10.31 강의

공부해볼 것.

Gather?

asyncio?

await

직렬 병렬

flask 마이그레이션 이름 : alembic
django 마이그레이션 이름 : south
tortoise 마이그레이션 이름 : aerich

내 MAC이 Docker를 키자마자 이륙할것 같다. 비행기 소리가 난다.
https://orbstack.dev/
orb stack 이 메모리를 더 적게 사용한다.

database_settings.py

- aerich init -t app.configs.database_settings.TORTOISE_ORM
- 설정을 aerich init에 전달한다.

aerich 를 설치하면 pyproject.toml 에 설정이 추가된다
-> aerich init-db 를 해준다.

GIT COMMIT

- git commit 은 stage에 올라간 변경 사항만 commit 한다.
- stage에 변경사항을 올리기 위해서는 git add를 사용한다.
- pycharm 에서 "untracked file" 은 "빨간색"으로 표시된다.
- pycharm 에서 "commit" 된 적이 있고, 새로운 변경 사항이 있는 파일은 "파란색" 으로 표시된다.

Migration

- 마이그레이션 하기 전에 "반드시" 해야되는 일 -> 쿼리 확인
  -> 내가(이 마이그레이션이) 실제로 어떤 sql을 실행하는지 반드시 확인해라!
  -> sqlmigrate

tortoise-aerich 는 db에서 default가 있는 부분에 직접 입력할 필요가 없다.

mermaid : 다이어그램을 그려주는 엔진
sequenceDiagram
autonumber
명수 ->> 진강 : 오늘 아침 뭐먹었어?
진강 -->> 명수 : 나는 계란 간장 밥 먹었어.

실무에서 많이쓴다

- draw.io로 직접 그리던가,
- mermaid 로 코드를 다이어그램으로 바꿀 수 있습니다.
- sqeuenceDiagra을 그릴 때는 꼭 autonumber 를 활성화 합시다
  - autonumber를 활성화하면 모든 화살표에 번호가 자동으로 붙습니다
  - 화살표 번호를 가지고 개발자들끼리 소통하기가 편리합니다
  - '>>' 꺽쇠를 두번 쓰면 화살표 방향이 보이고, 한 번 쓰면 안보입니다
  - '-' 대쉬를 한 번 쓰면 실선, 2번 쓰면 점선이 됩니다.
    - 요청은 실선, 응답은 점선으로 표시하는 것이 관례입니다.

직렬 요청과 병렬 요청

- 기존 django (sync 방식) 에서는 쿼리르 하면, 그 쿼리의 결과가 돌아올 때 까지 서버는 **기다린다.
  - 기다린다는 것은 곧, 시간을 낭비한다는 것을 의미한다.
- 게시글과 그 게시글의 댓글을 쿼리하는 상황을 가정해보면, django는 두 개의 쿼리를 "직렬"로 실행하고 2번 기다린다.
- 만약에, 게시글과 댓글을 병렬로 쿼리를 할 수 있다면 대기 시간을 줄일 수 있다. (더 효율적이다)

sync 방식의 문제점

- latency: 지연 -> 어떤 작업을 수행하는데 시간이 얼마나 걸리느냐
- cpu 의 latency 는 cycle 로 측정합니다.
    - 컴퓨터 사보신 분들은 cpu "클럭(clock)"이 높을 수록 좋다는 것을 알텐데,
    - cpu 1 clock -> 1 cycle 입니다. (컴퓨터를 잘 모르는 사람은 어렵다...)
- L1, L2 캐시 접근 -> 엄청나게 빠름. 메모리에 접근 -> 빠름
- 반대로, 느린 연산 -> 디스크에 접근: 41,000,000 cycle, 네트워크에 접근: 240,000,000 cycle
- network 호출을 하고 응답이 돌아올 때 까지 그냥 기다리는 것은 -> 매우 비효율적
- 기존의 query set -> 'Article.objects.get()' 은 샌프란시스코에서 도쿄까지 날아가는 동안 그냥 기다리는 것과 같다.
- I/O -> input, output의 약자

병렬 요청은 언제 가능할까?

