Unit Testing

Pada modul ini akan dilakukan unit testing pada node JS dengan menggunakan framework testing jest (https://jestjs.io/). Dan untuk pengerjaannya sudah diberikan program sederhana (template)[1], kalian diminta untuk membuat unit testing nya saja. Ikutilah Langkah pengerjaan berikut:

1. Lakukan persiapan project seperti melakukan "npm install" dari template project berikut.
2. Coba lakukan eksperimen dan analisis mengenai cara kerja program dengan menggunakan file index.js (dijalankan dengan menggunakan "node index.js"). Anda diperbolehkan melakukan uncomment kode console.log untuk memudahkan mempelajari alur kode program.
3. Tugas anda adalah menambahkan kode unit testing pada folder "test" (buat foldernya jika belum ada). Framework jest secara otomatis akan membaca seluruh file dengan suffix file "test.js". Jadi diharapkan hati-hati dalam menamai nama file.
4. Buatlah file dengan nama "m1.test.js" yang berisi kode berikut:

```
test('test 1-1', () => {
    let a = 123;
    expect(a).toBe(123);
});
```

5. Jalankan kode tersebut dengan menjalankan "npm test". Perhatikan keluaran dari perintah tersebut. Ubah nilai pada method toBe menjadi "1234" lalu jalankan kembali.
   Kesimpulan apa yang dapat didapatkan dari percobaan ini? Apa kegunaan method expect dan toBe?
   Kembalikan nilai ke angka semula.
6. Copy kode tersebut sehingga kita mempunyai test case kedua dengan nama "test 1-2". Lalu coba jalankan kembali test nya.
7. Ketika test semakin banyak, ada kemungkinan kita akan sulit untuk mengelola semua itu. Oleh karena itu, unit testing menyediakan fitur untuk mengelompokkan test tersebut. Buatlah file baru dengan nama "m2.test.js". Lalu tambahkan kode berikut:

```
describe('modules', () => {
    test('test 2-1', () => {
        let a = 123;
        expect(a).toBe(123);
    });

    describe('submodules', () =>{
        test('test 2-2', () => {
            let b = 456;
            expect(b).toBe(456);
        });
    });
});
```

   Jalankan test tersebut dan jelaskan apa yang terjadi.
8. Selanjutnya, setiap unit testing mengenal fitur yang dinamakan fixtures. Buatlah file dengan nama "m3.test.js" yang berisi kode berikut:

   Jalankan test tersebut dan jelaskan apa yang terjadi. Apa kegunaan dari test fixture ini? Apa berbedaan dari beforeEach dan beforeAll ?

[1] Clone atau unduh zip template di https://github.com/ifunpar/AIF233101-TemplateUnitTesting

```
beforeEach(() => { console.log("before each"); });
afterEach(() => { console.log("after each"); });
beforeAll(() => { console.log("before all"); });
afterAll(() => { console.log("after all"); });

test('test 3-1', () => {
    expect(true).toBe(true);
});

test('test 3-2', () => {
    expect(false).toBe(false);
});
```

9. Pada test case sebelumnya, mungkin test fixture tidak terlalu real kegunaannya. Kita akan pakai coba menggunakan test case lain sebagai berikut (file m4.test.js):

```
import { moveOnce }  from '../app/board.js';

test('test 4-1', async () => {
    let updated = await moveOnce();
    expect(updated).toBe(2);
});
```

Pada test case ini, hasil test kita sangat bergantung dengan kondisi / isi dari file test.txt yang ada. Ubahlah isi dari test.txt menjadi "1" lalu coba kembali.

10. Pada kasus ini, hasil test menjadi tidak stabil tergantung dengan factor eksternal. (Bayangkan korelasinya jika kita menggunakan basis data). Oleh karena itu kita harus memastikan sebelum test kondisi test selalu sama dan kondisi akhir test selalu sama. Tambahkan kode berikut:

```
import { moveOnce, writeFile }  from '../app/board.js';

beforeEach(async () => {
    await writeFile("1");
    //console.log("reset to 1");
});
```

Jalankan test tersebut dan jelaskan apa yang berubah dari penambahan kode tersebut.

11. Pada test case 4, method yang dipanggil adalah moveOnce yang dapat dipastikan hasil keluarannya. Copy file "m4.test.js" menjadi file "m5.test.js" lalu ubah method moveOnce menjadi move. Akan tetapi, apakah bisa kita menguji method move yang mengandung unsur random?
12. Secara mendasar hal ini tidak dapat dilakukan. Tetapi kita dapat memalsukan keluaran dari method tersebut dengan bantuan mocking. Mocking ini memiliki banyak kegunaan untuk komponen-komponen yang tidak bisa dikendalikan seperti faktor random, format masukkan yang terlalu kompleks, hasil developer lain yang belum selesai tetapi dibutuhkan, dll.
13. Salah satu keunggulan dari framework jest adalah dia menyediakan fitur unit testing dan mocking dalam 1 framework. Jadi kita tinggal menggunakan fitur tersebut. Lengkapi test case 5 dengan kode berikut:

```
import { jest } from '@jest/globals'

jest.unstable_mockModule('../app/dice.js', () => ({
    rolls: () => 6,
}));

const { move, writeFile } = await import('../app/board.js')

beforeEach(async () => {
    await writeFile("1");
    //console.log("reset to 1");
});

test('test 5-1', async () => {
    let updated = await move();
    expect(updated).toBe(7)
});
```

Jalankan test tersebut dan jelaskan apa yang berubah dari penambahan kode tersebut.

14. Kumpulkan semua pengerjaan dalam bentuk zip (folder node_module jangan dimasukkan dalam zip) pada tempat yang disediakan.