

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 6232-020402D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Александров А.А.

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

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc152725689)

[2 Результаты работы 4](#_Toc152725690)

[2.1 Результаты задания 1 4](#_Toc152725691)

[2.2 Результаты задания 2 6](#_Toc152725692)

[2.3 Результаты задания 3 7](#_Toc152725693)

[2.4 Результаты задания 4 8](#_Toc152725694)

[3 Вывод 9](#_Toc152725695)

[ПРИЛОЖЕНИЕ А Листинг программы 10](#_Toc152725696)

[ПРИЛОЖЕНИЕ Б Jasmine тесты 15](#_Toc152725697)

1. Постановка задачи

**Цель и постановка задания**

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

**Задание 1**

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.

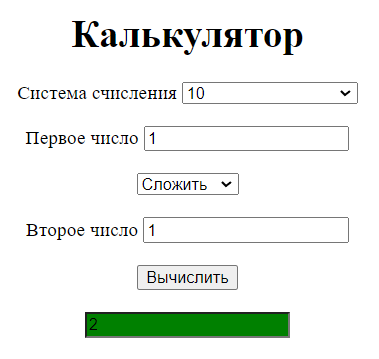
**Задание 4**

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

Приложение реализовано при помощи фреймворка Angular. В результате было сделано приложение, на странице которого присутствуют два инпута с лейблами, в которые пользователь может ввести значения в зависимости от выбранной системы счисления (2, 8, 10, 16). Присутствует dropdown элемент с возможными значениями: разделить, сложить, умножить, вычесть. Выбор системы счисления реализован также в виде dropdown элемента. Если выбрано разделить, то во второй инпут нельзя ввести 0. Инпуты реализованы в виде отдельного компонента. Пользователь может выбрать систему счисления, а также выбрать операцию с которой он будет работать. Для каждой системы счисления реализован контроль ввода чисел (например, если это двоичная система, то вводить можно только 0 и 1). На странице находиться кнопка, по нажатию на которую выполнится соответствующее арифметическое действие. Результат выводится в отдельном html тэге input.

Код фронтенд приложения описан в приложении А, на рисунке 1 представлена полученная страница приложения.

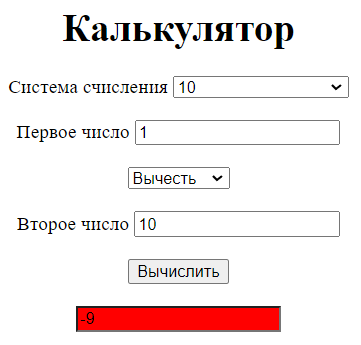
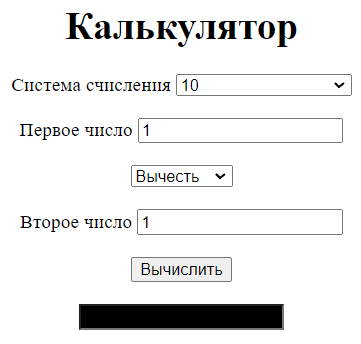
  
Рисунок 1 – Главная страница приложения калькулятор

* 1. Результаты задания 2

Для окраски результата в цвета была создана директива redblackgreen.directive.ts, которая окрасит результат в красный цвет, если он меньше нуля, в черный – если равен нулю и в зеленый – если больше нуля. На рисунке 2 представлен код директивы.

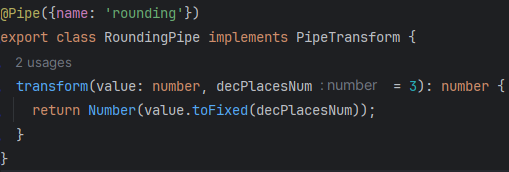
  
Рисунок – Код директивы redblackgreen.directive.ts

На рисунке 3 представлены результаты работы директивы (на рис.1 директива работает для положительного значения)

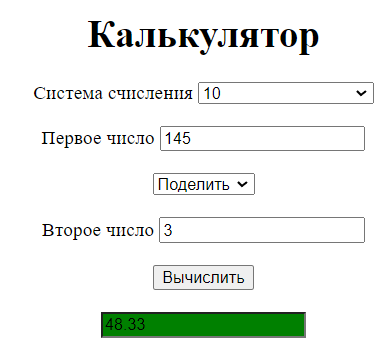
  
Рисунок 3 – Результат работы директивы

* 1. Результаты задания 3

Реализован собственный Pipe фильтр, который принимает в качестве входного значения количество знаков после запятой. На рисунке 4 представлен код реализованного pipe фильтра.

