11.05 강의

공부 해야 할 것

e2e 테스트 , unit test

freezegun

property-based testing

트랜잭션

테스트 크기에 관하여

1. 지금까지 만든 테스트... (when 절이 다르다)
    1. 서비스의 테스트 : 서비스는 단순 함수 (plain function). 호출하는 것만으로 테스트 가능
    2. 라우터의 테스트 : url을 거쳐서 실제로 api 를 호출하는 테스트
2. coverage : 테스트 슈트를 수행하는 중에 1번 이라도 실행된 제품 코드 / 전체 제품 코드
3. 라우터 테스트의 커버리지가 높은 이유 : 서비스 코드 + 라우터 코드(분자)가 실행되기 때문에

- e2e 테스트와 unit test
    - 같은 용어여도 상황에 따라 조금씩 의미가 다를 수도 있습니다만)
- end to end 테스트 : 직접 api 호출을 하는 테스트
- e2e 테스트를 선호하는 이유 : "유저에 집중하면 나머지 것들은 따라온다"
    - "유저 입장에서는 공개 되어 있는 api 를 호출하는 것이 서버의 기능을 이용할 수 있는 유일한 방법" (서버 내의 클래스나 서버 안의 작은 함수를 직접 호출할 수 있는 방법은 없다.)
- e2e 의 단점 : 실제로 테스트가 터졌을 때 어디서 터졌는지 알기가 어렵다.
    - e2e 테스트는 테스트가 커버하는 코드의 양이 많기 때문에 에러가 발생했을때 "용의 선상" 에 오르는 코드도 많다. unit test 에 비해서 조사해야

하는 코드의 양이 많다. -> 고치는 시간이 오래 걸린다.
    - 간헐적으로 실패하는 테스트가 생겼을 때, 원인 찾기가 상대적으로(unit에 비해) 어렵다.
- (간헐적 말고) 시한폭탄처럼 터지는 테스트가 있다..?
- 시간을 멈추고 테스트 하고 싶은 요구 사항도 종종 만난다.
- 날짜와 관련된 연산을 해야되는데, 윤년을 끼고도 정상 동작 하는지 검증하고 싶다.
    - 윤년의 2월로 날짜를 고정해야 한다.
    - https://github.com/spulec/freezegun
    - 이럴 때 freezegun 이 빛을 발합니다.

```
@freeze_time("2012-01-14")
def test():
        assert datetime.datetime.now() ==
datetime.datetime(2012, 1, 14)
```

- freeze_time 빼먹고 테스트를 만들면 "시한 폭탄 테스트"가 만들어진다.

올바른 피드백 루프

- 개발 시작할 때마다 './test.sh' 전체 테스트를 수행했다.
    - 개발을 조금 하고 테스트하고.
- 올바른 피드백 루프는 빠르고, 믿을 수 있고, 실패한 지점을 바로 알 수 있다.

단위 테스트

- service 함수를 직접 호출한 것, (하나의 클래스, 하나의 메소드를 검증하는 테스트)
- 단위 테스트는 위의 "올바른 피드백 루프"의 조건을 만족한다.

테스트 피라미드

- Unit > Integration > e2e : 작은 테스트의 개수가 많아야 되고, 범위가 넓은 테스트는 적어야 한다.
-

강사님 생각이 바뀐.. (현재 e2e를 더 선호하게 된 이유)

