ORM과 N+1 쿼리 / Django는 왜 망할 수도 있을까?

Instructors:
김찬욱 @narayo9

2022.11.26 (화) 19:30

Assignment 4 리뷰

목차

WA LE
STUDIO

Is ORM Silver bullet?

ㅋㅋㅋ

ORMHate: 지난번에 말씀드린 마틴 파울러 아저씨

> Essentially what you are doing is synchronizing between two quite different representations of data, one in the relational database, and the other in-memory.
>
> As you might have gathered, I think NoSQL is technology to be taken very seriously. If you have an application problem that maps well to a NoSQL data model

Are there good reasons not to use an ORM?

> ORMs are useful for automating the 95%+ of queries where they are applicable
>
> First off - using an ORM will not make your code any easier to test, nor will it necessarily provide any advantages in a Continuous Integration scenario.

많은 사람들이 SQL을 배우기 위해 ORM을 선택한다. SQL이 아닌 ORM 자체를 배우게 된다. 사람들은 SQL은 배우기 어렵고, ORM을 배우면 하나의 언어만 사용하여 애플리케이션을 작성할 수 있다는 믿음을 갖곤 한다. 언뜻 보기에 이는 맞는 것 같다. 그러나 ORM은 애플리케이션의 나머지 언어와 동일하게 작성되지만, SQL은 완전히 다른 문법을 가진 언어다.

