

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №4 по дисциплине «Автоматизация тестирования»

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гижевская В.Д.

Обучающийся группы 6231-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алкеев М.Г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобанков А.А.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146727791)

[2 Результаты работы 4](#_Toc146727792)

[2.1 Результаты задания 1 4](#_Toc146727793)

[2.2 Результаты задания 2 5](#_Toc146727794)

[2.3 Результаты задания 3](#_Toc146727795) 7

[2.4 Результаты задания 4](#_Toc146727795) 8

[ПРИЛОЖЕНИЕ А Листинг программы 9](#_Toc146727796)

1. Постановка задачи

В процессе выполнения заданий ознакомиться с фреймворком Angular. Изучить основы модульного тестирования angular приложения

Задание 1

Реализовать простое консольное приложение калькулятор, позволяющее производить следующие арифметические операции в 4х системах счисления: сложение, вычитание, умножение, деление. В процессе выполнения задания, рекомендуется использовать шаблон проектирования Фабрика.

Реализовать простой ангуляр приложение-калькулятор использую в качестве бэкенда программу, реализованную во второй лабораторной работе.

На странице должны присутствовать два инпута с лейблами, в которые пользователь может ввести только целые числа. Между инпутами должен располагаться dropdown элемент с возможными значениями: разделить, сложить, умножить, вычесть. Если выбрано разделить, то во второй инпут не должно быть возможности ввести 0 Инпуты должны быть реализованы в виде отдельного ангуляр компонента. Пользователю должна быть дана возможность выбрать систему счисления. Для каждой системы счисления должен быть контроль ввода чисел (например если это двоичная система, то вводить можно только 0 и 1). На странице также должна находиться кнопка по нажатию на которую выполнится соответствующее арифметическое действие. Результат должен быть выведен в отдельном html тэге (div, input и т.д.).

Задание 2

Реализовать директиву, которая окрасит результат в красный цвет, если он меньше нуля, в черный – если равен нулю и в зеленый – если больше нуля.