- 구글 테스팅 블로그에서 unit test 를 더 많이 작성하라 했음에도 불구하고 e2e 를 더 선호하게 된 이유
- router 에서 validation 을 한다고 했을 때, service 에는 validation 을 거친 데이터만 들어가게 된다.
    - 하지만 단위 테스트에서 router를 bypass 하고 service() 를 직접 호출하는 경우에는, validation 을 실패하는 데이터도 넣을 수 있다.
    - 테스트를 작성하지 않은 다른 동료 개발자는 "잘못된 데이터" (given 절) 을 보고 혼란을 느끼게 된다.
    - 처음부터 router를 반드시 거치도록 개발을 했다면 잘못된 given 절을 가진 테스트를 만들 수 없다.
- 가장 외부의 인터페이스 (api) 에 의존하므로, 내부 서비스나 모델에 변화가 있더라도 상대적으로 적게 테스트를 수정할 일이 적어진다.
- 될 수 있으면 e2e 테스트만 작성한다 (httpx 로 api를 쏘는 테스트)
    - given 절에서는 api를 직접 쏘아서 테스트 데이터를 생성한다.
    - when 절에서도 테스트 대상이 되는 api를 쏴서 그 결과를 받아옵니다.
    - then 절에서만 데이터베이스에 직접 접근해서 원하는 결과가 데이터베이스에 들어갔는지 확인합니다.

validation 예제

```
# validation
for stop_word in {"!","#","@",":",";"}:
    if stop_word in article_id:
        raise HTTPException(
            status_code=400,
            detail=f"특수 문자는 허용되지 않습니다만, {stop_word}가 들어있습니다."
        )

return await service_get_article_and_comments(article_id)
```

property-based testing

- https://hypothesis.readthedocs.io/en/latest/

- property based 는 (엣지 케이스를 포함한) given 절을 우리를 대신해서 만들어 준다.
- 일정한 규칙을 가지는 무작위 입력을 테스트 대상에 넣고, 테스트 한다.

트랜잭션

왜 트랜잭션이 필요한가?

- 결제
  - 사용자가 주문하고 결제를 완료했다면...
  - 재고를 1 감소
  - (보통 실제 금원 출금은 PG 에서 일어난다.) (PG는 payment gateway = inicis, kcp, naver, kakao, toss)
  - 결제 내역 insert
- 만약 2번 이후 3번이 완료되기 전에 서버가 강제 종료가 된다면? -> 재고만 감소하게 됨. 이 문제를 해결하려면...
  1. 로그를 전부 뒤진다.
  2. 실제 창고를 전부 뒤진다.
  3. 모두 다 힘들고, 비용(인건비 포함) 이 많이 드는 작업
- 트랜잭션을 사용한다면
  - 모든 요청이 전부 성공하거나, 아니면 그 중 하나라도 실패했다면 모든 요청이 마치 없었던 것처럼 만들 수 있다. -> 원자성 (Atomicity)

트랜잭션 예제

```
SELECT * FROM articles;
SELECT * FROM comments;

BEGIN; -- 트랜잭션을 시작하겠다! --
INSERT INTO articles(id, author, title, body)
VALUES('test_id', 'test_author', 'test_title',
'test_body');
-- commit 되지 않은 변경 사항은, 다른 커넥션에게 보이지 않는다 --
INSERT INTO comments (id, author, content, article_id)
VALUES('test_comment_id', 'test_author',
'test_content','test_id');
```

```
ROLLBACK; -- 트랜젝션을 취소시킨다. 지금까지의 변경사항을 없었던 것으로 만든다.--

COMMIT; -- 트랜잭션의 완료를 의미한다. 모든 변경 사항을 실제로 데이터베이스에 적용한다.--
-- commit 이 성공한 이후에는 다른 커넥션에서도 변경사항이 보인다.

DELETE FROM articles WHERE id IS NOT NULL;
DELETE FROM comments WHERE id IS NOT NULL;

-- begin 에서 commit 까지가 하나의 트랜젝션이다. --
```