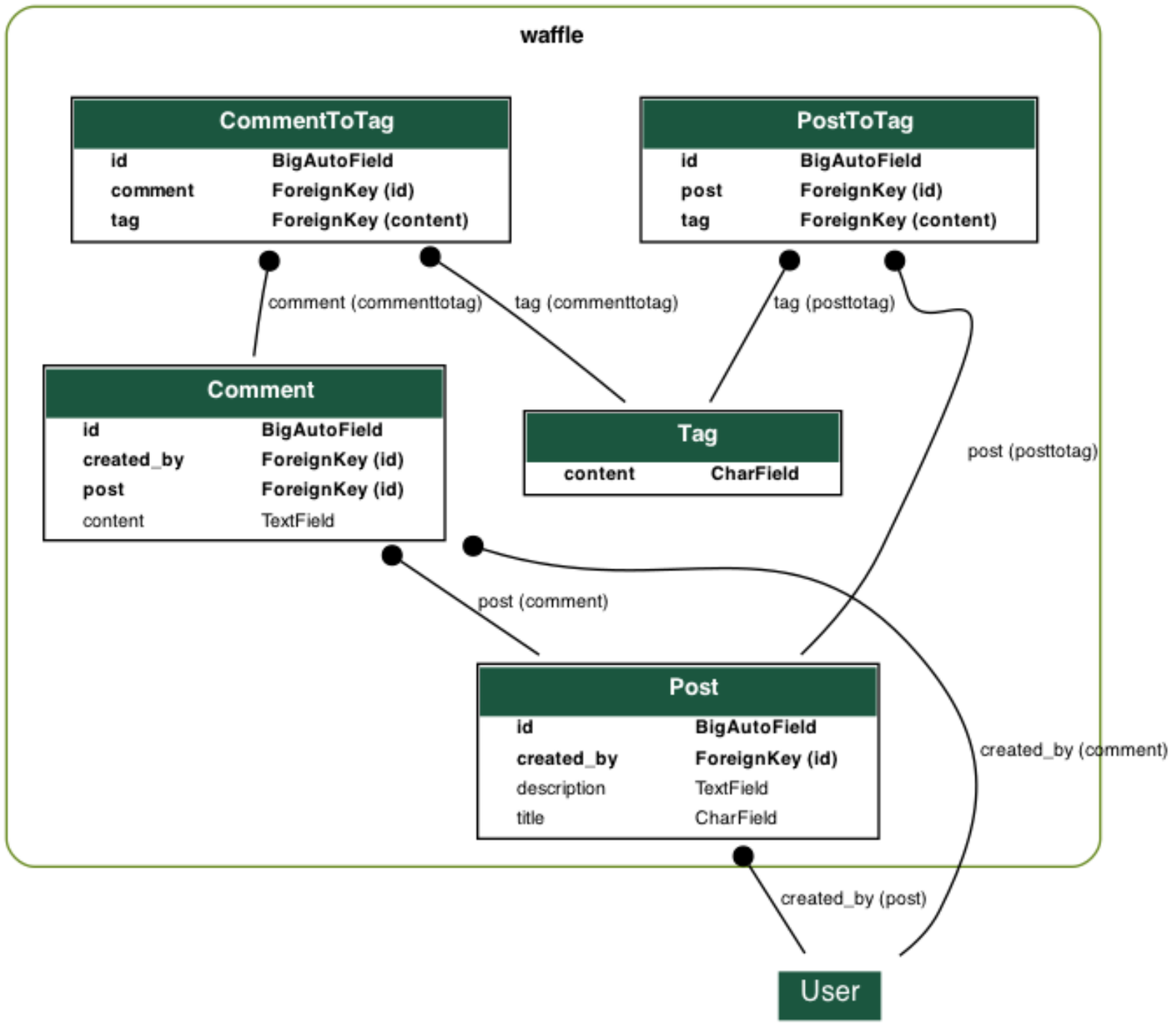
ORM은 피할 수 없는: N+1 쿼리

쿼리 1번으로 N건을 가져왔는데, 관련 컬럼을 얻기 위해 쿼리를 N번 추가 수행하는 문제

쿼리 결과 건수마다 참조 정보를 얻기 위해 건수만큼 반복해서 쿼리를 수행하게 되는 문제

DB쿼리 수행비용(횟수)이 크기 때문에, **eager 로딩** 등의 방법으로 해결하는 것이 권장됨

N+1 쿼리: 예시와 함께




```
[
  {
    "id": 1,
    "created_by": {
      "id": 5,
      "first_name": "찬욱",
      "last_name": "김"
    },
    "tags": [
      {
        "content": "와플"
      }
    ],
    "title": "세미나",
    "description": "뭐하는거니",
    "comment_set": [
      {
        "id": 1384,
        "created_by": {
          "id": 4,
          "first_name": "대용",
          "last_name": "정"
        },
        "post": 7,
        "content": "잘하고있어",
        "tags": [
          {
            "content": "넌최고야"
          }
        ]
      }
    ]
  },
  …
]
```


WAFFLE STUDIO

N+1 쿼리: 예시와 함께

Python

```
posts = Post.objects.all()

serializer = PostSerializer(posts, many=True)
serializer.data

post.created_by
post.comment_set.all()

comment.tags[1].created_by
```

SQL

```
SELECT "waffle_post"."id",
       "waffle_post"."title",
       "waffle_post"."description",
       "waffle_post"."created_by_id"
  FROM "waffle_post"

SELECT "auth_user"."id",
       "auth_user"."password",
       "auth_user"."last_login",
       …
       "auth_user"."is_active",
       "auth_user"."date_joined"
  FROM "auth_user"
 WHERE "auth_user"."id" = '5'

SELECT "waffle_comment"."id",
       "waffle_comment"."post_id",
       "waffle_comment"."created_by_id",
       "waffle_comment"."content"
  FROM "waffle_comment"
 WHERE "waffle_comment"."post_id" = '1'
```

X POST 개수

X POST 개수

N+1 쿼리: 예시와 함께(Better)

Python

```
posts = Post.objects.select_related(‘created_by’).prefetch_related('tags', ‘comment_set’)


serializer = PostSerializer(posts, many=True)
serializer.data

post.created_by
post.comment_set.all()

comment.tags[1].created_by
```

SQL

```
SELECT "waffle_post"."id",
       "waffle_post"."title",
       "waffle_post"."description",
       "waffle_post"."created_by_id",
       "auth_user"."id",
       "auth_user"."password",
       "auth_user"."last_login",
…
       "auth_user"."is_active",
       "auth_user"."date_joined"
FROM "waffle_post"
         INNER JOIN "auth_user"
                    ON ("waffle_post"."created_by_id" = "auth_user"."id")


SELECT "waffle_posttotag"."post_id",
       "waffle_tag"."content"
FROM "waffle_tag"
         INNER JOIN "waffle_posttotag"
                    ON ("waffle_tag"."content" = "waffle_posttotag"."tag_id")
WHERE "waffle_posttotag"."post_id" IN ('1', '2', '3', '4', '5', '6', '7', '8',
'9', '10')
```

EAGER LOADING!