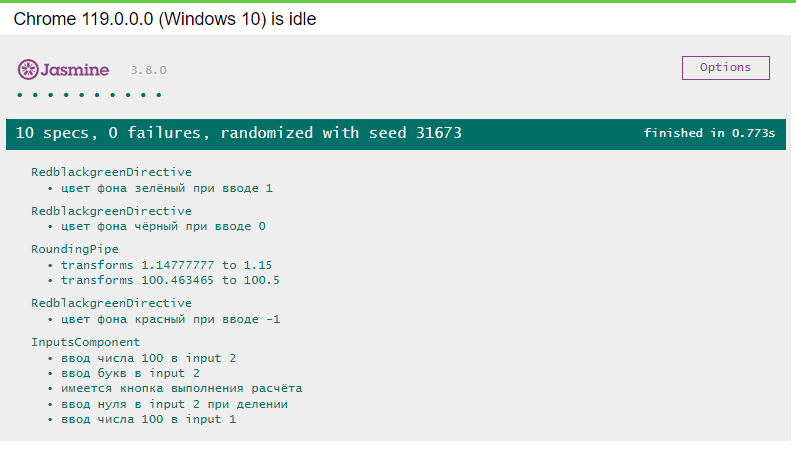
  
Рисунок 4 – Код pipe фильтра

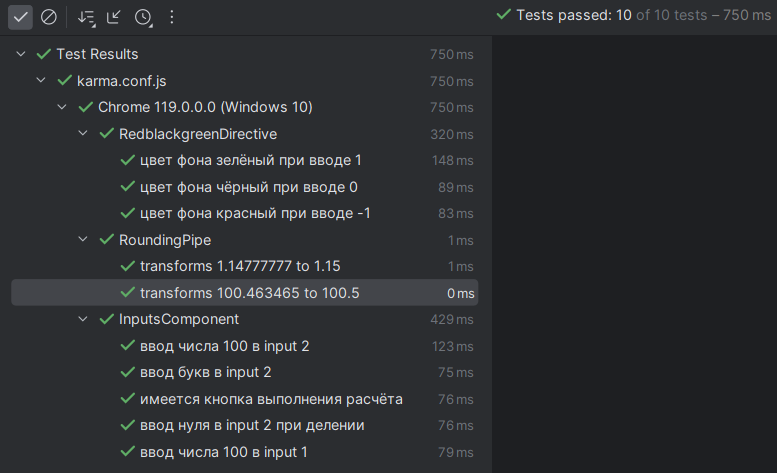
На рисунке 5 показано применение реализованного pipe фильтра к результату.

  
Рисунок 5 – Результат работы фильтра

* 1. Результаты задания 4

В качестве фреймворка для тестирования выбран Jasmine. Все проведенные тесты описаны в приложении Б. На рисунке 6 и 7 представлены результаты тестов, которые выполнены при помощи Jasmine и Karma.

  
Рисунок 7 – Результат выполнения тестов в браузере

  
Рисунок 7 – Результат тестов в консоли

1. Вывод

В результате выполнения лабораторной работы было реализовано приложение «Калькулятор систем счисления» при помощи фреймворка Angular.

Реализована директива, которая окрасит результат в нужный цвет в зависимости от получаемого значения. Реализован собственный Pipe фильтр.

Выполнены Unit тесты для директивы, фильтра и для каждого angular компонента. В качестве фреймворка для тестирования можно использовался Jasmine.

Написан отчёт.