트랜젝션의 격리 수준

- 앞서 트랜젝션을 직접 해보았는데, 커밋되지 않은 변경사항은 다른 커넥션에서는 보이지 않았다.
- 이 behavior 는 격리 수준에 따라 다르다.
- mysql innodb는 4개의 격리 수준을 갖고 있따.
  - READ UNCOMMITED
  - READ COMMITED
  - REPEATABLE READ
  - SERIALIZABLE
- READ UNCOMMITED 처럼 격리 수준이 낮을 때는 Phantom Read 와 같은 현상이 발생한다.
  - A,B 커넥션이 있고 B 커넥션에서 A의 변경사항을 조회했는데, A가 롤백해 버리는 경우, B에서는 갑자기 안보이게 됨.
- 결론 = 격리 수준은 REPEATABLE READ 이상으로 쓰자. (이게 default 임)

트랜젝션의 관리

- 트랜젝션이 너무 빈번하게 일어나면 -> disk io 가 많아짐 -> 부하가 커집니다.
- 트랜젝션이 너무 길어지면 -> dead lock 의 위험성이 증가합니다.
  -> 트랜젝션을 적당한 시간 안에 끝나도록 하는게 좋다.

- mysql 의 innodb 엔진을 사용하는 경우 -> 모든 쿼리는 트랜젝션 안에서 실행 됩니다. (쿼리가 하나일 지라도)
- django 의 트랜젝션 default 설정 :
- https://docs.djangoproject.com/en/5.1/topics/db/transactions/
- django 의 default 설정은 auto commit 이다.
    - auto commit 은 각 쿼리를 쿼리가 실행되자마자 바로 커밋한다.
- 단, transaction 이 active 하지 않을 경우에만 auto commit 한다.
- tortoise 에서도 비슷합니다. :https://tortoise.github.io/transactions.html
    - context manager : with
    - decorator : '@'

```
async with transactions.in_transaction():
        article = await Article.get_one_by_id(article_id)
        comments = await
Comment.get_all_by_article_id(article_id)
```

여담 inicis를 쓰지 마라 = 장애가 나도 전화를 안받는다.
- pg 사는 어디서 수익? -> 수수료 -> 수수료를 내는 (이커머스)회사는 중요한 고객
- pg 사에는 비상 연락망이 있다. -> pg 계약을 맺었다면 함부로 결제수단을 비노출 시키면 안되는데,...
- 장애가 났다면 결제 수단을 비노출 해야 한다.
- 결제 내역을 insert
- kcp, naver,kakao, toss

gunicorn 간단 소개

- https://gunicorn.org/
-

FastAPI 는 정말 빠른가? 증명

- WSGI : Web Server Gateway Interface -> 웹서버(gunicorn랑 파이썬 프레임워크(django, fastapi) 사이를 이어주는 인터페이스.
    - async WSGI 를 ASGI 라고 한다. (uvicorn)

        - XSGI 양식을 지켜서 프레임 워크를 만들어서 gunicorn, uvicorn 등이랑 소통이 가능합니다.
    - UNIX : Linux 의 할아버지. unix 에서 돌아간다는 말은, windows 에서 안돌아간다.
    - port 를 listen 한다는 의미 -> 내가 앞으로 포트 8000을 사용하고 싶다. (이 포트를 사용해서 계속해서 요청을 받겠다) 라고 운영체제에 요청하는 일.
        - 요청을 받았기 때문에 거절을 할 수도, 승낙을 할 수도 있다.
        - 언제 거절을 하느냐?? -> 다른 프로세스가 이미 해당 포트를 listen 하고 있을 경우 운영체제는 거절한다.
        - 만약에 쓰이고 있지 않다면 승낙하고, 포트 8000에 bind 된다.

pre-fork worker model :
- fork 의 의미 -> bind 에 성공한 후에 master 프로세스는 스스로를 복제해서 (이 복제를 fork 라고 합니다.) worker process 를 생성합니다.
- 실제 요청을 받아들이고 응답을 만들어 내는 것은 worker 프로세스입니다.

- master 가 하는일 : worker가 알 수 없는 이유로 죽었을 때 새로 만듭니다. 혹은, 어떤 시그널을 받아서 의도적 재시작 (역시 새로 만듭니다.) 합니다.