**Validate a Card number**

**4477 4683 4311 3002**

**2003 1134 3864 7744- Reverse the number**

**2+0\*2+0+3\*2+1+1\*2+3+4\*2+3+8\*2+6+4\*2+7+7\*2+4+4\*2 - identify the even position numbers and multiply by 2**

**2+0+0+6+1+2+3+8+3+16+6+8+7+14+4+8 - Multiplied**

**2+0+0+6+1+2+3+8+3+(1+6)+6+8+7+(1+4)+4+8 - If results in 2 digit number sum them up**

**2+0+0+6+1+2+3+8+3+7+6+8+7+5+4+8 - Sum up all the values**

**70%10-> Divide by 10 if 0 remainder then valid card number**

**Models**

**Card**

using System.ComponentModel.DataAnnotations;

namespace CardValidation.Models

{

public class Card

{

[MinLength(16, ErrorMessage = "Invalid card length")]

[MaxLength(16, ErrorMessage = "Invalid card length")]

public string CardNumber { get; set; }

}

}

**CardResult**

namespace CardValidation.Models

{

public class CardResult : Card

{

public bool IsValid { get; set; }

}

}

**ICardServices**

using CardValidation.Models;

namespace CardValidation.Interfaces

{

public interface ICardService

{

bool ValidateCard(string ccNumber);

}

}

**Services**

CardServices

using CardValidation.Interfaces;

using CardValidation.Models;

namespace CardValidation.Services

{

public class CardServices : ICardService

{

public bool ValidateCard(string ccNumber)

{

var numberCheckResult = CheckCardNumber(ccNumber);

if (numberCheckResult)

{

ccNumber = ReverseCardnumner(ccNumber);

int[] cardNumber = ConvertStringToArray(ccNumber);

cardNumber = IdentifyEvenPositionAndProcess(cardNumber);

int sum = SumTheArray(cardNumber);

return CheckMod(sum);

}

return false;

}

bool CheckCardNumber(string ccNumber)

{

foreach (var item in ccNumber)

{

if (char.IsNumber(item) == false)

return false;

}

return true;

}

string ReverseCardnumner(string ccNumber)

{

var reversedNumber = new string(ccNumber.Reverse().ToArray());

return reversedNumber;

}

int[] ConvertStringToArray(string ccNumber)

{

int[] cardNumber = new int[ccNumber.Length];

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

{

cardNumber[i] = Convert.ToInt32(ccNumber[i].ToString());

}

return cardNumber;

}

int[] IdentifyEvenPositionAndProcess(int[] ccNumber)

{

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

{

if (i % 2 != 0)

{

ccNumber[i] \*= 2;

if (ccNumber[i] > 9)

{

ccNumber[i] -= 9;

}

}

}

return ccNumber;

}

int SumTheArray(int[] ccNumber)

{

int sum = 0;

sum = ccNumber.Sum();

return sum;

}

bool CheckMod(int ccNumber)

{

if (ccNumber % 10 == 0)

return true;

return false;

}

}

}

Controller

CardController

using CardValidation.Interfaces;

using CardValidation.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace CardValidation.Controllers

{

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

[ApiController]

public class CardController : ControllerBase

{

private readonly ICardService \_cardService;

public CardController(ICardService cardService)

{

\_cardService = cardService;

}

[HttpPost]

public ActionResult ValidateCard(Card card)

{

if (ModelState.IsValid)

{

var result = \_cardService.ValidateCard(card.CardNumber);

if (result)

{

CardResult cardResult = new CardResult()

{

CardNumber = card.CardNumber,

IsValid = result

};

return Ok(cardResult);

}

}

return BadRequest("Invalid card number. Dont cheat!!!");

}

}

}

Program

using CardValidation.Interfaces;

using CardValidation.Services;

namespace CardValidation

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddScoped<ICardService, CardServices>();

// 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.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

Unit test

using CardValidation.Interfaces;

using CardValidation.Services;

namespace CreditCardTest

{

public class Tests

{

ICardService cardService;

[SetUp]

public void Setup()

{

cardService = new CardServices();

}

[TestCase("4477468343113002")]

[TestCase("Checkk")]

[TestCase("6477468343113002")]

[TestCase("447746834311302")]

public void ValidCheck(string number)

{

var result = cardService.ValidateCard(number);

Assert.IsTrue(result);

}

}

}

**Take 10 numbers from user and identify if a number and its square are present. If yes then print the number**

**Model**

using System.ComponentModel.DataAnnotations;

namespace NumberExercise.Models

{

public class NumberFromUser

{

[MinLength(10, ErrorMessage = "Invalid Array Length")]

[MaxLength(10, ErrorMessage = "Invalid Array Length")]

public int[] Numbers { get; set; }

}

}

**Interfaces**

namespace NumberExercise.Interfaces

{

public interface INumberService

{

List<int> FindSquare(int[] numbers);

}

}

**Services**

using NumberExercise.Interfaces;

namespace NumberExercise.Services

{

public class NumberServices : INumberService

{

public List<int> FindSquare(int[] numbers)

{

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

foreach (var a in numbers)

{

var square = a \* a;

if (numbers.Contains(square))

{

result.Add(a);

}

}

return result;

}

}

}

**Controller**

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using NumberExercise.Interfaces;

using NumberExercise.Models;

namespace NumberExercise.Controllers

{

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

[ApiController]

public class NumberSquareController : ControllerBase

{

private readonly INumberService \_numService;

public NumberSquareController(INumberService numberService)

{

\_numService = numberService;

}

[HttpPost]

public ActionResult FindSquare(NumberFromUser input)

{

if (input == null)

{

return BadRequest("Input can not be empty");

}

List<int> result = \_numService.FindSquare(input.Numbers);

if (result == null)

{

return NotFound("Not Found");

}

return Ok(result);

}

}

}

**Program.cs**

using NumberExercise.Interfaces;

using NumberExercise.Services;

namespace NumberExercise

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddScoped<INumberService, NumberServices>();

// 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();

}

}

}

Unit test

using Microsoft.VisualStudio.TestPlatform.ObjectModel;

using NumberExercise.Interfaces;

using NumberExercise.Services;

namespace NumberTest

{

public class Tests

{

INumberService \_num;

[SetUp]

public void Setup()

{

\_num = new NumberServices();

}

[Test]

public void check()

{

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 16 };

var result = \_num.FindSquare(numbers);

Assert.Contains(1, result);

Assert.Contains(2, result);

Assert.Contains(4, result);

Assert.Contains(3, result);

}

}

}

Take a string from user the words seperated by comma(","). Seperate the words and find out the longest and the shortest word in it

namespace shortestAndLongestString

{

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter the String");

string word = Console.ReadLine();

string[] words = word.Split(',');

if(words.Length > 0 )

{

string shortstring = words[0].Trim();

string longstring = words[0].Trim();

foreach(string i in words)

{

string sepWord = i.Trim();

if(sepWord.Length < shortstring.Length )

{

shortstring = sepWord;

}

if(sepWord.Length > longstring.Length )

{

longstring = sepWord;

}

}

Console.WriteLine($"Longest String is {longstring}");

Console.WriteLine($"shortest String is {shortstring}");

}

else

{

Console.WriteLine("No string Is found");

}

}

}

}