МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Тестирование и верификация программного обеспечения»

Практическое занятие № 3

Студентка группы *ИКБО-50-23, Зернова К.А.* _____
(подпись)

Преподаватель *Ильичев Г.П.*

_____
(подпись)

Отчет представлен «___»_____2025г.

Москва 2025г.

Оглавление

Цель и задачи практической работы

Цель работы состоит в изучении и применении различных подходов к разработке программного обеспечения, основанного на тестировании, для повышения качества, надёжности и поддерживаемости кода.

Для достижения поставленной цели работы необходимо выполнить ряд задач:

1) изучить теоретические основы методологий тестирования: TDD, ATDD, BDD и SDD;

2) исследовать преимущества и недостатки каждого подхода;

3) реализовать практические примеры для каждого метода;

4) проанализировать влияние интеграции тестирования на архитектуру и качество программного продукта;

5) подготовить итоговый отчёт с выводами и рекомендациями по интеграции подходов в современные процессы разработки.

Вариант задания: Система голосования

Цель работы разработать приложение, которое позволяет пользователю:

1) Создавать голосования;

2) Регистрировать голоса;

3) Подсчитывать голоса;

4) Отображать результаты.

Часть 1. TDD

Были написаны юнит-тесты для ключевых функций системы голосования, реализован минимально необходимый код и проведен рефакторинг.