[Исходный код доступен по ссылке](https://github.com/Allexeey/autotests-lab1/tree/lab-45)

ПРИЛОЖЕНИЕ А  
Листинг программы

<div style="text-align:center">

<h1>

{{ title }}

</h1>

<inputs (calcDataEvent)="calc($event)" (resultEvent)="updateResult($event)" ></inputs>

<!--<input [input]="num3 | rounding:2" [redblackgreen] id="input3" name="input3" type="text" />-->

<input [input]="num3" [redblackgreen]="" id="input3" name="input3" type="text" />

</div>

import {Component} from '@angular/core';

import { AppService } from './service/app.component.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

constructor(private appService: AppService){};

title: string = 'Калькулятор';

updateResult(result: string): void {

this.num3 = (result);

}

num3: string = '2';

operations = [

"Сложить",

"Вычесть",

"Поделить",

"Умножить"

];

calc(calcData: CalcData) {

switch (calcData.selectedValue) {

case (this.operations)[0]: {

this.appService.addition(calcData.num1, calcData.num2, calcData.system).subscribe()

break;

}

case (this.operations)[1]: {

this.appService.subtraction(calcData.num1, calcData.num2, calcData.system).subscribe()

break;

}

case (this.operations)[2]: {

this.appService.division(calcData.num1, calcData.num2, calcData.system).subscribe()

break;

}

case (this.operations)[3]: {

this.appService.multiplication(calcData.num1, calcData.num2, calcData.system).subscribe()

break;

}

}

}

}

export interface CalcData {

num1: string;

num2: string;

selectedValue: string;

system: string;

}

<label>Система счисления </label>

<select

id="system"

class="custom-select mb-3"

(change)="changeSystem($event)"

>

<option value="">Система счисления</option>

<option \*ngFor="let s of System">

{{ s }}

</option>

</select>

<br>

<br>

<label>Первое число </label>

<input [(ngModel)]="inputNum1" id="input1" name="inputNum1" type="text" (keypress)="keyPress($event, getPatternParam())">

<br>

<br>

<select [(ngModel)]="inputSelectedValue" id="operationSelector">

<option \*ngFor="let o of operations" [ngValue]="o">{{o}}</option>

</select>

<br>

<br>

<label>Второе число </label>

<input [(ngModel)]="inputNum2" (keypress)="keyPress($event, getPatternParam())" id="input2" name="inputNum2" type="text" oninput='let e = document.getElementById("operationSelector");

let text = e.options[e.selectedIndex].text;

if (text.localeCompare("Поделить") === 0)

{

if(!this.value.match("^[1-9][0-9]\*$"))

this.value="";

}

else {

if(!this.value.match("^[0-9]\*$"))

this.value="";

}'

pattern="^[1-9][0-9]\*$" required/>

<br>

<br>

<button (click)="emitCalcData()" id="calcButton">Вычислить</button>

<br>

<br>

import { Component, EventEmitter, Output } from '@angular/core';

import { CalcData } from "../../app.component";

@Component({

selector: 'inputs',

templateUrl: './inputs.component.html'

})

export class InputsComponent {

@Output() calcDataEvent = new EventEmitter<CalcData>();

@Output() resultEvent = new EventEmitter<string>();

inputNum1: string = '1';

inputNum2: string = '1';

pattern: RegExp = /[]/

operations = [

"Сложить",

"Вычесть",

"Поделить",

"Умножить"

];

inputSelectedValue = (this.operations)[0];

System: any = ['2', '8', '10', '16'];

param3\_val: any= this.System[0];

changeSystem(e: any): string {

console.log(e.target.value)

this.param3\_val = e.target.value

return this.param3\_val

}

keyPress(event: KeyboardEvent, pattern: any) {

const inputChar = String.fromCharCode(event.charCode);

if (!pattern.test(inputChar)) {

event.preventDefault();

}

}

getPatternParam() {

if (this.param3\_val == '2') {

this.pattern = /[0-1]/

}

if (this.param3\_val == '8') {

this.pattern = /[0-7]/

}

if (this.param3\_val == '16') {

this.pattern = /[0-9A-F]/

}

if (this.param3\_val == '10') {

this.pattern = /[0-9]/

}

return this.pattern

}

emitCalcData() {

this.inputNum2 = this.inputNum2;

const calcData: CalcData = {

num1: this["inputNum1"],

num2: this["inputNum2"],

selectedValue: this["inputSelectedValue"],

system: this["param3\_val"]

