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 },

]

**Denomination project**

static void Main(string[] args)

{

int[] DenominationArray = { 10, 50, 100 };

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

foreach (var targetSum in targetSumArray)

{

List<List<int>> answerList = combinationSum(DenominationArray.ToList(), targetSum);

Console.WriteLine("Start possible combinations for: " + targetSum);

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

{

var combination = answerList[i].Select(s => s.ToString()).Aggregate((p, q) => p + ',' + q);

Console.WriteLine(combination);

}

Console.WriteLine("End Combination for:" + targetSum);

Console.ReadKey();

}

}

public static List<List<int>> combinationSum(List<int> inputArray, int targetSum)

{

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

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

findCombination(answerList, inputArray, targetSum, 0, tempAnswer);

return answerList;

}

static void findCombination(List<List<int>> answerList, List<int> inputList, int targetSum, int index, List<int> tempAnswer)

{

if (targetSum == 0)

{

answerList.Add(new List<int>(tempAnswer));

return;

}

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

{

var leftSum = targetSum - inputList[i];

if (leftSum >= 0)

{

tempAnswer.Add(inputList[i]);

findCombination(answerList, inputList, leftSum, i, tempAnswer);

tempAnswer.Remove(inputList[i]);

}

}

}

**RESTserver project**

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddSingleton<CustomerContext>();

}

public class Customer

{

[Required]

public string firstName { get; set; }

[Required]

public string lastName { get; set; }

[Required]

[Range (19,int.MaxValue, ErrorMessage = "Age can be above 18")]

public int age { get; set; }

[Required]

public int id { get; set; }

public string getFullName()

{

return lastName + firstName;

}

public override string ToString()

{

return id + " " + firstName + " " + lastName;

}

}

public class Node

{

public Customer customerData;

public Node next;

public Node(Customer cus)

{

customerData = cus;

next = null;

}

}

public class CustomerContext

{

private Object thisLock = new Object();

public Node head;

public void sortedInsert(Node new\_node)

{

lock (thisLock)

{

Node current;

if (head == null || head.customerData.getFullName().CompareTo(new\_node.customerData.getFullName()) >= 0)

{

new\_node.next = head;

head = new\_node;

}

else

{

current = head;

while (current.next != null && current.next.customerData.getFullName()

.CompareTo(new\_node.customerData.getFullName()) < 0)

current = current.next;

new\_node.next = current.next;

current.next = new\_node;

}

}

}

public List<int> getIdList()

{

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

Node temp = head;

while (temp != null)

{

idList.Add(temp.customerData.id);

temp = temp.next;

}

return idList;

}

public List<Customer> getCustomerList()

{

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

Node temp = head;

while (temp != null)

{

customers.Add(temp.customerData);

temp = temp.next;

}

return customers;

}

}

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

[ApiController]

public class CustomerController : Controller

{

private readonly CustomerContext context;

public CustomerController(CustomerContext context)

{

this.context = context;

}

[HttpPost]

public async Task<IActionResult> Post([FromBody] IEnumerable<Models.Customer> customerList)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

Node new\_node;

foreach (Customer customer in customerList)

{

if (context.getIdList().Contains(customer.id))

return BadRequest("Error: Id " + customer.id + " has been used before");

new\_node = new Node(customer);

context.sortedInsert(new\_node);

}

return Ok(customerList);

}

[HttpGet]

public IEnumerable<Models.Customer> Get()

{

return context.getCustomerList();

}

}

}

**RESTclient project**

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 ConsumeWebApi

{

HttpClient client = new HttpClient();

private List<Customer> CustomerStore()

{

Random r = new Random();

int minAge = 10;

int maxAge = 90;

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

customerList.Add(new Customer() { firstName = "Leia", lastName = "Liberty", age = r.Next(minAge, maxAge), id = 25 });

customerList.Add(new Customer() { firstName = "Sadie", lastName = "Ray", age = r.Next(minAge, maxAge), id = 26 });

customerList.Add(new Customer() { firstName = "Jose", lastName = "Harrison", age = r.Next(minAge, maxAge), id = 27 });

customerList.Add(new Customer() { firstName = "Sara", lastName = "Ronan", age = r.Next(minAge, maxAge), id = 28 });

customerList.Add(new Customer() { firstName = "Frank", lastName = "Drew", age = r.Next(minAge, maxAge), id = 29 });

customerList.Add(new Customer() { firstName = "Dewey", lastName = "Powell", age = r.Next(minAge, maxAge), id = 30 });

customerList.Add(new Customer() { firstName = "Tomas", lastName = "Larsen", age = r.Next(minAge, maxAge), id = 31 });

customerList.Add(new Customer() { firstName = "Joel", lastName = "Chan", age = r.Next(minAge, maxAge), id = 32 });

customerList.Add(new Customer() { firstName = "Lukas", lastName = "Anderson", age = r.Next(minAge, maxAge), id = 33 });

customerList.Add(new Customer() { firstName = "Carlos", lastName = "Lane", age = r.Next(minAge, maxAge), id = 34 });

return customerList;

}

public async Task<IEnumerable<Customer>> PostCustomer(IEnumerable<Customer> customerList)

{

var response = await client

.PostAsync(

"https://localhost:44324/api/customer",

new StringContent(JsonSerializer.Serialize(customerList), Encoding.UTF8, "application/json"))

.ConfigureAwait(false);

var users = JsonSerializer.Deserialize<IEnumerable<Customer>>(await response.Content.ReadAsStringAsync());

return users;

}

public async Task<IEnumerable<Customer>> PostCustomersInParallel()

{

var tasks = new List<Task<IEnumerable<Customer>>>();

var customerList1 = CustomerStore().Take(2);

var customerList2 = CustomerStore().Skip(2).Take(3);

var customerList3 = CustomerStore().Skip(5).Take(5);

tasks.Add(PostCustomer(customerList1));

tasks.Add(PostCustomer(customerList2));

tasks.Add(PostCustomer(customerList3));

return (await Task.WhenAll(tasks)).SelectMany(u => u);

}

}

ConsumeWebApi consumeWebApi = new ConsumeWebApi();

await consumeWebApi.PostCustomersInParallel();