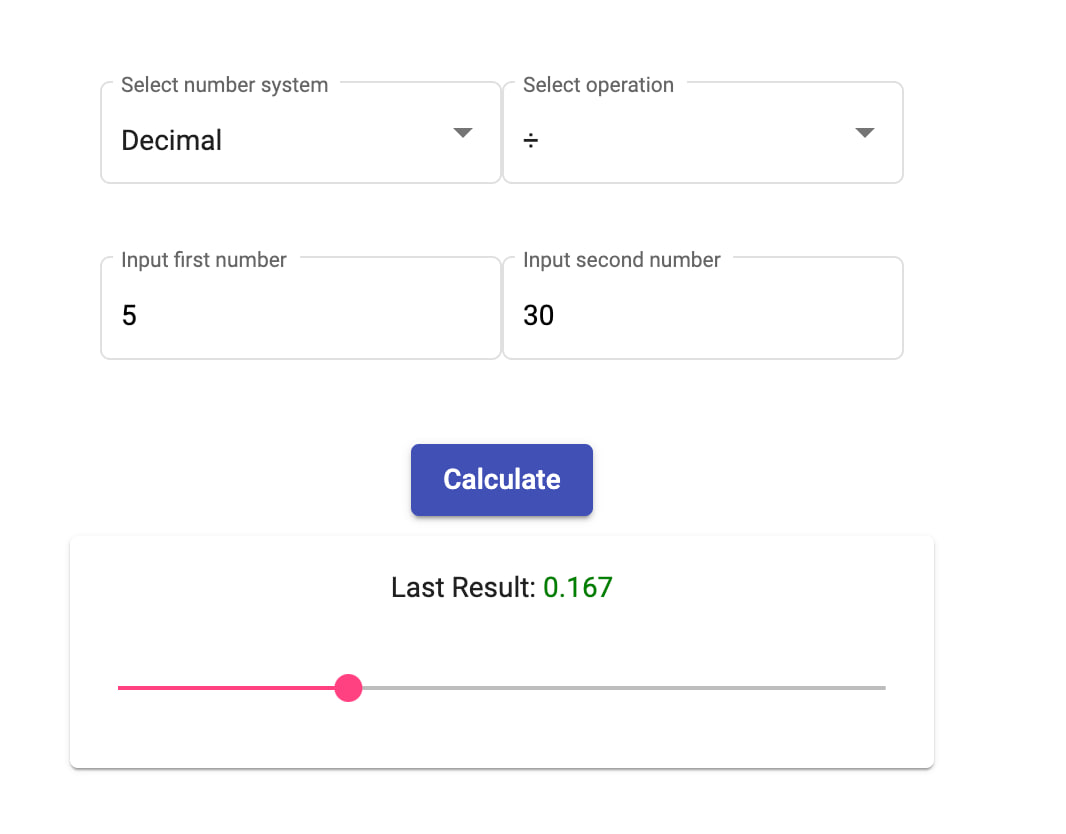
Задание 3

Реализовать собственный Pipe фильтр, который принимает в качестве входного значения количество знаков после запятой. Применить реализованный pipe фильтр к результату.

Задание 4

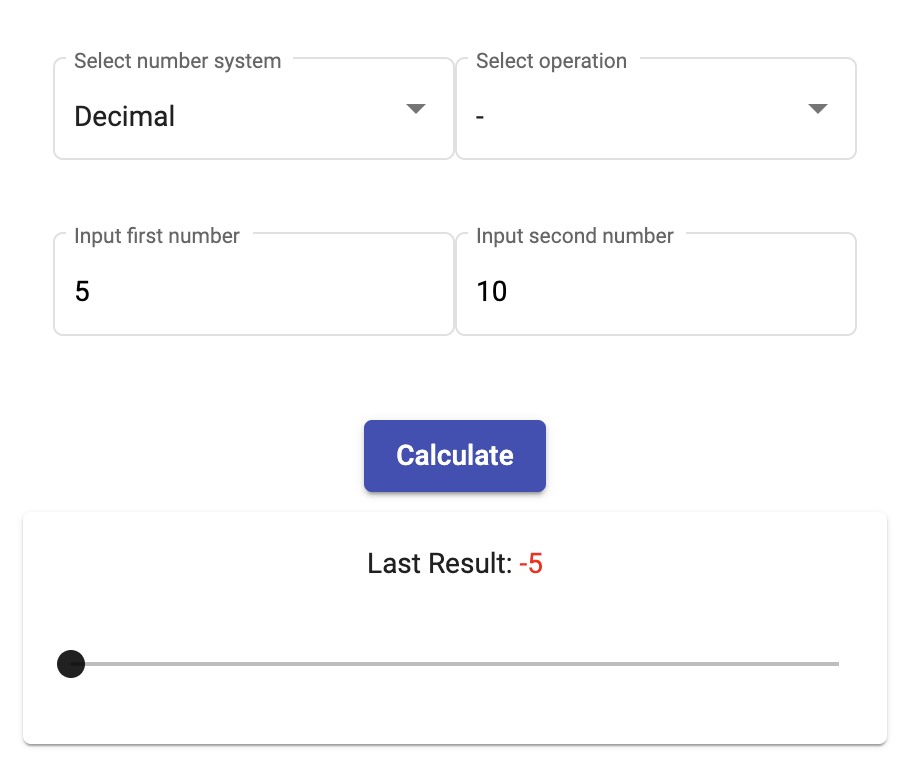
Написать Unit тесты для директивы, фильтра и для каждого angular компонента. В качестве фреймворка для тестирования можно использовать Jasmine или Jest.

