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 },

]

Test results:

**1. Denomination routine:**

Project Structure in the solution



namespace DenominationRoutine

{

internal class Program

{

private static string currency = "EUR";

static void Main()

{

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

Console.WriteLine("Please enter valid amount with multiples of 10 (eg: 10, 50, 100):");

if (int.TryParse(Console.ReadLine(), out int x))

{

var possibleDenominations = CalculatePossibleDenominations(x, atmDenominations);

PrintAllDenominations(x, possibleDenominations);

}

else

Console.WriteLine($"The entered value is invalid. Please enter valid amount with multiples of 10 (eg: 10, 50, 100)\n");

Console.ReadLine();

}

static List<List<int>>? CalculatePossibleDenominations(int amountToBeWithdrawn, List<int> atmDenominations)

{

int minimumCartdidgeType = atmDenominations.Min();

if (amountToBeWithdrawn <= 0 || amountToBeWithdrawn < minimumCartdidgeType)

{

Console.WriteLine($"The amount '{amountToBeWithdrawn} {currency}' is invalid. The minimum possible amount and denomination which can be dispensed is {minimumCartdidgeType}.\n");

return null;

}

var result = new List<List<int>>();

CalculatePossibleRecursiveDenominations(result, atmDenominations, amountToBeWithdrawn, 0, new List<int>());

return result;

}

static void CalculatePossibleRecursiveDenominations(List<List<int>> allDenominations, List<int> cartridgeTypes, int amountToBeDispensed, int index, List<int> currentDenomination)

{

if (amountToBeDispensed is 0)

{

allDenominations.Add(new List<int>(currentDenomination));

return;

}

for (int i = index; i < cartridgeTypes.Count; i++)

{

if ((amountToBeDispensed - cartridgeTypes[i]) >= 0)

{

currentDenomination.Add(cartridgeTypes[i]);

CalculatePossibleRecursiveDenominations(allDenominations, cartridgeTypes, amountToBeDispensed - cartridgeTypes[i], i, currentDenomination);

currentDenomination.Remove(cartridgeTypes[i]);

}

}

}

static void PrintAllDenominations(int amount, List<List<int>> possibleDenominations)

{

if (possibleDenominations is not null)

{

Console.WriteLine($"The amount '{amount} {currency}' can be dispensed in following possible ways:");

Parallel.ForEach(possibleDenominations, combination =>

{

var groupedDenominations = combination.GroupBy(x => x)

.Select(x => new { Note = x.Key, Count = x.Count() })

.ToList();

var res = string.Join(" + ", groupedDenominations.Select(x => $"{x.Count} X {x.Note} EUR"));

Console.WriteLine(res);

});

Console.WriteLine();

}

}

}

}

**2. Rest Server:**

Project structure in the solution   


using CustomerManagement.Common.Models;

using CustomerManagement.Infrastructure.Services.Customer;

using Microsoft.AspNetCore.Mvc;

namespace CustomerManagement.Controllers

{

[ApiController]

[Route("[controller]")]

public class CustomerController : ControllerBase

{

private readonly ILogger<CustomerController> \_logger;

private readonly ICustomerService \_customerService;

public CustomerController(ILogger<CustomerController> logger, ICustomerService customerService)

{

\_logger = logger;

\_customerService = customerService;

}

[HttpPost()]

public async Task<ActionResult> Post([FromBody] List<CustomerModel> model)

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

var res = await \_customerService.Post(model);

return Ok(res);

}

[HttpGet()]

public async Task<ActionResult> Get()

{

var res = \_customerService.GetAll(null).Result;

return Ok(res);

}

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

using Newtonsoft.Json;

namespace CustomerManagement.Infrastructure.Repositories.Customer

{

public class CustomerRepository : ICustomerRepository

{

private readonly string \_filePath;

private readonly ILogger<CustomerRepository> \_logger;

public CustomerRepository(ILogger<CustomerRepository> logger)

{

\_filePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "ePayCustomerData.json");

\_logger = logger;

}

SortedList<string, CustomerModel>? IBaseRepository<SortedList<string, CustomerModel>>.Read

{

get

{

try

{

return JsonConvert.DeserializeObject<SortedList<string, CustomerModel>>(File.ReadAllText(\_filePath));

}

catch (Exception)

{

\_logger.LogError($"Unable to read JSON file from {\_filePath}");

