ASP.NET Core Practice - Inventory System Day 1

Deadline: Monday, September 28th 2020, 09:00 AM

GitHub Classroom Link

Introduction

This practice is meant to challenge your mastery of ASP.NET and how well you are able to use ASP.NET Web API to create a CRUD application. Your goal in this practice is to create an inventory management system which helps a company keep track of goods and supplies. As goods and supplies are restocked, sold, or used, your application should update the data appropriately to reflect the changes. This is a cumulative activity. You will build on this practice for your next practice.

Requirements (See Rubric for Details)

- ❑ Product Class:
  - ❑ Product must have unique product "ID"
  - ❑ Product must have a product "Name"
  - ❑ Product must have a "Quantity" that is greater than or equal to zero
  - ❑ Product must have an "IsDiscontinued" boolean
    - ❑ Set to false by default
- ❑ Admin Controller:
  - ❑ Create an HttpPost "AddProduct" endpoint that allows the user to add a product to the database
  - ❑ Create an HttpPut "DiscontinueProduct" endpoint that allows the user to discontinue a product
    - ❑ Accepts the product ID as a parameter
      - ❑ If Product ID is invalid - respond with Http Status 403 and include a descriptive message in the Content
    - ❑ Set IsDiscontinued to true.
  - ❑ Create an HttpPut "AddQuantityProduct" endpoint that allows the user to add to a product's quantity
    - ❑ Accepts the product ID as a parameter
      - ❑ If Product ID is invalid - respond with Http Status 403 and include a descriptive message in the Content
    - ❑ AmountAdded must be a positive integer
      - ❑ If integer invalid - respond with Http Status 403 and include a descriptive message in the Content
    - ❑ User cannot add to a product that has been discontinued
      - ❑ If Product is discontinued - respond with Http Status 403 and include a descriptive message in the Content
  - ❑ Create an HttpPut "SubtractQuantityProduct" endpoint that allows the user to subtract from a product's quantity
    - ❑ Accepts the product ID as a parameter
      - ❑ If Product ID is invalid - respond with Http Status 403 and include a descriptive message in the Content
    - ❑ AmountSubtracted must be less than or equal to the current quantity and greater than zero

            - ❏ If user attempts to subtract more than is in stock, reject the entire transaction, respond with Http Status 403 and include a descriptive message in the Content
        - ❏ User **can** subtract from a product that has been discontinued provided other rules are also followed
    - ❏ Create an HttpGet "ShowInventory" endpoint that displays the entire inventory
        - ❏ Requires no parameters
        - ❏ This endpoint will not return products that have been discontinued
        - ❏ Order by "Quantity" from lowest to highest so that user will know what needs to be restocked
- ❏ Use Postman to test each endpoint
    - ❏ Take screenshots of Postman receiving the response from "AddProduct"
    - ❏ Take screenshots of Postman receiving the response from "DiscontinueProduct"
    - ❏ Take screenshots of Postman receiving the response from "AddQuantityProduct"
    - ❏ Take screenshots of Postman receiving the response from "SubtractQuantityProduct"
    - ❏ Take screenshots of Postman receiving the response from "ShowInventory"
- ❏ All information must be stored in a database

Challenges (See Rubric for Details)

- ❏ Make it look nice with some CSS!
- ❏ Have an unexpected feature!

Hints

- Focus on the requirements first, challenges are extra!
- You need to write a table to keep track of the unique product id, product name, and product quantity
- You must use a database to store all the information submitted in the "Admin Controller" page and you must use an API to fetch all the information on your "Inventory" page
- This 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

Rubric

Requirement	**Points**
MANDATORY: ● Product class ○ Has unique product code ○ Has a name ○ Has a quantity that is greater than or equal to zero ○ Has an IsDiscontinued boolean set to false by default	4
● Admin Controller: ○ "AddProduct" ○ "DiscontinueProduct" ▪ Error 403 if Product ID is invalid ● Error has descriptive message in Content ○ "AddQuantityProduct" ▪ Error 403 if Product ID is invalid ● Error has descriptive message in Content ▪ Error 403 if Product is discontinued	19

● Error has descriptive message in Content ▪ Error 403 if amount added is invalid ● Error has descriptive message in Content ○ "SubtractQuantityProduct" ▪ Error 403 if Product ID is invalid ● Error has descriptive message in Content ▪ Error 403 if amount subtracted is invalid ● Error has descriptive message in Content ○ "ShowInventory" ▪ Order product by quantity from lowest to highest ▪ Does not return discontinued items	
● Postman ○ Screenshots of Postman receiving the response from "AddProduct" ○ Screenshots of Postman receiving the response from "DiscontinueProduct" ○ Screenshots of Postman receiving the response from "AddQuantityProduct" ○ Screenshots of Postman receiving the response from "SubtractQuantityProduct" ○ Screenshots of Postman receiving the response from "ShowInventory"	5
● Database contains all information	1
CHALLENGE: ● Make it look nice with some CSS! ● Have an unexpected feature!	2
Total:	31

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));
```