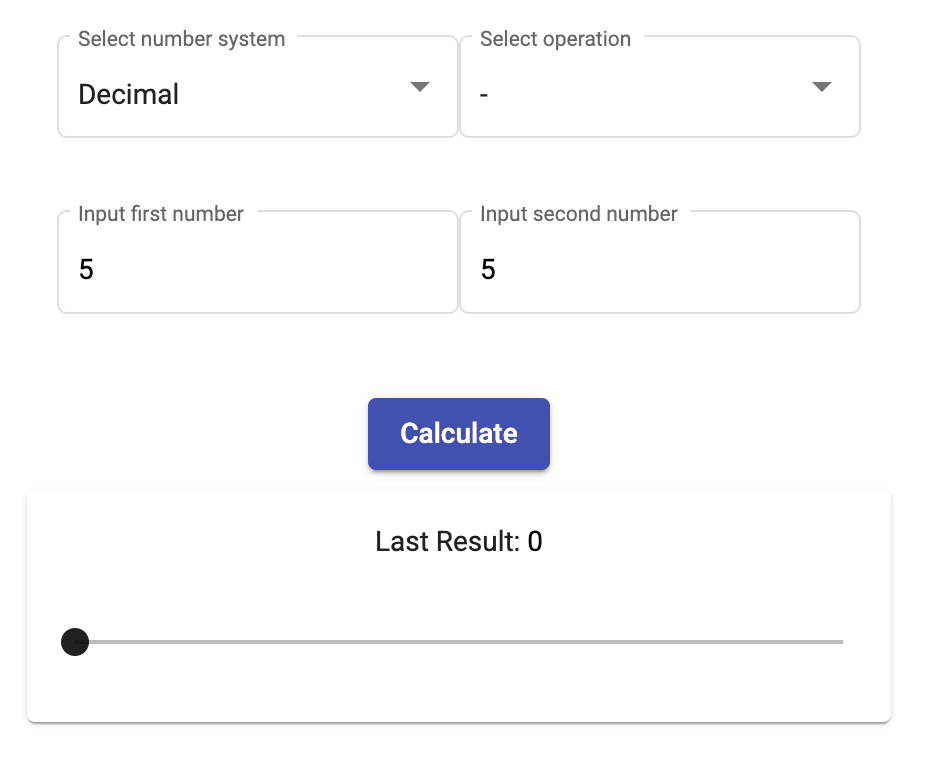
1. Результаты работы
   1. Результаты задания 1

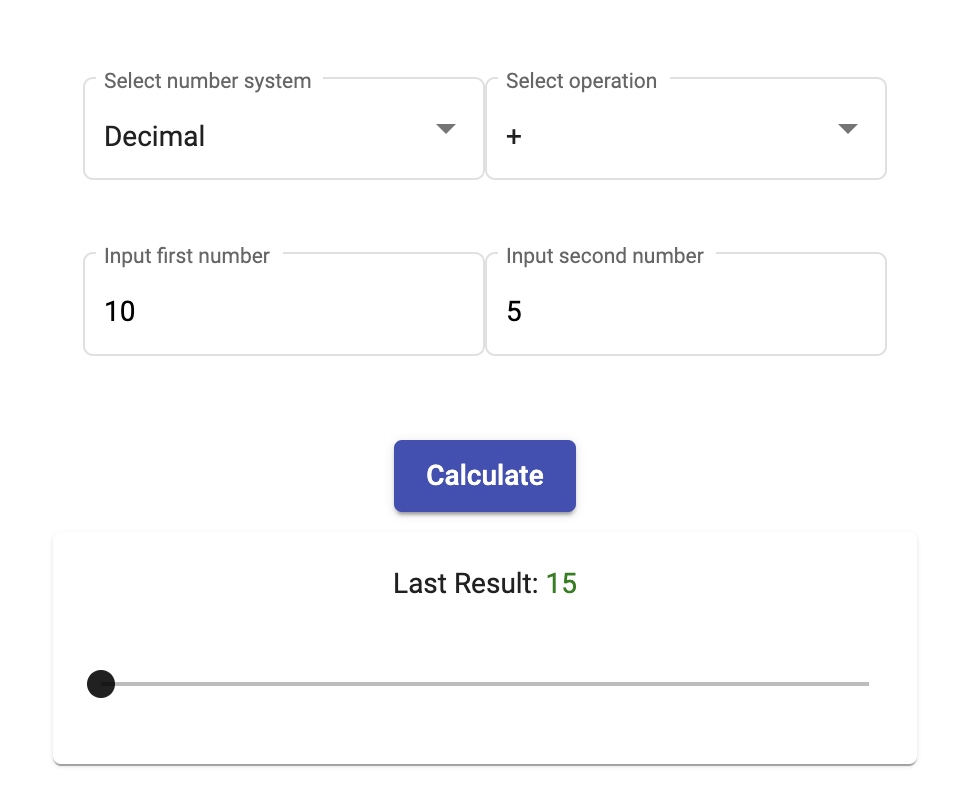
Мы реализовали angular приложение-калькулятор, состоящее из двух инпутов с лейблами, в которые пользователь может ввести только целочисленные значения, dropdown элементом со значениями умножения, деленния, вычитания и сложения, в случае выбора пользователем деления – второй инпут не позволяет ввести ноль. Инпуты реализованы в виде отдельного angular компонента. Также пользователь имеет возможность выбрать систему счисления, в зависимости от этого осуществляется контроль ввода. На странице находится кнопка «Calculate» при нажатии на которую отправляется запрос на бекенд, результат выводится в отдельном тэге