}

return null;

}

}

public void Write(SortedList<string, CustomerModel> data)

{

File.WriteAllText(\_filePath, JsonConvert.SerializeObject(data));

}

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

namespace CustomerManagement.Infrastructure.Repositories.Customer

{

public interface ICustomerRepository : IBaseRepository<SortedList<string, CustomerModel>>

{

}

}

----------------------------------Next file----------------------------------------

namespace CustomerManagement.Infrastructure.Repositories

{

public interface IBaseRepository<T>

{

T Read { get; }

void Write(T data);

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

using CustomerManagement.Infrastructure.Repositories.Customer;

namespace CustomerManagement.Infrastructure.Services.Customer

{

public class CustomerService : ICustomerService, IBaseService<CustomerModel>

{

private readonly ICustomerRepository \_customerRepository;

private readonly ILogger<CustomerModel> \_logger;

private static SortedList<string, CustomerModel> \_customerList = new SortedList<string, CustomerModel>();

private static bool \_isInitialized = false;

object obj = new();

public CustomerService(ILogger<CustomerModel> logger, ICustomerRepository customerRepository)

{

\_customerRepository = customerRepository;

\_logger = logger;

if (!\_isInitialized)

{

var previousData = \_customerRepository.Read;

if (previousData != null) //Only look for past data just once during lifetime of server running

{

\_customerList = previousData;

}

\_isInitialized = true;

}

}

public async Task<bool> Delete(int id)

{

throw new NotImplementedException();

}

public async Task<List<CustomerModel>> GetAll(int? parentId)

{

return \_customerList.Values.ToList();

}

public async Task<CustomerModel> GetById(int resourceId)

{

throw new NotImplementedException();

}

public async Task<bool> Post(CustomerModel entity)

{

throw new NotImplementedException();

}

public async Task<bool> Post(List<CustomerModel> entities)

{

lock (obj)

{

Parallel.ForEach(entities, entity =>

{

if (IsUnique(entity))

{

\_customerList.Add($"{entity.LastName}{entity.FirstName}", entity);

}

else

{

\_logger.LogWarning($"Unable to add record with Id : {entity.Id} because it already exists in the system");

}

});

\_customerRepository.Write(\_customerList);

}

return true;

}

public async Task<CustomerModel> Put(int id, CustomerModel entity)

{

throw new NotImplementedException();

}

private bool IsUnique(CustomerModel model)

{

bool isUnique = true;

foreach (var key in \_customerList.Keys)

{

CustomerModel item;

if (\_customerList.TryGetValue(key, out item))

{

if (item.Id == model.Id)

{

isUnique = false;

break;

}

}

}

return isUnique;

}

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

namespace CustomerManagement.Infrastructure.Services.Customer

{

public interface ICustomerService : IBaseService<CustomerModel>

{

public Task<bool> Post(List<CustomerModel> entities);

}

}

----------------------------------Next file----------------------------------------

namespace CustomerManagement.Infrastructure.Services

{

public interface IBaseService<TEntity>

{

public Task<List<TEntity>> GetAll(int? parentId);

public Task<TEntity> GetById(int resourceId);

public Task<bool> Post(TEntity entity);

public Task<bool> Delete(int id);

public Task<TEntity> Put(int id, TEntity entity);

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Infrastructure.Repositories.Customer;

using CustomerManagement.Infrastructure.Services.Customer;

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();

builder.Services.AddHttpContextAccessor();

builder.Services.AddSingleton<ICustomerService, CustomerService>();

builder.Services.AddSingleton<ICustomerRepository, CustomerRepository>();

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();

----------------------------------Next file----------------------------------------

using System.ComponentModel.DataAnnotations;

namespace CustomerManagement.Common.CustomAttributes

{

public class MaxValueAttribute : ValidationAttribute

{

private readonly int \_maxValue;

public MaxValueAttribute(int maxValue)

{

\_maxValue = maxValue;

}

public override bool IsValid(object value) => (int)value <= \_maxValue;

}

public class MinValueAttribute : ValidationAttribute

{

private readonly int \_minValue;

public MinValueAttribute(int minValue)

{

\_minValue = minValue;

}

public override bool IsValid(object value) => (int)value >= \_minValue;

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.CustomAttributes;

using System.ComponentModel.DataAnnotations;

namespace CustomerManagement.Common.Models

{

[Serializable]

public class BaseModel

{

[Required]