};

this.calcDataEvent.emit(calcData);

const num1Decimal = parseInt(this.inputNum1, parseInt(this.param3\_val, 10));

const num2Decimal = parseInt(this.inputNum2, parseInt(this.param3\_val, 10));

let resultDecimal: number;

switch (this.inputSelectedValue) {

case 'Сложить':

resultDecimal = num1Decimal + num2Decimal;

break;

case 'Вычесть':

resultDecimal = num1Decimal - num2Decimal;

break;

case 'Поделить':

resultDecimal = num1Decimal / num2Decimal;

break;

case 'Умножить':

resultDecimal = num1Decimal \* num2Decimal;

break;

default:

resultDecimal = 0;

break;

}

const resultInSelectedSystem = resultDecimal.toString(parseInt(this.param3\_val, 10));

this.resultEvent.emit(resultInSelectedSystem);

}

}

import { HttpClient, HttpParams } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

@Injectable()

export class AppService {

private url: string;

constructor(private http: HttpClient) {

this.url = 'http://localhost:8080/calculations';

}

public addition(param1: string, param2: string, param3: string): Observable<any> {

return this.http.get(`${this.url + '/addition'}`, {

params: {

num1: param1,

num2: param2,

system: param3

}

})

}

public subtraction(param1: string, param2: string, param3: string): Observable<any> {

return this.http.get(`${this.url + '/subtraction'}`, {

params: {

num1: param1,

num2: param2,

system: param3

}

})

}

public division(param1: string, param2: string, param3: string): Observable<any> {

return this.http.get(`${this.url + '/division'}`, {

params: {

num1: param1,

num2: param2,

system: param3

}

})

}

public multiplication(param1: string, param2: string, param3: string): Observable<any> {

return this.http.get(`${this.url + '/multiplication'}`, {

params: {

num1: param1,

num2: param2,

system: param3

}

})

}

}

import {Directive, ElementRef, Input, OnChanges, SimpleChanges} from '@angular/core';

@Directive({

selector: '[redblackgreen]'

})

export class RedblackgreenDirective implements OnChanges {

// @ts-ignore

@Input() public redblackgreen: any;

@Input() public input: any;

constructor(private el: ElementRef) {

this.highlight('red');

}

ngOnChanges(changes: SimpleChanges) {

const inputValue = parseFloat(this.input);

if (isNaN(inputValue)) {

this.highlight('yellow');

this.el.nativeElement.value = 'NaN';

} else if (inputValue < 0) {

this.highlight('red');

} else if (inputValue === 0) {

this.highlight('black');

} else if (inputValue > 0) {

this.highlight('green');

}

/\*if (inputValue) {

this.el.nativeElement.value = this.input.toUpperCase();

}\*/

}

private highlight(color: string) {

this.el.nativeElement.style.backgroundColor = color;

}

}

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

@Pipe({name: 'rounding'})

export class RoundingPipe implements PipeTransform {

transform(value: number, decPlacesNum = 3): number {

return Number(value.toFixed(decPlacesNum));

}

}

ПРИЛОЖЕНИЕ Б  
Jasmine тесты

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

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

import {InputsComponent} from "../shared/input/inputs.component";

import {RedblackgreenDirective} from "../shared/directive/redblackgreen.directive";

import {RoundingPipe} from "../pipe/rounding.pipe";

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

describe('InputsComponent', () => {

let inpFixt: ComponentFixture<InputsComponent>;

let inpComp: InputsComponent;

beforeEach(async () => {

await TestBed.configureTestingModule({

declarations: [

AppComponent, InputsComponent, RedblackgreenDirective, RoundingPipe

],

}).compileComponents();

});

beforeEach(() => {

inpFixt = TestBed.createComponent(InputsComponent);

inpComp = inpFixt.componentInstance;

inpFixt.detectChanges();

});

it('ввод числа 100 в input 1', () => {

const input1 = inpFixt.debugElement.query(By.css('#input1'));

input1.nativeElement.value = '100';

input1.nativeElement.dispatchEvent(new Event('input'));

inpFixt.detectChanges();

expect(input1.nativeElement.value).toBe('100');