* 1. Результаты задания 2

Реализована директива, которая окрашивает результат в красный, если он меньше нуля, в черный, если равен нулю и в зеленый, если больше нуля.

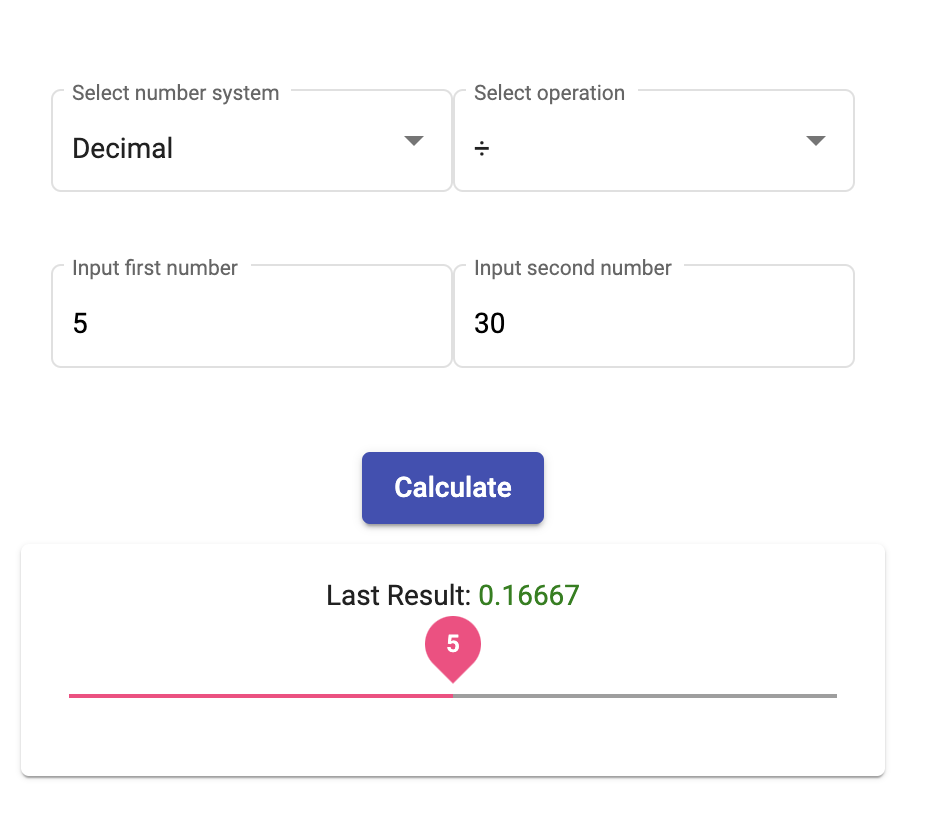






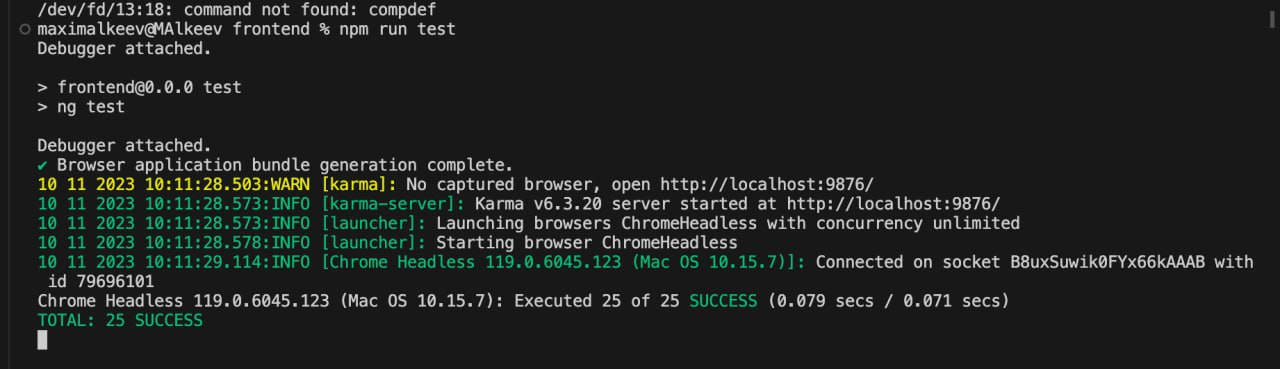
* 1. Результаты задания 3

Реализован pipe фильтр, который принимает в качестве входного параметра значение количества знаков после запятой и форматирует число, pipe применен к результату.

\

* 1. Результаты задания 4

Написаны unit тесты на все компоненты приложения, в качестве фреймворка использовали jasmine.



ПРИЛОЖЕНИЕ А  
Листинг программы

import {

Directive,

ElementRef,

Input,

OnChanges,

Renderer2,

SimpleChanges,

} from '@angular/core';

@Directive({

selector: '[changeColor]',

})

export class ChangeColorDirective implements OnChanges {

@Input('changeColor')

public value!: number | string;

constructor(

private readonly elementRef: ElementRef,

private readonly renderer: Renderer2

) {}

public ngOnChanges(changes: SimpleChanges) {

this.updateColor();

}

private updateColor() {

const value = Number(this.value);

if (value < 0) {

this.renderer.setStyle(this.elementRef.nativeElement, 'color', 'red');

} else if (value === 0) {

this.renderer.setStyle(this.elementRef.nativeElement, 'color', 'black');

} else {

this.renderer.setStyle(this.elementRef.nativeElement, 'color', 'green');

}

}

}

import { ChangeColorDirective } from './change-color.directive';

import { ElementRef, Renderer2 } from '@angular/core';

describe('ChangeColorDirective', () => {

let directive: ChangeColorDirective;

let elementRefMock: ElementRef;

let rendererMock: Renderer2;

beforeEach(() => {

elementRefMock = new ElementRef(document.createElement('div'));

rendererMock = jasmine.createSpyObj('Renderer2', ['setStyle']);

directive = new ChangeColorDirective(elementRefMock, rendererMock);

});

it('should create an instance', () => {

expect(directive).toBeTruthy();

});

it('should set color to red when value is less than 0', () => {

directive.value = -5;

directive.ngOnChanges({});

expect(rendererMock.setStyle).toHaveBeenCalledWith(

elementRefMock.nativeElement,

'color',

'red'