[MinValue(1)]

public int Id { get; set; }

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.CustomAttributes;

using System.ComponentModel.DataAnnotations;

namespace CustomerManagement.Common.Models

{

[Serializable]

public class CustomerModel : BaseModel

{

[Required]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

[Required]

[MinValue(19)]

public int Age { get; set; }

public CustomerModel(string firstName, string lastName, int age)

{

FirstName = firstName;

LastName = lastName;

Age = age;

}

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

namespace CustomerManagement.Test.Helpers

{

public static class DataGenerator

{

public static List<string> GenerateLastName()

{

List<string> lastNames = new()

{

"Liberty"

,"Ray"

,"Harrison"

,"Ronan"

,"Drew"

,"Powell"

,"Larsen"

,"Chan"

,"Anderson"

,"Lane"

};

return lastNames;

}

public static List<string> GenerateFirstName()

{

List<string> firstNames = new()

{

"Leia"

,"Sadie"

,"Jose"

,"Sara"

,"Frank"

,"Dewey"

,"Tomas"

,"Joel"

,"Lukas"

,"Carlos"

};

return firstNames;

}

public static List<CustomerModel> GenerateRandomCustomerObjects(List<string> firstNames, List<string> lastNames)

{

List<CustomerModel> lstCustomers = new();

List<int> randomIndecesFirstName = new();

List<int> randomIndecesLastName = new();

Random rnd = new();

while (randomIndecesFirstName.Count != 10)

{

int num = rnd.Next(1, 11);

if (!randomIndecesFirstName.Contains(num))

{

randomIndecesFirstName.Add(num);

}

}

while (randomIndecesLastName.Count != 10)

{

int num = rnd.Next(1, 11);

if (!randomIndecesLastName.Contains(num))

{

randomIndecesLastName.Add(num);

}

}

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

{

CustomerModel model = new(firstNames[i], lastNames[i], rnd.Next(19, 50));

model.Id = i + 1;

lstCustomers.Add(model);

}

return lstCustomers;

}

}

}

----------------------------------Next file----------------------------------------

using CustomerManagement.Common.Models;

using CustomerManagement.Test.Helpers;

using Newtonsoft.Json;

using RestSharp;

using Xunit;

namespace CustomerManagement.Test

{

public class CustomerControllerTest

{

private readonly string \_apiBaseUrl = "<https://localhost:7165/>";

[Fact]

public async void Post\_Parallel\_Customers\_And\_Verify\_By\_Getting\_Test()

{

var customers = DataGenerator.GenerateRandomCustomerObjects(DataGenerator.GenerateFirstName(), DataGenerator.GenerateLastName());

var responses = new List<Task<RestResponse>>();

var multipleCustomersInParallel = new List<List<CustomerModel>>

{

customers.OrderBy(x => x.Id).Skip(0).Take(2).ToList(),

customers.OrderBy(x => x.Id).Skip(2).Take(2).ToList(),

customers.OrderBy(x => x.Id).Skip(4).Take(4).ToList(),

customers.OrderBy(x => x.Id).Skip(8).Take(2).ToList()

};

Parallel.ForEach(multipleCustomersInParallel, c =>

{

RestClient restClient = new(\_apiBaseUrl);

RestRequest request = new("Customer", Method.Post);

request.AddBody(c);

responses.Add(restClient.ExecuteAsync(request));

});

var res = await Task.WhenAll(responses);

RestClient getRestClient = new(\_apiBaseUrl);

RestRequest getRequest = new("Customer", Method.Get);

var allRecords = getRestClient.Execute(getRequest);

var result = allRecords.Content;

var customersReturnedByServer = JsonConvert.DeserializeObject<List<CustomerModel>>(result);

customers = customers

.Where(x => x.Id > 0 && x.Age > 18)

.OrderBy(x => x.LastName)

.ThenBy(x => x.FirstName)

.ToList();

if (customersReturnedByServer != null && customersReturnedByServer.Count > 0)

{

bool sequenceMatched = true;

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

{

if (customers[i].Id != customersReturnedByServer[i].Id)

{

sequenceMatched = false;

break;

}

}

Assert.True(sequenceMatched);

}

else

{

Assert.True(false);

}

}

}

}

----------------------------------End----------------------------------------