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

**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 },

]

Atm :

List<int> notes = new List<int>();

List<int> amounts = new List<int>() { 50, 10, 100 };

Change(notes, amounts, 0, 0, 100);

static void Change(List<int> notes, List<int> amounts, int highest, int sum, int withdraw)

{

if (sum == withdraw)

{

show(notes, amounts);

return;

}

if (sum > withdraw)

{

return;

}

foreach (int value in amounts)

{

if (value >= highest)

{

List<int> copy = new List<int>(notes);

copy.Add(value);

Change(copy, amounts, value, sum + value, withdraw);

}

}

}

static void show(List<int> notes, List<int> amounts)

{

foreach (int amount in amounts)

{

int count = notes.Count(value => value == amount);

Console.WriteLine("{0}: {1}", amount, count);

}

Console.WriteLine();

}

Server:

namespace epay.Server.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class epayController : ControllerBase

{

[HttpPost("InsertCustomer")]

public async Task<ActionResult<object>> InsertCustomer(List<Customer> customers)

{

try

{

var errors = new List<Error>();

foreach (var customer in customers)

customer.Validate(ref errors);

if (errors.Count > 0)

return Ok(errors);

DataContext.WriteData(customers);

return Ok(new ApiResult

{

IsSuccess = true,

MetaData = new MetaData

{

Message = "ok",

AppStatusCode = AppStatusCode.Success

}

});

}

catch (Exception ex)

{

return BadRequest(new ApiResult

{

IsSuccess = false,

MetaData = new MetaData

{

Message = ex.Message,

AppStatusCode = AppStatusCode.LogicError

}

});

}

}

[HttpGet("GetCustomer")]

public async Task<ActionResult<object>> GetCustomer()

{

try

{

var AllCustomers = await DataContext.ReadData();

return Ok(new ApiResult<List<Customer>>

{

Data = AllCustomers,

IsSuccess = true,

MetaData = new MetaData

{

AppStatusCode = AppStatusCode.Success,

Message = "ok"

}

});

}

catch (Exception ex)

{

return BadRequest(new ApiResult

{

IsSuccess = false,

MetaData = new MetaData

{

Message = ex.Message,

AppStatusCode = AppStatusCode.LogicError

}

});

}

}

}

}

namespace epay.Server.Models

{

public class ApiResult

{

public bool IsSuccess { get; set; }

public MetaData MetaData { get; set; }

}

public class ApiResult<TData>

{

public bool IsSuccess { get; set; }

public TData Data { get; set; } = default(TData);

public MetaData MetaData { get; set; }

}

public class MetaData

{

public string Message { get; set; }

public AppStatusCode AppStatusCode { get; set; }

}

public enum AppStatusCode

{

Success = 1,

NotFound = 2,

BadRequest = 3,

LogicError = 4,

UnAuthorize = 5,

ServerError

}

}

using System.Runtime.Serialization;

namespace epay.Server.Models

{

public class Error

{

public string ErrorDescription { get; set; }

public string ReferenceName { get; set; }

public string OriginalValue { get; set; }

public string ExtraData { get; set; }

}

public enum Errordescription

{

NotNull = 1,

NotFound = 2,

BadRequest = 3,

LogicError = 4,

UnAuthorize = 5,

ServerError

}

}

using epay.Server.data;

using epay.Server.Models;

using epay.Server.Utils;

using Microsoft.AspNetCore.Mvc.RazorPages;

namespace epay.Server.Validation

{

public static class CustomerValidation

{

public static bool Validate(this Customer customer, ref List<Error> errors)

{

#region firstName Validation

if (string.IsNullOrWhiteSpace(customer.firstName))

errors.Add(

new Error

{

ErrorDescription = "mandatory fields",

ReferenceName = $"firstName , id: {customer.id}"

});

else if (customer.firstName.HasSpecial())

{

errors.Add(

new Error

{

ErrorDescription = "you are not allowed to use special character like ?,$,..",

ReferenceName = $"firstName , id: {customer.id}" ,

OriginalValue = customer.firstName

});

