Testing

How to model readable system tests

O mnie

Kamil Gajowy

- backend developer
- boomer 👴🏻
- boardgames fan

Agenda

1. How we usually see test written.
2. Understanding tests. Knowing drivers.
3. Refactor tests to be readable.

Every story begins with Nest

```
afterAll(async () ⇒ {/** teardown db */});

describe('OrganizationsController (e2e)', () ⇒ {
    beforeAll(async () ⇒ {
        const moduleFixture: TestingModule = await Test.createTestingModule({
            imports: [FeatureModule],
        }).compile();

        app = moduleFixture.createNestApplication();
        await app.init();
    });

    afterAll(async () ⇒ {
        await Promise.all([app.close()]);
    });
});
```

Setup hell

```
describe('OrganizationsController (e2e)', () ⇒ {
    let app: INestApplication;
    let jwtToken: string;

    beforeAll(async () ⇒ {
        app = ... ;
        const response = await request(app.getHttpServer())
            .post('/auth/sign-in').send({}).expect(201);
        jwtToken = response.body.accessToken;
    });

    beforeEach(async () ⇒ {
        // stubs, mocks, spies ...
    })

    it('should ... ', () ⇒ { ... })
});
```

Nest-ed Evolution

```
describe('OrganizationsController (e2e)', () ⇒ {
    let anOrganization: { id: string; type: 'organizations' };
    let aProject: { id: string; type: 'organizations' };

    describe('Organizations', () ⇒ {
        it('Creates an organization', async () ⇒ {
            const response = await request(app.getHttpServer()).post('/api/v1/organizations')
            anOrganization = response.body.data;
            expect(anOrganization.type).toBe('organizations');
        });

        it('Creates a project in the newly created organization', async () ⇒ {
            const response = await request(app.getHttpServer())
                .post('/api/v1/projects')
                .set('Authorization', `Bearer ${jwtToken}`)
                .send(createProjectDTO)
                .expect(201);

            aProject = response.body.data;
            expect(aProject.type).toBe('projects');
        });
    });
});
```

Un(expected)

```
await waitForExpect(async () => {
    expect(await queue.getJob(scenarioId)).toMatchObject({ data });
});

expect(response.body.data.length).toEqual(1);
expect(response.body.data[0].id).toEqual(publicProjectId);

expect(resources.length).toBeLessThanOrEqual(25);
expect(resources.length).toBeGreaterThanOrEqual(1);

it('Retrieves a JWT token ' +
    'when authenticating' +
    ' with valid credentials', async () => {
    const body = await request(app.getHttpServer())
        .post('/auth/sign-in')
        .send({
            username: E2E_CONFIG.users.basic.aa.username,
            password: E2E_CONFIG.users.basic.aa.password,
        })
        .expect(201);
    expect(response.body.accessToken).toBeDefined();
});
```

How we read?

Full example of pieces shown so far:

Don’t do that at home

Questions:

- Can we understand WHAT is needed?
- Can we understand, HOW feature is supposed to work?
- Can we understand, WHEN the actions taken happen?

No-ise - GWT / AAA

Both share some principles, including **grouping** code related to particular actions, thus make the code more structured.

AAA - **Arrange**, **Act**, **Assert**

GWT - **Given**, **When**, **Then**

Many say that AAA does not fit into checking business requirements.

Is this a reason why we cannot describe our module/unit behavior?