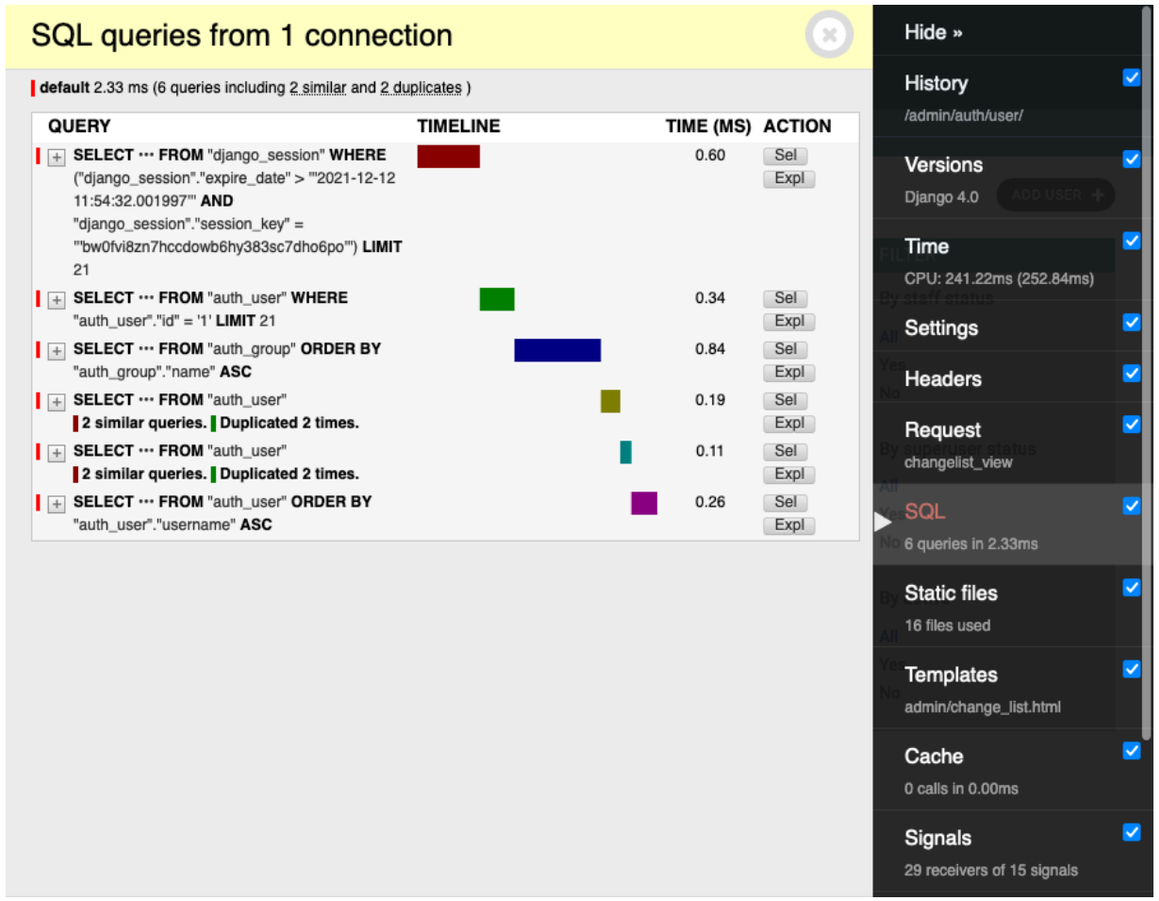
WAFFLE STUDIO

Django-debug-toolbar

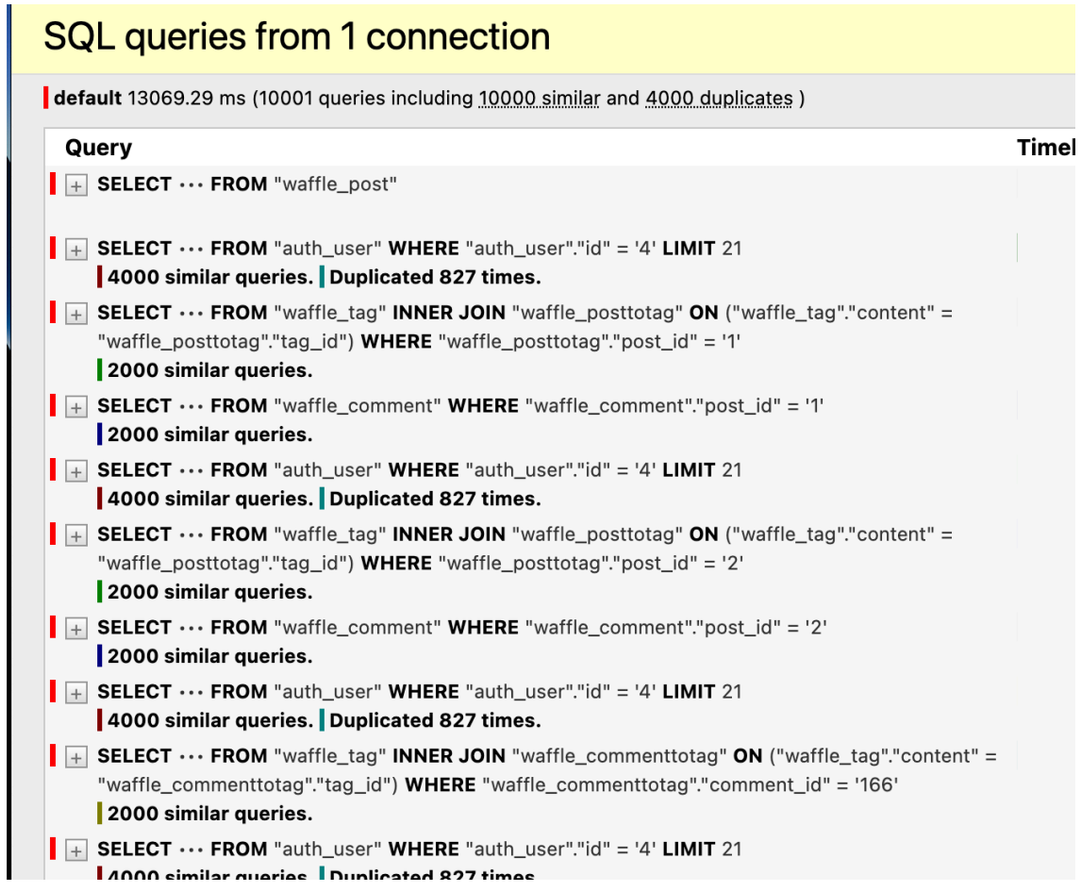
아주 유용해요!

Cpu time, sql, static files, …

Django Debug Toolbar는 현재 요청/응답에 대한 다양한 디버그 정보를 표시하고 클릭하면 패널 내용에 대한 자세한 정보를 표시하는 구성 가능한 패널 집합입니다.



실습: 직접 확인해보자



WAFFLE STUDIO

인스턴스 당 하나: select_related

Returns a QuerySet that will “follow” foreign-key relationships, selecting additional related-object data when it executes its query.

쿼리 개수는 그대로
1:n 관계(역)일수도, 1:1 관계일수도

```python
class Post(models.Model):
    …
    created_by = models.ForeignKey(User, on_delete=models.PROTECT)

   …


Post.objects.select_related('created_by')
```

```
SELECT "waffle_post"."id",
       "waffle_post"."title",
       "waffle_post"."description",
       "waffle_post"."created_by_id",
       "auth_user"."id",
       "auth_user"."password",
       "auth_user"."last_login",
…
       "auth_user"."is_active",
       "auth_user"."date_joined"
FROM "waffle_post"
         INNER JOIN "auth_user"
                    ON ("waffle_post"."created_by_id" = "auth_user"."id")
```

인스턴스 당 여러개: prefetch_related

Returns a QuerySet that will automatically retrieve, in a single batch, related objects for each of the specified lookups.

쿼리 개수 한 개 추가해서 해결!
1:n 관계일수도, n:n 관계일수도

select_related 문제는 prefetch_related로 풀 수 있다(not best optimized, but usable)