);

});

it('should set color to black when value is 0', () => {

directive.value = 0;

directive.ngOnChanges({});

expect(rendererMock.setStyle).toHaveBeenCalledWith(

elementRefMock.nativeElement,

'color',

'black'

);

});

it('should set color to green when value is greater than 0', () => {

directive.value = 10;

directive.ngOnChanges({});

expect(rendererMock.setStyle).toHaveBeenCalledWith(

elementRefMock.nativeElement,

'color',

'green'

);

});

});

Max Alkeev, [11/10/2023 10:13 AM]

import { ChangeDetectionStrategy, Component, Input } from '@angular/core';

import { FormControl } from '@angular/forms';

import { EOperation } from '../../../../contracts/operation.enum';

import { ENumberSystemType } from '../../../../contracts/number-system-type.enum';

@Component({

selector: 'app-number-input',

templateUrl: './number-input.component.html',

styleUrls: ['./number-input.component.scss'],

changeDetection: ChangeDetectionStrategy.OnPush,

})

export class NumberInputComponent {

@Input()

public mask = '0\*';

@Input()

public label = '';

@Input()

public operation!: EOperation;

@Input()

public system!: ENumberSystemType;

@Input()

public isDivisor!: boolean;

@Input()

public control = new FormControl();

public handleInput(event: Event) {

const inputElement = event.target as HTMLInputElement;

if (this.system === ENumberSystemType.Binary) {

inputElement.value = inputElement.value.replace(/[^-01]/g, '');

}

if (this.operation === EOperation.Divide && this.isDivisor) {

if (inputElement.value.startsWith('0')) {

inputElement.value = inputElement.value.slice(1);

}

}

inputElement.value = inputElement.value.replace(/[^-0123456789]/g, '');

}

}

Max Alkeev, [11/10/2023 10:14 AM]

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { NumberInputComponent } from './number-input.component';

import { EOperation } from '../../../../contracts/operation.enum';

import { ENumberSystemType } from '../../../../contracts/number-system-type.enum';

import { FormControl } from '@angular/forms';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatInputModule } from '@angular/material/input';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { By } from '@angular/platform-browser';

import { NO\_ERRORS\_SCHEMA } from '@angular/core';

describe('NumberInputComponent', () => {

let component: NumberInputComponent;

let fixture: ComponentFixture<NumberInputComponent>;

beforeEach(() => {

TestBed.configureTestingModule({

declarations: [NumberInputComponent],

imports: [MatFormFieldModule, MatInputModule, BrowserAnimationsModule],

schemas: [NO\_ERRORS\_SCHEMA],

});

fixture = TestBed.createComponent(NumberInputComponent);

component = fixture.componentInstance;

component.control = new FormControl();

component.system = ENumberSystemType.Decimal;

component.operation = EOperation.Add;

component.isDivisor = false;

});

it('should create', () => {

expect(component).toBeTruthy();

});

describe('Model', () => {

describe('handleInput()', () => {

it('should handle input for binary number system (leave only 0 and 1)', () => {

const event = { target: { value: '011209792dfhy5tq' } };

component.system = ENumberSystemType.Binary;

component.handleInput(event as any);

expect(event.target.value).toBe('0110');

});

it('should handle input for Divide operation (remove zero if first)', () => {

const event = { target: { value: '01' } };

component.system = ENumberSystemType.Binary;

component.operation = EOperation.Divide;

component.isDivisor = true;

component.handleInput(event as any);

expect(event.target.value).toBe('1');

});

it('should leave only numbers', () => {

const event = { target: { value: '01fefe341gg4&^%$' } };

component.system = ENumberSystemType.Decimal;

component.operation = EOperation.Add;

component.handleInput(event as any);

expect(event.target.value).toBe('013414');

});

});

});

describe('View', () => {

it('should render the label', () => {

component.label = 'Test Label';

fixture.detectChanges();

const labelDebugElement = fixture.debugElement.query(By.css('mat-label'));

expect(labelDebugElement.nativeElement.textContent).toBe('Test Label');