});

it('ввод букв в input 2', () => {

const input2 = inpFixt.debugElement.query(By.css('#input2'));

input2.nativeElement.value = 'qwerty';

input2.nativeElement.dispatchEvent(new Event('input'));

inpFixt.detectChanges();

expect(input2.nativeElement.value).toBe('');

});

it('ввод числа 100 в input 2', () => {

const input2 = inpFixt.debugElement.query(By.css('#input2'));

input2.nativeElement.value = '100';

input2.nativeElement.dispatchEvent(new Event('input'));

inpFixt.detectChanges();

expect(input2.nativeElement.value).toBe('100');

});

it('ввод нуля в input 2 при делении', () => {

const input2 = inpFixt.debugElement.query(By.css('#input2'));

input2.nativeElement.value = '0';

const selector = inpFixt.debugElement.query(By.css('#operationSelector'));

selector.nativeElement.value = "Поделить";

input2.nativeElement.dispatchEvent(new Event('input'));

inpFixt.detectChanges();

expect(input2.nativeElement.value).toBe('');

});

it('имеется кнопка выполнения расчёта', () => {

const calcButton = inpFixt.debugElement.query(By.css('#calcButton'));

expect(calcButton).toBeTruthy();

});

});

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

import {RedblackgreenDirective} from "../shared/directive/redblackgreen.directive";

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

import {Component} from "@angular/core";

@Component({

template: `

<input [input]="-1" [redblackgreen] id="input3" type="text"/>`

})

class DirectiveTestComponentRed {

}

describe('RedblackgreenDirective', () => {

let fixt: ComponentFixture<DirectiveTestComponentRed>;

beforeEach(() => {

fixt = TestBed.configureTestingModule({

declarations: [RedblackgreenDirective, DirectiveTestComponentRed]

})

.createComponent(DirectiveTestComponentRed);

fixt.detectChanges();

});

it('цвет фона красный при вводе -1', () => {

let des = fixt.debugElement.queryAll(By.directive(RedblackgreenDirective));

const input = des[0].nativeElement as HTMLInputElement;

expect(input.style.backgroundColor).toBe('red');

});

});

@Component({

template: `

<input [input]="0" [redblackgreen] id="input3" type="text"/>`

})

class DirectiveTestComponentBlack {

}

describe('RedblackgreenDirective', () => {

let fixt: ComponentFixture<DirectiveTestComponentBlack>;

beforeEach(() => {

fixt = TestBed.configureTestingModule({

declarations: [RedblackgreenDirective, DirectiveTestComponentBlack]

})

.createComponent(DirectiveTestComponentBlack);

fixt.detectChanges();

});

it('цвет фона чёрный при вводе 0', () => {

let des = fixt.debugElement.queryAll(By.directive(RedblackgreenDirective));

const input = des[0].nativeElement as HTMLInputElement;

expect(input.style.backgroundColor).toBe('black');

});

});

@Component({

template: `

<input [input]="1" [redblackgreen] id="input3" type="text"/>`

})

class DirectiveTestComponentGreen {

}

describe('RedblackgreenDirective', () => {

let fixt: ComponentFixture<DirectiveTestComponentGreen>;

beforeEach(() => {

fixt = TestBed.configureTestingModule({

declarations: [RedblackgreenDirective, DirectiveTestComponentGreen]

})

.createComponent(DirectiveTestComponentGreen);

fixt.detectChanges();

});

it('цвет фона зелёный при вводе 1', () => {

let des = fixt.debugElement.queryAll(By.directive(RedblackgreenDirective));

const input = des[0].nativeElement as HTMLInputElement;

expect(input.style.backgroundColor).toBe('green');

});

});

import {RoundingPipe} from "../pipe/rounding.pipe";

describe('RoundingPipe', () => {

const pipe = new RoundingPipe();

it('transforms 1.14777777 to 1.15', () => {

expect(pipe.transform(1.14777777, 2)).toBe(1.15);

});

it('transforms 100.463465 to 100.5', () => {

expect(pipe.transform(100.463465, 1)).toBe(100.5);

});

});