Листинг 1 - юнит-тесты для ключевых функций

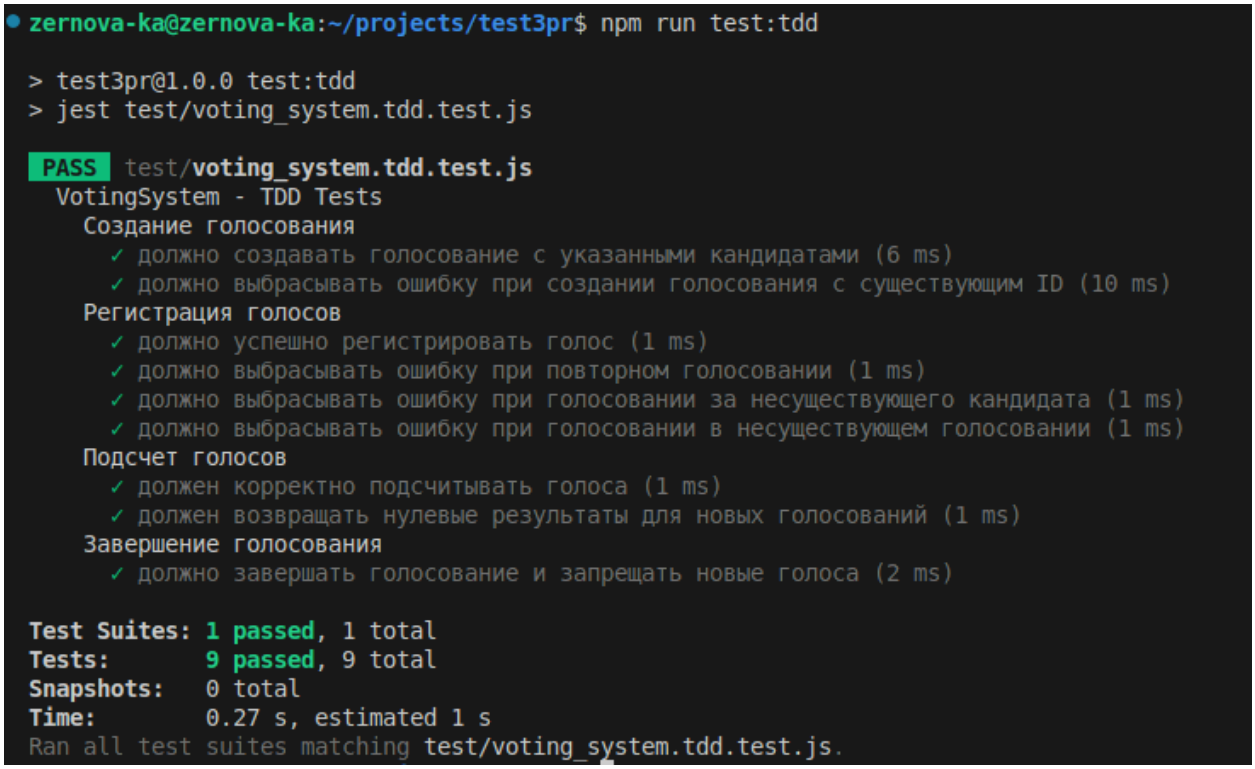
```
describe('VotingSystem - TDD Tests', () => {
    let votingSystem;

    beforeEach(() => {
        votingSystem = new VotingSystem();
    });

    describe('Создание голосования', () => {
        test('должно создавать голосование с указанными кандидатами', () => {
            const voting = votingSystem.createVoting('election1', ['Alice', 'Bob', 'Charlie']);

            expect(voting.id).toBe('election1');
            expect(voting.candidates.has('Alice')).toBe(true);
            expect(voting.candidates.has('Bob')).toBe(true);
            expect(voting.candidates.has('Charlie')).toBe(true);
            expect(voting.isActive).toBe(true);
        });

        test('должно выбрасывать ошибку при создании голосования с существующим ID', () => {
            votingSystem.createVoting('election1', ['Alice', 'Bob']);

            expect(() => {
                votingSystem.createVoting('election1', ['Charlie', 'David']);
            }).toThrow('Голосование с ID election1 уже существует');
        });
    });

    describe('Регистрация голосов', () => {
        beforeEach(() => {
            votingSystem.createVoting('election1', ['Alice', 'Bob']);
        });

        test('должно успешно регистрировать голос', () => {
            const result = votingSystem.registerVote('election1', 'voter1', 'Alice');

            expect(result).toBe(true);

            const voting = votingSystem.getVotingStatus('election1');
            expect(voting.candidates.get('Alice')).toBe(1);
            expect(voting.voters.has('voter1')).toBe(true);
        });
```

```
        test('должно выбрасывать ошибку при повторном голосовании', () => {
            votingSystem.registerVote('election1', 'voter1', 'Alice');

            expect(() => {
                votingSystem.registerVote('election1', 'voter1', 'Bob');
            }).toThrow('Голосующий уже проголосовал');
        });

        test('должно выбрасывать ошибку при голосовании за несуществующего кандидата', () => {
            expect(() => {
                votingSystem.registerVote('election1', 'voter1', 'Eve');
            }).toThrow('Кандидат Eve не существует');
        });

        test('должно выбрасывать ошибку при голосовании в несуществующем голосовании', () => {
            expect(() => {
                votingSystem.registerVote('nonexistent', 'voter1', 'Alice');
            }).toThrow('Голосование с ID nonexistent не найдено');
        });
    });

    describe('Подсчет голосов', () => {
        test('должен корректно подсчитывать голоса', () => {
            votingSystem.createVoting('election1', ['Alice', 'Bob']);

            votingSystem.registerVote('election1', 'voter1', 'Alice');
            votingSystem.registerVote('election1', 'voter2', 'Bob');
            votingSystem.registerVote('election1', 'voter3', 'Alice');

            const results = votingSystem.countVotes('election1');

            expect(results.results.Alice).toBe(2);
            expect(results.results.Bob).toBe(1);
            expect(results.totalVotes).toBe(3);
        });

        test('должен возвращать нулевые результаты для новых голосований', () => {
            votingSystem.createVoting('election1', ['Alice', 'Bob']);

            const results = votingSystem.countVotes('election1');

            expect(results.results.Alice).toBe(0);
            expect(results.results.Bob).toBe(0);
            expect(results.totalVotes).toBe(0);
        });
    });

    describe('Завершение голосования', () => {
        test('должно завершать голосование и запрещать новые голоса', () => {
            votingSystem.createVoting('election1', ['Alice', 'Bob']);

            votingSystem.endVoting('election1');

            const voting = votingSystem.getVotingStatus('election1');
            expect(voting.isActive).toBe(false);

            expect(() => {
                votingSystem.registerVote('election1', 'voter1', 'Alice');
            }).toThrow('Голосование завершено');
        });
    });
});
```

