TDD로 REST API 구현하기

20.11.21

TDD로 REST API 구현하기

TDD란?

- 테스트 주도 개발 (Test-Driven Development)

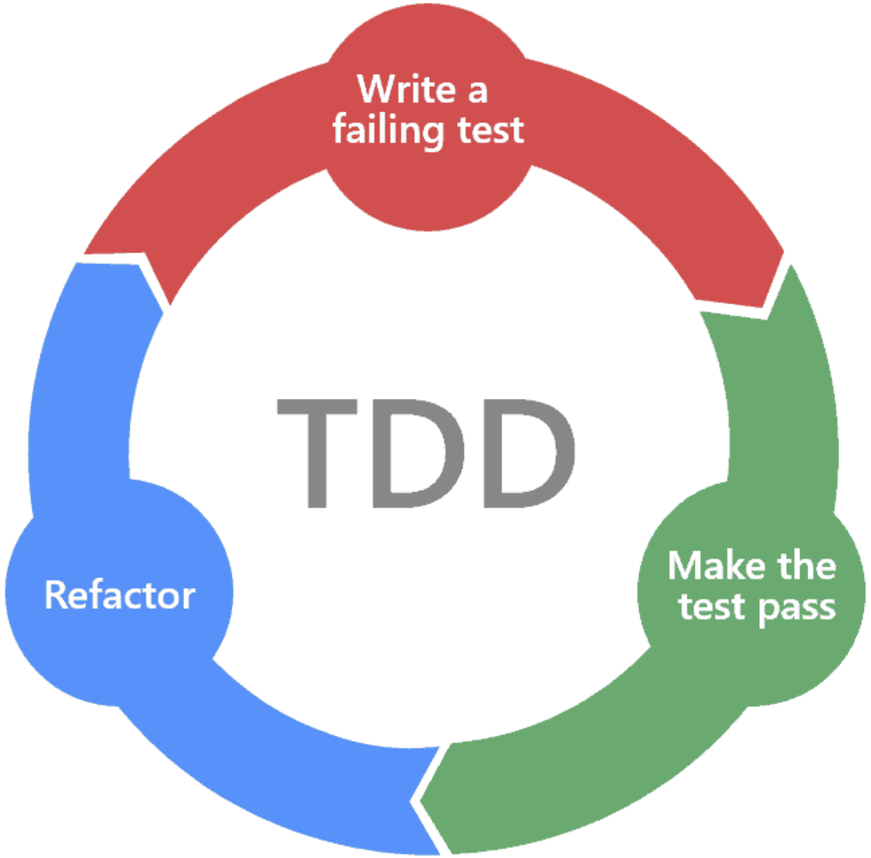
구현 -> 테스트

TDD

테스트 -> 구현

- Extreme programming 기법 중 하나

: 요구사항이 급변하는 분야에 적합한 개발 방법론


Write a
failing test
TDD
Make the
test pass
Refactor

TDD로 REST API 구현하기

목차

TDD로 REST API 구현하기

REST

- HTTP 주요 저자 중 한명인 로이 필딩이 자신의 박사 학위에 소개한 하나의 통신 아키텍쳐

- HTTP의 장점을 활용한 통신 방법

- Resource / http method로 api 작성

    Not restful : POST /users/delete/3

    Restful : DELETE /users/3

TDD로 REST API 구현하기

HTTP Status Codes : 2xx Success

Status Code	Name	Info
200	OK	성공
201	Created	리소스 생성 성공
204	No Content	리소스 삭제 성공

TDD로 REST API 구현하기

HTTP Status Codes : 4xx Error

Status Code	Name	Info
400	Bad Request	문법 오류 등의 잘못된 형식의 요청(클라이언트 잘못)
401	Unauthorized	비로그인 상태에서의 비허가된 접근
403	Forbidden	비/로그인 상태에서의 비허가된 접근
404	Not Found	리로스(DB, 경로)를 찾을 수 없음
409	Conflict	현 리소스에 의한 서버 내부의 충돌 ex. 중복된 아이디

TDD로 REST API 구현하기

Node에서 TDD 하기

- Mocha : Node TDD를 도와주는 프레임워크
- Should : 노드의 기본 검증 모듈인 assert를 대체
- Supertest : express 서버를 구동한뒤 Http 요청을 보내고 응답을 받는 등 api 테스트를 가능하게 해줌

```
describe("DELETE /users/:id", () => {
  desc (method) supertest.Test.expect(status: number, callback?: supertest.CallbackHandler): supertest.Test (+7
    it overloads)
      request(app)
      .delete('/users/4')
      .expect(204)
      .end(done);
    });

  });

  describe("실패시", () => {
    it('ID가 숫자형이 아닌 경우 400 반환', (done) => {
      request(app)
      .delete('/users/ten')
      .expect(400)
      .end(done);
    });
  });
});

describe("POST /users", () => {
  describe("성공시", () => {
    it("user 생성시 201 응답 및 name 반환", (done) => {
      request(app)
      .post('/users').send({name: 'Sangyeon'})   // send로 request의 body에 넣어줌
      .expect(201)
      .end((err, res) => {
        res.body.should.have.property('name', 'Sangyeon');
        done();
      })
```

TDD로 REST API 구현하기

TDD 장단점

- 장점

1. 동작하는 코드에 대한 자신감 (Clean Code that works)

2. 과도한 설계를 피하고, 간결성 증대

3. 실행 가능한 문서를 가짐 -> 코드 자체가 문서로서 기능

4. 디자인적 유연함, 의존성 관리가 편함

- 단점

코드 구현에 더 많은 비용이 소모