Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

Тестирование программных блоков
тема

Вариант 4

Преподаватель _____ А. С. Кузнецов
подпись, дата

Студент _____ А. М. Сотниченко
подпись, дата

Красноярск 2021

1 Цель работы

Изучить процесс блочного тестирования программного обеспечения.

2 Общая постановка задачи

Продемонстрировать понимание и владение навыками создания:

- простых блочных тестов;
- фикстуры тестирования;
- параметризованных тестов;
- документации разработчика.

2.1 Задание, соответствующее варианту

Для варианта 4 Рациональное число (дробь из двух целых чисел), реализовать и протестировать следующие функции:

- сравнение;
- сложение:
- вычитание;
- умножение;
- деление;

3 Ход работы

Для работы был выбран язык TypeScript (надмножество над JavaScript добавляющее статическую типизацию), среда выполнения NodeJS и фреймворк для тестирования Jest.

Напишем класс для работы с рациональными числами.

```ts
export class Rational {
  #dividend: number
  #divisor: number

  constructor(dividend: number, divisor: number) {
    this.#dividend = dividend
    this.#divisor = divisor
  }

  get dividend() {
    return this.#dividend
  }

  get divisor() {
    return this.#divisor
  }

  toString(): string {
    return `${this.dividend}/${this.divisor}`
  }

}
```

Рисунок 1 – Класс Rational

Напишем методы реализующие требуемую функциональность.

```ts
  equals(other: Rational): boolean {
    return this.dividend * other.divisor === this.divisor * other.dividend
  }

  compare(other: Rational) {
    return this.dividend * other.divisor < this.divisor * other.dividend ? other : this
  }

  add(other: Rational): Rational {
    return new Rational(this.dividend * other.divisor + this.divisor * other.dividend, this.divisor * other.divisor)
  }

  subtract(other: Rational): Rational {
    return new Rational(this.dividend * other.divisor - this.divisor * other.dividend, this.divisor * other.divisor)
  }

  multiply(other: Rational): Rational {
    return new Rational(this.dividend * other.dividend, this.divisor * other.divisor)
  }

  divide(other: Rational): Rational {
    return new Rational(this.dividend * other.divisor, this.divisor * other.dividend)
  }
}
```

Рисунок 2 – Методы Rational

Теперь задокументируем класс.

```
/**
 * Create a Rational number.
 * @param {number} dividend - The top of fraction.
 * @param {number} divisor - The bottom of fraction.
 */
constructor(dividend: number, divisor: number) {
  this.#dividend = dividend
  this.#divisor = divisor
}

/**
 * @return {number} dividend.
 */
get dividend() {…
}

/**…
get divisor() {…
}

/**
 * Get the string "a/b"
 * @return {number} divisor.
 */
toString(): string {
  return `${this.dividend}/${this.divisor}`
}

/**
 * Compare with other Rational
 * @param {Rational} other Rational
 * @return {boolean} result of comparison.
 */
equals(other: Rational): boolean {
  return this.dividend * other.divisor === this.divisor * other.dividend
}
```

Рисунок 3 – Пример документации

Напишем юнит тесты для методов класса.

```
describe('Comparison', () => {
  test('7/11 > 3/5', () => {
    const a = new Rational(3, 5)
    const b = new Rational(7, 11)

    expect(a.compare(b).equals(b)).toBeTruthy()
  })
})

describe('Addition', () => {
  test('1/2 + 3/3 = 8/4', () => {
    const a = new Rational(1, 2)
    const b = new Rational(3, 2)
    const c = new Rational(8, 4)

    expect(a.add(b).equals(c)).toBeTruthy()
  })
})

describe('Subtraction', () => {
  test('7/3 - 2/4 = 22/12', () => {
    const a = new Rational(7, 3)
    const b = new Rational(2, 4)
    const c = new Rational(22, 12)

    expect(a.subtract(b).equals(c)).toBeTruthy()
  })
})
```

Рисунок 4 – Пример юнит-тестов

Напишем параметризированные юнит-тесты.

```
describe('parametrized unit tests', () => {
  describe('Addition', () => {
    each([
      [new Rational(1, 2), new Rational(-3, -3), new Rational(9, 6)],
      [new Rational(7, 4), new Rational(-1, 5), new Rational(31, 20)],
      [new Rational(11, 12), new Rational(13, 14), new Rational(310, 168)],
    ]).test('add', (a: Rational, b: Rational, c: Rational) => {
      expect(a.add(b).equals(c)).toBeTruthy()
    })
  })
})
```

