The requisite virtual properties and constructors for references.
- Create a "ShelfMaterials" model:
    - A table called ShelfMaterial with:
        - An int primary key called "ID".
        - A varchar of length 25 called "MaterialName".
        - The requisite virtual properties for references.
- Use Entity Framework to update the "code_first_practice" database using your models.
    - This must be done in a **migration**.
- Update your code in Program.cs to include:
    - A parameter for "shelfMaterial"
        - Ensure the material exists in the "ShelfMaterial" table
            - if material does not exist, let the user know and exit.
            - "shelfMaterial" parameter should be case insensitive.
            - "shelfMaterial" parameter should be trimmed.
        - When you add the new shelf, ensure the foreign key for "ShelfMaterial" is populated correctly.

Challenges

- Research Data Seeding - EF and seed the table with 10 shelves consisting of at least 3 different materials when you push it to MySQL.

- If the shelf material the user enters doesn't exist, ask if they would like to add it. Ensure that it is stored in "Title Case" (upper case first letter of each word).

Reminders

Ensure you are tracking your time and that your timesheet is in the appropriate folder for viewing and marking.

Ensure you are committing frequently. It is advised to commit once per successful feature implementation at a minimum.

Ensure you are pushing your repository to the GitHub Classroom repository, and not a personal, private repository.

Ensure your repository has a readme with at a minimum your name, the name of the project, the project's purpose and a link to your Trello board.

Ensure you are using Trello appropriately to keep track of outstanding features, and that it is linked in the README.md file.

Ensure you are using the #class channel to request clarifications on assignment specifications.

Ensure you are using the #homework-help channel in slack to request for assistance from instructional staff if needed, and that you include (at a minimum) a specific description of the problem and a list of what you have tried.

Feel free to reach out on #peer-support if the #homework-help queue is lengthy. If someone helps you out, give them a shoutout using our handy-dandy form!

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.

```
const inputArr = [5,1,3,4,2];

/*Borrowed code for bubbleSort starts*/
let bubbleSort = (inputArr) => {
    let len = inputArr.length;
    for (let i = 0; i < len; i++) {
        for (let j = 0; j < len; j++) {
            if (inputArr[j] > inputArr[j + 1]) {
                let tmp = inputArr[j];
                inputArr[j] = inputArr[j + 1];
                inputArr[j + 1] = tmp;
            }
        }
    }
    return inputArr;
};

/*Borrowed code from bubbleSort ends*/

//Source: bubbleSort function obtained from https://medium.com/javascript-algorithms/javascript-algorithms-bubble-sort-3d27f285c3b2
//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
//How it works: I read the following article to understand how bubble sorts work (http://www.pkirs.utep.edu/CIS3355/Tutorials/chapter9/tutorial9A/bubblesort.htm)
//Demonstration of understanding:
//Example array: [3,1,2]
//Step 1: Compare 3 and 1. Since 1 is smaller, swap places.
//Array: [1,3,2]
//Step 2: Compare 3 and 2. Since 2 is smaller, swap places.
//Array: [1,2,3]
//Step 3: Compare 1 and 2. No need to swap.
//Array: [1,2,3]
//Step 4: Compare 2 and 3. No need to swap.
//Array: [1,2,3]
//Function complete.
console.log(bubbleSort(inputArr));
```