Листинг 2 - реализация минимально необходимого кода

```
class VotingSystem {
    constructor() {
        this.votings = new Map();
        this.voters = new Set();
    }

    createVoting(votingId, candidates) {
        if (this.votings.has(votingId)) {
            throw new Error(`Голосование с ID ${votingId} уже существует`);
        }

        const voting = {
            id: votingId,
            candidates: new Map(candidates.map(candidate => [candidate, 0])),
            isActive: true,
            voters: new Set()
        };

        this.votings.set(votingId, voting);
        return voting;
    }

    registerVote(votingId, voterId, candidate) {
        if (!this.votings.has(votingId)) {
            throw new Error(`Голосование с ID ${votingId} не найдено`);
        }
    countVotes(votingId) {
        if (!this.votings.has(votingId)) {
            throw new Error(`Голосование с ID ${votingId} не найдено`);
        }

        const voting = this.votings.get(votingId);
        const results = {};

        for (const [candidate, votes] of voting.candidates) {
            results[candidate] = votes;
        }

        return {
            votingId: votingId,
            results: results,
            totalVotes: voting.voters.size,
            isActive: voting.isActive
        };
    }

    displayResults(votingId) {
        const results = this.countVotes(votingId);

        console.log(`\n=== Результаты голосования ${votingId} ===`);
        console.log(`Всего голосов: ${results.totalVotes}`);
        console.log('Результаты по кандидатам:');

        for (const [candidate, votes] of Object.entries(results.results)) {
            console.log(`  ${candidate}: ${votes} голосов`);
        }

    getVotingStatus(votingId) {
        if (!this.votings.has(votingId)) {
            throw new Error(`Голосование с ID ${votingId} не найдено`);
        }

        return this.votings.get(votingId);
    }
}
```

Рисунок 1 – тесты пройдены



Часть 2. ATDD

Реализация с помощью ATDD. Реализованы приёмочные тесты для сценариев использования приложения, которые были согласованы с конечными пользователями.

Приёмочные тесты для сценариев использования системы голосования

- Сценарий 1. Каждый голос должен учитываться один раз

Предусловие: Создано новое голосование «ВыборыПрезидента» с тремя кандидатами.

Действия:

1. Первый избиратель голосует за кандидата «Иванов»;
2. Второй избиратель голосует за кандидата «Петров»;
3. Третий избиратель голосует за кандидата «Иванов»;
4. Администратор запрашивает результаты голосования.

Ожидаемый результат: Система отображает точные результаты: 2 голоса за «Иванова», 1 голос за «Петрова», 0 голосов за «Сидорова». Общее количество голосов равно 3.

- Сценарий 2. Голосующий не может проголосовать дважды

Предусловие: Избиратель уже проголосовал в голосовании.

Действия:

1. Избиратель с идентификатором «user123» голосует за вариант «Да»;
2. Тот же избиратель пытается снова проголосовать в том же голосовании, выбирая вариант «Нет»;
3. Система обрабатывает вторую попытку голосования.

Ожидаемый результат: Система отклоняет повторное голосование и показывает сообщение об ошибке. В результатах остается только первый голос за вариант «Да».

- Сценарий 3. Голосование должно корректно завершаться и подсчитывать итоги

Предусловие: Активное голосование «ВыборыСовета» с несколькими поданными голосами.

Действия:

1. Пять избирателей голосуют за различных кандидатов: два за «Джона», два за «Джейн», один за «Майка»;

2. Администратор завершает голосование;

3. Система подсчитывает итоговые результаты;

4. Шестой избиратель пытается проголосовать после завершения.

Ожидаемый результат: Голосование успешно завершено, отображаются финальные результаты (2 голоса за Джона, 2 за Джейн, 1 за Майка), новые голоса не принимаются.

Сценарий 4. Система должна обрабатывать несколько независимых голосований

Предусловие: Созданы два параллельных голосования с разными кандидатами.