});

});

});

Max Alkeev, [11/10/2023 10:14 AM]

import { RoundNumberPipe } from './round-number.pipe';

describe('RoundNumberPipe', () => {

let pipe: RoundNumberPipe;

beforeEach(() => {

pipe = new RoundNumberPipe();

});

it('create an instance', () => {

expect(pipe).toBeTruthy();

});

it('should round a number with given precision', () => {

expect(pipe.transform(3.14159, 2)).toBe(3.14);

});

it('should return 0 if precision is not a number', () => {

expect(pipe.transform(5.555, 'invalid' as any)).toBe(0);

});

it('should return 0 if value is not a number', () => {

expect(pipe.transform('invalid', 2)).toBe(0);

});

it('should return 0 if precision is less than 0', () => {

expect(pipe.transform(10, -2)).toBe(0);

});

it('should return 0 if precision is NaN', () => {

expect(pipe.transform(8.88, NaN)).toBe(0);

});

it('should return 0 if value is NaN', () => {

expect(pipe.transform(NaN, 2)).toBe(0);

});

});

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'roundNumber',

})

export class RoundNumberPipe implements PipeTransform {

transform(value: number | string, precision: number): number {

value = Number(value);

if (isNaN(value) isNaN(precision) precision < 0) {

return 0;

}

const factor = Math.pow(10, precision);

return Math.round(value \* factor) / factor;

}

}

import {

ChangeDetectionStrategy,

ChangeDetectorRef,

Component,

OnDestroy,

OnInit,

} from '@angular/core';

import { FormControl } from '@angular/forms';

import { EOperation } from '../../../contracts/operation.enum';

import { ENumberSystemType } from '../../../contracts/number-system-type.enum';

import { CalculateApiService } from './services/calculate-api.service';

import { Subject, merge, takeUntil } from 'rxjs';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss'],

changeDetection: ChangeDetectionStrategy.OnPush,

})

export class AppComponent implements OnInit, OnDestroy {

private readonly destroyer$ = new Subject<void>();

public lastResult: string | number | null = null;

public decimalPlaces = 0;

public readonly firstNumberControl = new FormControl<string>('', {

nonNullable: true,

});

public readonly secondNumberControl = new FormControl<string>('', {

nonNullable: true,

});

public readonly operationControl = new FormControl<EOperation>(

EOperation.Add,

{ nonNullable: true }

);

public readonly numberSystemControl = new FormControl<ENumberSystemType>(

ENumberSystemType.Decimal,

{ nonNullable: true }

);

public readonly operations = [

{ value: EOperation.Add, label: '+' },

{ value: EOperation.Subtract, label: '-' },

{ value: EOperation.Multiply, label: '\*' },

{ value: EOperation.Divide, label: '÷' },

];

public readonly numberSystemsTypes = [

{ value: ENumberSystemType.Binary, label: 'Binary' },

{ value: ENumberSystemType.Decimal, label: 'Decimal' },

{ value: ENumberSystemType.Octal, label: 'Octal' },

{ value: ENumberSystemType.Hexadecimal, label: 'Hexadecimal' },

];

public get isNumbersEmpty(): boolean {

if (

this.firstNumberControl.value === '' ||

this.secondNumberControl.value === ''

) {

return true;

}

return false;

}

constructor(

private readonly calculateApiService: CalculateApiService,

private readonly cdr: ChangeDetectorRef

) {}

public ngOnInit(): void {

merge(

this.operationControl.valueChanges,

this.numberSystemControl.valueChanges

)

.pipe(takeUntil(this.destroyer$))

.subscribe(() => this.resetNumbers());

}

public submit(): void {

if (this.isNumbersEmpty) {

return;