prefetch_related 문제는 select_related로 풀 수 없다

```python
class Post(models.Model):
    …
    created_by = models.ForeignKey(User, on_delete=models.PROTECT)

   …


User.objects.prefetch_related('post_set')
```

```sql
SELECT "auth_user"."id",
       "auth_user"."password",
       "auth_user"."last_login",
       "auth_user"."is_superuser",
       "auth_user"."username",
       "auth_user"."first_name",
       "auth_user"."last_name",
       "auth_user"."email",
       "auth_user"."is_staff",
       "auth_user"."is_active",
       "auth_user"."date_joined"
  FROM "auth_user";


SELECT "waffle_post"."id",
       "waffle_post"."title",
       "waffle_post"."description",
       "waffle_post"."created_by_id"
  FROM "waffle_post"
 WHERE "waffle_post"."created_by_id" IN ('1', '2', '3', '4', '5');
```

Prefetch object

Prefetch 하는 queryset 건들기

prefetch하는 queryset 조차 prefetch

filter된 queryset에서만 prefetch

특정 field만 가져오기 (.only)

Prefetch 되는 attr을 control하고 싶을 때

~~그냥 raw query를 날리면 안될까?~~

Prefetch object (con't)

```
Post.objects.select_related('created_by').prefetch_related(
    'tags',
    Prefetch(
        'comment_set',
        queryset=Comment.objects.select_related('created_by').prefetch_related('tags')
    )
)
```

재사용이 불가능

어떤 prefetch_related가 필요할지 결정은 serializer 단에서 결정된다

⇒ serializer 단에 정리?