Действия:

1. В голосовании «Выборы1» два избирателя голосуют: один за «А», другой за «Б»;

2. В голосовании «Выборы2» три избирателя голосуют: два за «X», один за «Y»;

3. Запрашиваются результаты обоих голосований.

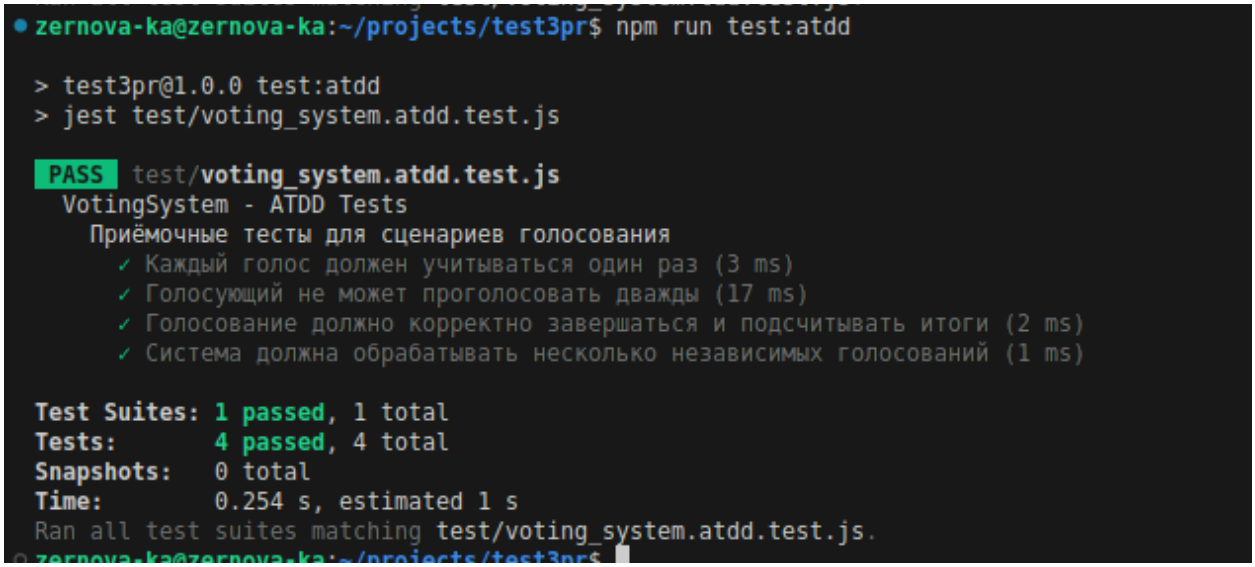
Ожидаемый результат: Каждое голосование показывает независимые результаты. Выборы1: 1 голос за А, 1 за Б. Выборы2: 2 голоса за X, 1 за Y, 0 за Z.