}

#endregion

#region lastname Validation

if (string.IsNullOrWhiteSpace(customer.lastName))

errors.Add(

new Error

{

ErrorDescription = "mandatory fields",

ReferenceName = $"lastName , id: {customer.id}"

});

else if (customer.lastName.HasSpecial())

{

errors.Add(

new Error

{

ErrorDescription = "you are not allowed to use special character like ?,$,..",

ReferenceName = $"lastName , id: {customer.id}",

OriginalValue = customer.lastName

});

}

#endregion

#region id Validation

if (string.IsNullOrWhiteSpace(customer.id.ToString()))

errors.Add(

new Error

{

ErrorDescription = "mandatory fields",

ReferenceName = "id"

});

if (DataContext.CheckDuplicateID(customer.id).Result)

errors.Add(

new Error

{

ErrorDescription = "duplicate",

ReferenceName = "id",

OriginalValue=customer.id.ToString()

});

#endregion

#region age Validation

if (customer.age == 0)

errors.Add(

new Error

{

ErrorDescription = "mandatory fields",

ReferenceName = "age"

});

else if (customer.age <= 18)

errors.Add(

new Error

{

ErrorDescription = "you must be more than 18 years old",

ReferenceName = $"age , id: {customer.id} ",

OriginalValue = customer.age.ToString()

});

#endregion

return !errors.Any();

}

}

}

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

using System.Text.RegularExpressions;

namespace epay.Server.Utils

{

public static class Extension

{

public static bool HasSpecial(this string code)

=> !string.IsNullOrWhiteSpace(code) && !Regex.IsMatch(code, "^[A-Za-z0-9]\*$");

public static bool IsValidAge(this int age)

=> age > 18 ;

}

}

using epay.Server.Models;

using Newtonsoft.Json;

using static System.Net.Mime.MediaTypeNames;

namespace epay.Server.data

{

public static class DataContext

{

public static async Task<List<Customer>> ReadData()

{

List<Customer> customers = new List<Customer>();

using (StreamReader r = new StreamReader(@"data/Customers.json"))

{

string json = await r.ReadToEndAsync();

customers = JsonConvert.DeserializeObject<List<Customer>>(json);

}

return customers;

}

public static async void WriteData(List<Customer> customers)

{

List<Customer> oldCustmerlst = new List<Customer>();

using (StreamReader r = new StreamReader(@"data/Customers.json"))

{

string json = await r.ReadToEndAsync();

oldCustmerlst = JsonConvert.DeserializeObject<List<Customer>>(json);

}

oldCustmerlst.AddRange(customers);

BubblesortBy(x => x.lastName, oldCustmerlst);

BubblesortBy(x => x.firstName, oldCustmerlst);

//var newCustomerlst = oldCustmerlst.OrderBy(x => x.lastName).ThenBy(x => x.firstName).ToList();

string json2 = JsonConvert.SerializeObject(oldCustmerlst);

File.WriteAllText(@"data/Customers.json", json2);

}

public static async Task<bool> CheckDuplicateID(int id)

{

var customerIds = ReadData().Result.Select(x => x.id).ToList();

return customerIds.Any(x => x == id);

}

public static void BubblesortBy<TSource, TKey>(Func<TSource, TKey> keySelector,

List<TSource> stocks)

{

int loopCount = 0;

bool doBreak = true;

for (int i = 0; i < stocks.Count; i++)

{

doBreak = true;

for (int j = 0; j < stocks.Count - 1; j++)

{

if (Compare(keySelector(stocks[j]), keySelector(stocks[j + 1])))

{

TSource temp = stocks[j + 1];

stocks[j + 1] = stocks[j];

stocks[j] = temp;

doBreak = false;

}

loopCount++;

}

if (doBreak) { break; }

}

}

private static bool Compare<T>(T l, T r)

{

return Comparer<T>.Default.Compare(l, r) > 0;

}

}

}