Coding test .NET

Please write a small program for each of the exercises

**1. Denomination routine**

An ATM has three cartridges for different denominations:

* 10 EUR cartridge
* 50 EUR cartridge
* 100 EUR cartridge

Now we want to pay out following amounts from the ATM:

* 30 EUR
* 50 EUR
* 60 EUR
* 80 EUR
* 140 EUR
* 230 EUR
* 370 EUR
* 610 EUR
* 980 EUR

Write a program which will calculate for each payout the possible combinations which the ATM can pay out.

For example, for 100 EUR the available payout denominations would be:

* 10 x 10 EUR
* 1 x 50 EUR + 5 x 10 EUR
* 2 x 50 EUR
* 1 x 100 EUR

using System;

using System.Collections.Generic;

class Program

{

    static int[] denominations = { 10, 50, 100 };

    static Dictionary<int, int[]> combinations = new Dictionary<int, int[]>();

    static void Main()

    {

        int[] payouts = { 30, 50, 60, 80, 140, 230, 370, 610, 980 };

        foreach (int payout in payouts)

        {

            Console.WriteLine($"Possible combinations for {payout} EUR:");

            combinations.Clear();

            FindCombinations(payout, 0, new int[denominations.Length]);

            PrintCombinations();

            Console.WriteLine();

        }

    }

    static void FindCombinations(int remainingAmount, int index, int[] currentCount)

    {

        if (remainingAmount == 0)

        {

            int[] countCopy = new int[currentCount.Length];

            Array.Copy(currentCount, countCopy, currentCount.Length);

            combinations.Add(combinations.Count + 1, countCopy);

            return;

        }

        if (index >= denominations.Length || remainingAmount < 0)

            return;

        int denomination = denominations[index];

        int maxCount = remainingAmount / denomination;

        for (int count = maxCount; count >= 0; count--)

        {

            currentCount[index] = count;

            FindCombinations(remainingAmount - (count \* denomination), index + 1, currentCount);

        }

        currentCount[index] = 0;

    }

    static void PrintCombinations()

    {

        foreach (var combination in combinations)

        {

            int[] count = combination.Value;

            Console.Write($"{combination.Key}: ");

            for (int i = 0; i < count.Length; i++)

            {

                if (count[i] > 0)

                {

                    Console.Write($"{denominations[i]} EUR x {count[i]}");

                    if (i < count.Length - 1)

                        Console.Write(", ");

                }

            }

            Console.WriteLine();

        }

    }

}

**2. REST server**

A small REST server with good performance for simple customer management has two functions:

* POST customers

Request:

[

{

firstName: 'Aaaa',

lastName: 'Bbbb',

age: 20,

id: 5

},

{

firstName: 'Bbbb',

lastName: 'Cccc',

age: 24,

id: 6

}

]

Multiple customers can be sent in one request.

The server validates every customer of the request:

* checks that every field is supplied
* validates that the age is above 18
* validates that the ID has not been used before

The server then adds each customer as an object to an internal array – the customers will not be appended to the array but instead it will be inserted at a position so that the customers are sorted by last name and then first name WITHOUT using any available sorting functionality (an example for the inserting is in the Appendix).

The server also persists the array so it will be still available after a restart of the server.

* GET customers

Returns the array of customers with all fields

Write the server and a small simulator which can send several requests for POST customers and GET customers in parallel to the server.

For that program it is not allowed to use any sorting mechanism like array.sort().

The simulated POST customers requests have following requirements:

* Each request should contain at least 2 different customers
* Age should be randomized between 10 and 90
* ID should be increasing sequentially.
* The first names and last names of the Appendix should be used in random combinations

**Appendix:**

**Data:**

First names: Last names:

Leia Liberty

Sadie Ray

Jose Harrison

Sara Ronan

Frank Drew

Dewey Powell

Tomas Larsen

Joel Chan

Lukas Anderson

Carlos Lane

**Example for the inserting mechanism:**

Array in server:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

Request POST customers:

[{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 }]

Array after insert:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

Request POST customers:

[{ lastName: 'Bbbb', firstName: 'Aaaa', age: 28, id: 7 }]

Array after insert:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Bbbb', firstName: 'Aaaa', age: 28, id: 7 },

{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

using System;

using System.Collections.Generic;

using System.IO;

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

public class Customer

{

    public string FirstName { get; set; }

    public string LastName { get; set; }

    public int Age { get; set; }

    public int Id { get; set; }

}

public class CustomersServer

{

    private List<Customer> customers;

    public CustomersServer()

    {

        customers = LoadCustomersFromFile();

    }

    public async Task StartAsync()

    {

        var host = new WebHostBuilder()

            .UseKestrel()

            .Configure(app =>

            {

                app.UseRouting();

                app.UseEndpoints(endpoints =>

                {

                    endpoints.MapPost("/customers", HandlePostCustomers);

                    endpoints.MapGet("/customers", HandleGetCustomers);

                });

            })

            .Build();

        await host.RunAsync();

    }

    private async Task HandlePostCustomers(HttpContext context)

    {

        try

        {

            var requestBody = await new StreamReader(context.Request.Body).ReadToEndAsync();

            var receivedCustomers = JsonConvert.DeserializeObject<List<Customer>>(requestBody);

            if (receivedCustomers == null)

            {

                context.Response.StatusCode = (int)HttpStatusCode.BadRequest;

                return;

            }

            var invalidCustomers = new List<Customer>();

            foreach (var customer in receivedCustomers)

            {

                if (string.IsNullOrEmpty(customer.FirstName) || string.IsNullOrEmpty(customer.LastName)

                    || customer.Age <= 18 || IsIdUsed(customer.Id))

                {

                    invalidCustomers.Add(customer);

                }

            }

            if (invalidCustomers.Count > 0)

            {

                context.Response.StatusCode = (int)HttpStatusCode.BadRequest;

                await context.Response.WriteAsync(JsonConvert.SerializeObject(invalidCustomers));

                return;

            }

            foreach (var customer in receivedCustomers)

            {

                        while (index < customers.Count)

)

        {

            index++;

        }

customers.Insert(index, customer);

            }

            SaveCustomersToFile();

            context.Response.StatusCode = (int)HttpStatusCode.OK;

        }

        catch

        {

            context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

        }

    }

    private async Task HandleGetCustomers(HttpContext context)

    {

        try

        {

            context.Response.StatusCode = (int)HttpStatusCode.OK;

            await context.Response.WriteAsync(JsonConvert.SerializeObject(customers));

        }

        catch

        {

            context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

        }

    }

    private bool IsIdUsed(int id)

    {

        foreach (var customer in customers)

        {

            if (customer.Id == id)

            {

                return true;

            }

        }

        return false;

    }

    private List<Customer> LoadCustomersFromFile()

    {

        // Load customers from a file (e.g., JSON, database) and return the list

        // If no persisted data is available, return an empty list

        // Example implementation:

        if (File.Exists("customers.json"))

        {

            var customersJson = File.ReadAllText("customers.json");

            return JsonConvert.DeserializeObject<List<Customer>>(customersJson);

        }

        return new List<Customer>();

    }

    private void SaveCustomersToFile()

    {

        var customersJson = JsonConvert.SerializeObject(customers);

        File.WriteAllText("customers.json", customersJson);

    }

}

public class Program

{

    public static async Task Main(string[] args)

    {

        var server = new CustomersServer();

        await server.StartAsync();

    }

}