Листинг 3 - приёмочные тесты для сценариев использования приложения

```
        test('Система должна обрабатывать несколько независимых голосований', () => {
            // Создание нескольких голосований
            votingSystem.createVoting('election1', ['A', 'B']);
            votingSystem.createVoting('election2', ['X', 'Y', 'Z']);

            // Голосование в первом
            votingSystem.registerVote('election1', 'voter1', 'A');
            votingSystem.registerVote('election1', 'voter2', 'B');

            // Голосование во втором
            votingSystem.registerVote('election2', 'voter1', 'X');
            votingSystem.registerVote('election2', 'voter2', 'Y');
            votingSystem.registerVote('election2', 'voter3', 'X');

            // Проверка независимости результатов
            const results1 = votingSystem.countVotes('election1');
            const results2 = votingSystem.countVotes('election2');

            expect(results1.totalVotes).toBe(2);
            expect(results1.results.A).toBe(1);
            expect(results1.results.B).toBe(1);

            expect(results2.totalVotes).toBe(3);
            expect(results2.results.X).toBe(2);
            expect(results2.results.Y).toBe(1);
            expect(results2.results.Z).toBe(0);
        });
    });
});
```

```
            // Попытка повторного голосования
            expect(() => {
                votingSystem.registerVote('referendum', 'voter123', 'No');
            }).toThrow('Голосующий уже проголосовал');

            const results = votingSystem.countVotes('referendum');
            expect(results.totalVotes).toBe(1);
            expect(results.results.Yes).toBe(1);
            expect(results.results.No).toBe(0);
        });

        test('Голосование должно корректно завершаться и подсчитывать итоги', () => {
            votingSystem.createVoting('board_election', ['John', 'Jane', 'Mike']);

            // Регистрация голосов
            const voters = ['v1', 'v2', 'v3', 'v4', 'v5'];
            const votes = ['John', 'Jane', 'John', 'Mike', 'Jane'];

            voters.forEach((voter, index) => {
                votingSystem.registerVote('board_election', voter, votes[index]);
            });

            // Завершение голосования
            votingSystem.endVoting('board_election');

            // Проверка результатов
            const results = votingSystem.countVotes('board_election');

            expect(results.isActive).toBe(false);
            expect(results.totalVotes).toBe(5);
            expect(results.results.John).toBe(2);
            expect(results.results.Jane).toBe(2);
            expect(results.results.Mike).toBe(1);
        });
```

Рисунок 2 – пройденные приемочные тесты



Часть 3. BDD

Созданы сценарии на языке Gherkin, которые описывают поведение системы.

Листинг 4 - сценарии на языке Gherkin

```
# language: ru
Функция: Управление системой голосования
  Как организатор выборов
  Я хочу проводить честные и прозрачные голосования
  Чтобы обеспечить демократический процесс принятия решений

  Сценарий: Успешное голосование участника
    Пусть голосование "Выборы президента" запущено с кандидатами "Иванов,Петров,Сидоров"
    Когда участник "user123" голосует за "Иванов"
    Тогда голос учитывается
    Также отображаются результаты голосования

  Сценарий: Попытка повторного голосования
    Пусть голосование "Референдум" запущено с кандидатами "Да,Нет"
    Когда участник "voter456" голосует за "Да"
    Также участник "voter456" голосует за "Нет"
    Тогда участник "voter456" не может проголосовать повторно

  Сценарий: Подсчет результатов голосования
    Пусть голосование "Выборы в совет" запущено с кандидатами "Анна,Борис,Виктор"
    Когда участник "v1" голосует за "Анна"
    Также участник "v2" голосует за "Анна"
    Также участник "v3" голосует за "Борис"
    Также участник "v4" голосует за "Виктор"
    Также участник "v5" голосует за "Анна"
    Тогда "Анна" получает 3 голосов
    Также "Борис" получает 1 голосов
    Также "Виктор" получает 1 голосов
```

Листинг 5 – реализация тестов

```
const { Given, When, Then } = require('@cucumber/cucumber');
const { expect } = require('chai');
const VotingSystem = require('../../voting_system');

let votingSystem;
let currentVoting;
let errorMessage;

Given('голосование {string} запущено с кандидатами {string}', function (votingId, candidatesString) {
    votingSystem = new VotingSystem();
    const candidates = candidatesString.split(',').map(c => c.trim());
    currentVoting = votingSystem.createVoting(votingId, candidates);
});

When('участник {string} голосует за {string}', function (voterId, candidate) {
    try {
        votingSystem.registerVote(currentVoting.id, voterId, candidate);
    } catch (error) {
        errorMessage = error.message;
    }
});

Then('голос учитывается', function () {
    const results = votingSystem.countVotes(currentVoting.id);
    expect(results.totalVotes).to.be.greaterThan(0);
});

Then('участник {string} не может проголосовать повторно', function (voterId) {
    expect(errorMessage).to.include('уже проголосовал');
});

Then('отображаются результаты голосования', function () {
    const results = votingSystem.displayResults(currentVoting.id);
    expect(results).to.have.property('results');
    expect(results).to.have.property('totalVotes');
});

Then('{string} получает {int} голосов', function (candidate, expectedVotes) {
    const results = votingSystem.countVotes(currentVoting.id);
    expect(results.results[candidate]).to.equal(expectedVotes);
});
```

```
zernova-ka@zernova-ka:~/projects/test3pr$ npm run test:bdd

> test3pr@1.0.0 test:bdd
> cucumber-js --require features/**/*.js

...
=== Результаты голосования Выборы президента ===
Всего голосов: 1
Результаты по кандидатам:
  Иванов: 1 голосов
  Петров: 0 голосов
  Сидоров: 0 голосов
.............

3 scenarios (3 passed)
17 steps (17 passed)
0m00.028s (executing steps: 0m00.005s)
zernova-ka@zernova-ka:~/projects/test3pr$
```