Рисунок 5 – Пример параметризированных юнит-тестов

Теперь настроим тестирование с фиксатурами. Создадим фиксатуры (Рисунок 6, 7), и скрипт, который будет генерировать тесты используя фиксатуры (Рисунок 8). На выходе получаем файл тестов (Рисунок 9).

```
import { Rational } from '..'

const a = new Rational(1, 2)
const b = new Rational(3, 2)
const c = new Rational(8, 4)

const result = a.add(b)
const expected = c

export { result, expected }
```

Рисунок 6 – Фиксатура сложения

```
import { Rational } from '..'

const a = new Rational(11, 8)
const b = new Rational(6, 11)
const c = new Rational(66, 88)

const result = a.multiply(b)
const expected = c

export { result, expected }
```

Рисунок 7 – Фиксатура умножения

```
export function generateFixtureJestSnippets(files: string[]) {
  if (!files || !files.length)
    throw new Error('A valid file name must be provided')

  const open = '// WARNING: this file is generated automatically\n\ndescribe(\'hehe\', () => {'
  const close = '\n})'

  const specs = files
    .map((fname) => {
      if (EXLUDE_FILES.includes(fname.split('.ts')[0]))
        return ''

      const specName = fname.split('.ts')[0].replace(/[-]/gi, ' ')

      return `
  it('${specName}', async() => {
    const { result, expected } = await import('./fixtures/${fname.split('.ts')[0]}')
    expect(result.equals(expected)).toBeTruthy()
  })`
    })
    .join('\n')

  return `${open}${specs}${close}`
}

function parseFileTree(err: any, files: string[]) {
  if (err) {
    console.error(err)
    process.exit(1)
  }

  const tmp = generateFixtureJestSnippets(files.filter(f => f !== 'tests'))

  fs.writeFileSync(`${SAVE_DIR}/fixtures.test.ts`, tmp)
}

fs.readdir(FIXTURES_DIR, parseFileTree)
```

Рисунок 8 – Скрипт

```
// WARNING: this file is generated automatically

describe('hehe', () => {
  it('addition.ts', async() => {
    const { result, expected } = await import('./fixtures/addition')
    expect(result.equals(expected)).toBeTruthy()
  })

  it('multiplication.ts', async() => {
    const { result, expected } = await import('./fixtures/multiplication')
    expect(result.equals(expected)).toBeTruthy()
  })
```

Рисунок 9 – Сгенерированные тесты

Запустим тесты и посмотрим на результаты.

Рисунок 10 – Результаты тестирования

Увидев результаты тестирования, идем искать ошибку, понимаем, что неправильно переписали формулу с википедии (Рисунок 11), исправляем (Рисунок 12) и снова запускаем тесты (Рисунок 13).

```
divide(other: Rational): Rational {
  return new Rational(this.dividend * other.dividend, this.divisor * other.divisor)
}
```

Рисунок 11 – Неправильная формула

```
divide(other: Rational): Rational {
  return new Rational(this.dividend * other.divisor, this.divisor * other.dividend)
}
```

Рисунок 12 – Исправленная формула

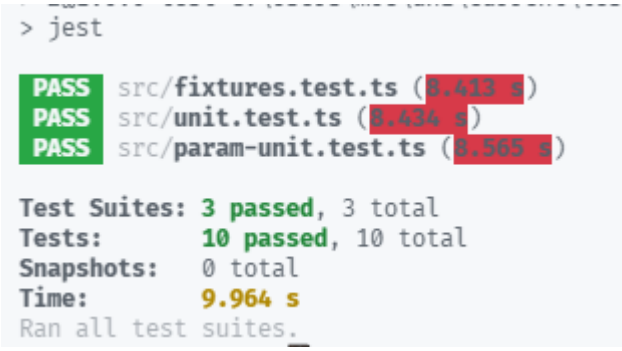

Рисунок 13 – Успешные результаты тестирования

4 Вывод

В данной работе мы ознакомились с основами блочного тестирования на языке TypeScript(JavaScript) с применением фреймворка Jest. Были рассмотрены простые и параметризованные блочные тесты. В следующей работе нам предстоит разобраться с контролируемой средой тестирования - фикстурами и фиктивными объектами.