}

const system = this.numberSystemControl.value;

const operation = this.operationControl.value;

const firstNumber = this.firstNumberControl.value;

const secondNumber = this.secondNumberControl.value;

this.calculateApiService

.getResult(system, operation, firstNumber, secondNumber)

.pipe(takeUntil(this.destroyer$))

.subscribe((response) => {

this.lastResult = response.result;

this.cdr.detectChanges();

});

}

public resetNumbers(): void {

this.firstNumberControl.reset();

this.secondNumberControl.reset();

}

public ngOnDestroy(): void {

this.destroyer$.next();

this.destroyer$.complete();

}

}

import { ComponentFixture, TestBed } from '@angular/core/testing';

import { AppComponent } from './app.component';

import { CalculateApiService } from './services/calculate-api.service';

import { EOperation } from '../../../contracts/operation.enum';

import { ENumberSystemType } from '../../../contracts/number-system-type.enum';

import { of } from 'rxjs';

import { RoundNumberPipe } from './round-number/round-number.pipe';

import { NO\_ERRORS\_SCHEMA } from '@angular/core';

describe('AppComponent', () => {

let component: AppComponent;

let fixture: ComponentFixture<AppComponent>;

beforeEach(() => {

TestBed.configureTestingModule({

declarations: [AppComponent, RoundNumberPipe],

providers: [

{

provide: CalculateApiService,

useValue: { getResult: () => of({ result: 10 }) },

},

],

schemas: [NO\_ERRORS\_SCHEMA],

});

fixture = TestBed.createComponent(AppComponent);

component = fixture.componentInstance;

});

it('should create the app', () => {

expect(component).toBeTruthy();

});

describe('Model', () => {

describe('get isNumbersEmpty', () => {

it('should return TRUE if inputs are empty', () => {

component.firstNumberControl.setValue('');

component.secondNumberControl.setValue('');

expect(component.isNumbersEmpty).toBe(true);

});

it('should return FALSE if inputs have values', () => {

component.firstNumberControl.setValue('5');

component.secondNumberControl.setValue('10');

expect(component.isNumbersEmpty).toBe(false);

});

});

describe('ngOnInit()', () => {

it('should call resetNumbers() when operation or number system change', () => {

spyOn(component, 'resetNumbers');

component.ngOnInit();

component.operationControl.setValue(EOperation.Add);

component.numberSystemControl.setValue(ENumberSystemType.Decimal);

expect(component.resetNumbers).toHaveBeenCalledTimes(2);

});

});

describe('submit()', () => {

it('should update lastResult on form submit', () => {

const calculateApiServiceGetResultSpy = spyOn(

component['calculateApiService'],

'getResult'

).and.returnValue(of({ result: 20 }));

component.operationControl.setValue(EOperation.Add);

component.numberSystemControl.setValue(ENumberSystemType.Decimal);

component.firstNumberControl.setValue('5');

component.secondNumberControl.setValue('15');

component.submit();

expect(calculateApiServiceGetResultSpy).toHaveBeenCalledOnceWith(

ENumberSystemType.Decimal,

EOperation.Add,

'5',

'15'

);

expect(component.lastResult).toBe(20);

});

it('should NOT call calculateApiService.getResult()', () => {

const calculateApiServiceGetResultSpy = spyOn(

component['calculateApiService'],

'getResult'

).and.returnValue(of({ result: 20 }));

component.operationControl.setValue(EOperation.Add);

component.numberSystemControl.setValue(ENumberSystemType.Decimal);

component.firstNumberControl.setValue('');

component.secondNumberControl.setValue('');

component.submit();

expect(calculateApiServiceGetResultSpy).not.toHaveBeenCalled();

expect(component.lastResult).toBeNull();

});

});

describe('resetNumbers()', () => {

it('should reset numbers', () => {

component.firstNumberControl.setValue('10');

component.secondNumberControl.setValue('20');

component.resetNumbers();

expect(component.firstNumberControl.value).toBe('');

expect(component.secondNumberControl.value).toBe('');

});

});

});

});