Рисунок 3 – пройденные тесты сценариев

Часть 4. SDD

В проекте созданы спецификации требований с использованием конкретных примеров, которые затем были преобразованы в автоматизированные тесты.

Таблица 1 – описание спецификаций требований

Описание сценария	**ID голосования**	**Кандидаты**	**Голоса (избиратель → кандидат)**	**Ожидаемые результаты**	**Всего голосов**
Простое голосование с одним кандидатом	simple_vote	Option A	v1 → Option A	Option A: 1	1
Голосование с тремя кандидатами и равным распределением	three_way	Red, Green, Blue	v1 → Red v2 → Green v3 → Blue	Red:1 Green:1 Blue: 1	3
Голосование с явным лидером	leader_election	Alice, Bob, Charlie	v1 → Alice v2 → Alice v3 → Alice v4 → Bob v5 → Charlie	Alice:3 Bob:1 Charlie: 1	5
Голосование без голосов	no_votes	Candidate 1, Candidate 2	-	Candidate 1:0 Candidate 2:0	0

Листинг 6 – автоматизированные тесты по спецификациям

```
describe('VotingSystem - SDD Tests', () => {
    let votingSystem;

    beforeEach(() => {
        votingSystem = new VotingSystem();
    });

    // Табличные тесты на основе спецификаций
    const testCases = [
        {
            description: 'Простое голосование с одним кандидатом',
            votingId: 'simple_vote',
            candidates: ['Option A'],
            votes: [
                { voter: 'v1', candidate: 'Option A' }
            ],
            expectedResults: { 'Option A': 1 },
            expectedTotal: 1
        },
        {
            description: 'Голосование с тремя кандидатами и равным распределением',
            votingId: 'three_way',
            candidates: ['Red', 'Green', 'Blue'],
            votes: [
                { voter: 'v1', candidate: 'Red' },
                { voter: 'v2', candidate: 'Green' },
                { voter: 'v3', candidate: 'Blue' }
            ],
            expectedResults: { 'Red': 1, 'Green': 1, 'Blue': 1 },
            expectedTotal: 3
        },
        {
            description: 'Голосование с явным лидером',
            votingId: 'leader_election',
            candidates: ['Alice', 'Bob', 'Charlie'],
            votes: [
                { voter: 'v1', candidate: 'Alice' },
                { voter: 'v2', candidate: 'Alice' },
                { voter: 'v3', candidate: 'Alice' },
                { voter: 'v4', candidate: 'Bob' },
                { voter: 'v5', candidate: 'Charlie' }
            ],
            expectedResults: { 'Alice': 3, 'Bob': 1, 'Charlie': 1 },
            expectedTotal: 5
        },
        {
            description: 'Голосование без голосов',
            votingId: 'no_votes',
            candidates: ['Candidate 1', 'Candidate 2'],
            votes: [],
            expectedResults: { 'Candidate 1': 0, 'Candidate 2': 0 },
            expectedTotal: 0
        }
    ];

    testCases.forEach((testCase, index) => {
        test(`SDD Test ${index + 1}: ${testCase.description}`, () => {
            // Создание голосования
            votingSystem.createVoting(testCase.votingId, testCase.candidates);

            // Регистрация голосов
            testCase.votes.forEach(vote => {
                votingSystem.registerVote(testCase.votingId, vote.voter, vote.candidate);
            });

            // Проверка результатов
            const results = votingSystem.countVotes(testCase.votingId);

            expect(results.votingId).toBe(testCase.votingId);
            expect(results.results).toEqual(testCase.expectedResults);
            expect(results.totalVotes).toBe(testCase.expectedTotal);
        });
    });
```

```
    describe('Тесты граничных условий', () => {
        test('Обработка большого количества голосов', () => {
            const votingId = 'mass_voting';
            const candidates = ['Yes', 'No'];

            votingSystem.createVoting(votingId, candidates);