No-ise - setup chore

```typescript
const getFixtures = async () => {
    const app = await bootstrapApplication(); // or module

    const sut = app.get(SubjectUnderTestClass);
    const cleanups: (() => Promise<void>)[] = []

    return {
        cleanup: async () => {
            await Promise.all(cleanups);
            await app.close();
        }
    }
}
```

No-ise - Arrange/Given

Something already happened

```
// before
const response = await request(app.getHttpServer())
    .post('/sign').send({/** */});

jwtToken = response.body.accessToken;

// after, within getFixtures()
return {
    cleanup: async () ⇒ { ... },
    GivenUserIsLoggedIn: async () ⇒
        (await request(app.getHttpServer())
            .post('/sign')
            .send({})).body.accessToken,
}
```

No-ise - Arrange/Given

Dependency setup

```
fakeQueue.getJob.mockImplementation(async (id) ⇒ {
        expect(id).toBe('123');
        return {id: '123'} as Job;
    });

// fixtures
fakeQueue
    .getJob
    .mockImplementation(async () ⇒
        throw new Error(`Unexpected call`))

return {
    cleanup: async () ⇒ { ... },
    GivenAHeavyComputationRequestWasSubmitted() ⇒ {
        fakeQueue.getJob.mockImplementation(async (id) ⇒ {
            expect(id).toBe('123');
            return {id: '123'} as Job;
        });
    }
}
```

No-ise - Act/When (beforeEach)

```
it('Creates a project in the newly created organization', async () ⇒ {
    const response = await request(app.getHttpServer())
        .post('/api/v1/projects')
        .set('Authorization', `Bearer ${jwtToken}`)
        .send(createProjectDTO)
        .expect(201);

    aProject = response.body.data;
    expect(aProject.type).toBe('projects');
});

{
    WhenCreatingAProjectWithName: async (name: string) ⇒ (await request(app.getHttpServer())
        .post('/api/v1/projects')
        .set('Authorization', `Bearer ${jwtToken}`)
        .send({name})
        .expect(201)).body

}
```

No-ise - Asset/Then (it, test)

Mysteries explained

```
expect(response.body.data.length).toEqual(1);
expect(response.body.data[0].id).toEqual(publicProjectId);

expect(resources.length).toBeLessThanOrEqual(25);
expect(resources.length).toBeGreaterThanOrEqual(1);

return {
    ThenResponseContainsTheOnlyPublicProject: (response: supertest.Response) ⇒ {
        expect(response.body.data.length).toEqual(1);
        expect(response.body.data[0].id).toEqual(publicProjectId);
    },
    ThenResponseHasPagination: (response: supertest.Response) ⇒ {
        expect(response.body.metadata.pagination).toEqual({
            pages: 1,
            limit: 25, // different? yes, developer meant to check pagination!
        })
    }
}
```

Clean'em up

```
{
    GivenSomething: async () ⇒ {
        const ids = await repo.insert([/** */]);
        cleanups.push(() ⇒ repo.delete({
            where: {
                id: In(ids), // "compensate"
            }
        }))
    }
    cleanup: async () ⇒ {
        // previously pushed clean functions
        await Promise.all(cleanups.map(clean ⇒ clean()));
        await app.close();
    }
}
```

Putting pieces together

```
let fixtures: FixtureType
beforeEach(async () ⇒ {
    fixtures = getFixtures();
})

// e2e/integration example
test(`User can create project within his organization`, async () ⇒ {
    await fixtures.GivenUserIsLoggedIn();
    const organizationId = await fixtures.GivenOrganizationWasCreated();
    const projectId = await fixtures.WhenCreatingAProject(organizationId);
    await fixtures.ThenProjectIsListedInCatalogue(projectId);
})

// unit/integration example
test(`completing computation job triggers SomeIntention`, async () ⇒ {
    const requestId = await fixtures.GivenAHeavyComputationRequestWasSubmitted();
    await fixtures.WhenJobIsCompleted(requestId);
    await fixtures.ThenSomeIntentionIsOnCommandBus();
})
```

Takeaways

- We read code much more often than we write
- Be the "good guy" - for others and yourself
- Express your intentions in enjoyable way
- Write tests by staring from spec, filling fixtures later

Takeaways

> We, as engineers, strive to have clear expectations, we ask questions to leave no room for guesses.

… and if we do expect that from others …

> Act accordingly to match your own expectations.

> Leave things behind you so no one curses you in pain.