- 모든 직렬 요청을 병렬 요청으로 바꿀 수 있는건 아니다.
- 불가능 한 경우는 언제?
    - 직전 응답에 의존하는 경우
    - A, B 요청 2개를 해야할 때, B 요청을 하기 위해서 A의 응답이 필요한 경우
    - -> 어쩔 수 없이 직렬로 요청해야 한다.

DTO
리스폰트 json을 리턴하게 되는데 결과 안에 뭐가 들어가냐.
response dto 는 결과 json 에 어떤 값이 들어갈지 결정한다.

fastapi 는 response dto를 보고
response spec 을 자동으로 생성합니다. (자동으로 swagger가 만들어짐)

DTO -> Data Transfer Object -> 정보를 담고 있는 객체
-> 정보를 수정한다면 DTO 가 아닙니다. 정보를 담고만 있어야 DTO 입니다.

fastapi 에서 dto 는 보통 pydantic 의 BaseModel 을 상속하는 클래스로 만듭니다.

함수가 dict 를 리턴하도록 하지 마세요.(99.9% 의 상황에서는 dict 대신에 dto를 사용할 수 있습니다.)

don't:
return {"abc", "def"}

do:
return MyDto(abc="def")

await:

- 비동기 호출을 합니다. (async def 로 시작하는 함수를 호출합니다.)
- 이벤트 루프에게 이 함수를 호출해 달라고 등록합니다.
- 내가 등록한 함수가 호출 완료되기 까지 기다립니다.
    - 만약 db 호출이라면 결과가 돌아올 때까지 기다리는 것을 의미합니다.
- 이벤트 루프 : 여러가지 비동기 함수들을 실행해주는 친구.(무한 루프)

Breadcrumbs

- 한국말 -> 빵조각
- 빵조각 -> 헨젤과 그레텔 -> 길찾는데 (들어온 길로 다시 돌아가기 위해서)

Structure View

- pycharm 에서 'command + 7' 을 누르면 structure view를 볼 수 있다.
- structure view 에서는 클래스가 가진 모든 멤버 변수와 메소드가 일목요연하게 정리되어서 보여진다.

직전 커서로 돌아가는 단축키 - `Command + option + 화살표 왼쪽` 을 누르면 직전 커서로 돌아간다. (굉장히 많이 쓰는 단축키)

asyncio.gather( ):
- 비동기 호출을 "Concurrent" 하게 진행하는 2가지 방법 중 하나입니다.
- gather(), as_completed() 가 있다.
- gather() 를 호출할 때는 다수의 코루틴을 전달합니다.
- 모든 코루틴이 실행 완료됐을 때 까지 대기한 후, 다 되었다면 순서대로 반환합니다.

병렬 처리 방식의 차이

parallel 과 concurrent 의 차이:

- 은행에 가서 통장도 만들고,
- 치킨집에서 치킨 포장도 하고,
- 마트에서 우유도 사야한다.
- 총 3개의 task 존재

  parallel의 경우 :

  - 사람 3명이 각각 1명은 은행에 가고, 1명은 치킨집에 가고, 한 명은 마트에 간다.
  - 진짜로 task 가 병렬로 실행된다.

  concurrent 의 경우
  - 1 명이
  - 은행에 먼저 가서 번호표를 뽑고
  - 치킨집에 가서 치킨을 주문시켜 놓은 뒤
  - 마트에 가서 우유를 사고
  - 치킨집에서 치킨을 포장해서 나오고
  - 은행에 가서 통장을 만들고
  - 귀가

-. 더 적은 자원으로 여러 개의 task 를 수행할 수 있는 것은 -> concurrent
-. 하지만 마트에서 뭔가 사고가 나서 우유를 사는데 지연이 되었다면 (마트에 갇혔다..?)
-. 치킨과 은행 task 도 함께 지연 된다.
-. parallel 은 사람이 진짜 3명이기 때문에 마트에 갇히는 일이 발생해도 치킨과 은행은 영향을 받지 않는다.

-. 사람 -> cpu core를 말합니다. (혹은 process 를 의미한다고도 볼 수 있다.)
-. task 는 IO 요청을 말합니다. (데이터베이스 호출, http 호출)

-. 은행, 치킨집, 마트 등은 IO 요청을 받아들이는 주체입니다. (데이터 베이스, 외부 서버)