            // Симуляция 1000 голосов
            for (let i = 0; i < 1000; i++) {
                const candidate = i % 3 === 0 ? 'No' : 'Yes'; // 2/3 за Yes, 1/3 за No
                votingSystem.registerVote(votingId, `voter${i}`, candidate);
            }

            const results = votingSystem.countVotes(votingId);

            expect(results.totalVotes).toBe(1000);
            // Примерное распределение (может немного отличаться из-за округления)
            expect(results.results.Yes).toBeGreaterThan(650);
            expect(results.results.No).toBeLessThan(350);
        });

        test('Голосование с одним участником', () => {
            votingSystem.createVoting('single_voter', ['Option 1', 'Option 2']);
            votingSystem.registerVote('single_voter', 'only_voter', 'Option 1');

            const results = votingSystem.countVotes('single_voter');

            expect(results.totalVotes).toBe(1);
            expect(results.results['Option 1']).toBe(1);
            expect(results.results['Option 2']).toBe(0);
        });
    });
});
```

```
zernova-ka@zernova-ka:~/projects/test3pr$ npm run test:sdd
> test3pr@1.0.0 test:sdd
> jest test/voting_system.sdd.test.js

 PASS  test/voting_system.sdd.test.js
  VotingSystem - SDD Tests
    ✓ SDD Test 1: Простое голосование с одним кандидатом (3 ms)
    ✓ SDD Test 2: Голосование с тремя кандидатами и равным распределением (1 ms)
    ✓ SDD Test 3: Голосование с явным лидером (1 ms)
    ✓ SDD Test 4: Голосование без голосов (1 ms)
    Тесты граничных условий
      ✓ Обработка большого количества голосов (2 ms)
      ✓ Голосование с одним участником (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.251 s, estimated 1 s
Ran all test suites matching test/voting_system.sdd.test.js.
```

Рисунок 4 – пройденные тесты

Заключение

Проведенная работа по разработке системы голосования позволила создать надежный программный продукт, соответствующий современным стандартам прозрачности и безопасности. Общая оценка качества системы является высокой благодаря комплексному подходу к тестированию на всех этапах разработки. Разработанное приложение демонстрирует полную функциональную готовность к эксплуатации в различных сценариях выборов и опросов. Все ключевые операции - создание голосований, регистрация голосов, подсчет результатов и завершение процессов - реализованы с учетом требований точности и надежности.

Вывод

В данном примере «Система голосования» продемонстрирована практическая реализация приложения с последовательным применением четырёх ключевых подходов разработки через тестирование:

1. TDD - разработаны комплексные юнит-тесты для всех критических операций, реализована базовая функциональность управления паролями.

2. ATDD - определены и автоматизированы приёмочные тесты, согласованные с конечными пользователями, которые охватывают основные сценарии работы с паролями в условиях реальной эксплуатации.

3. BDD - сформулированы поведенческие сценарии в формате Gherkin, обеспечивающие понятное описание функциональности как для технических специалистов, так и для бизнес-пользователей.

4. SDD - составлена детализированная спецификация с конкретными примерами данных, которая служит одновременно живой документацией и основой для автоматизированных тестовых проверок.

Комплексное применение методологий позволило не только всесторонне протестировать функциональные возможности менеджера паролей, но и на практике продемонстрировать эффективность интеграции различных подходов к разработке через тестирование. Общий эффект от их сочетания существенно повысил качество, надежность и безопасность конечного программного продукта, обеспечив соответствие современным стандартам разработки критически важных приложений для управления учетными данными.