DRY prefetching (tips)

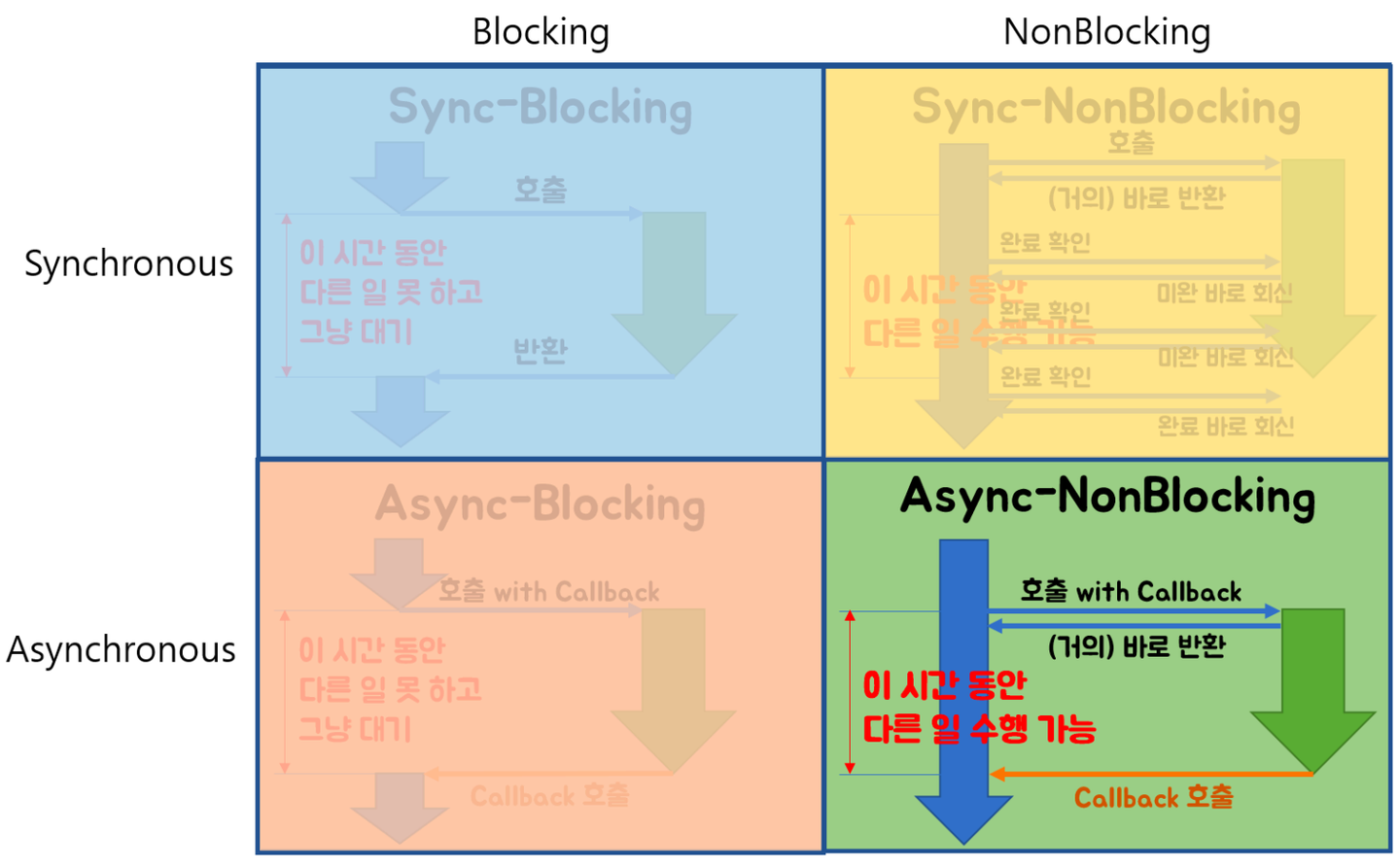
```python
class CommentSerializer(serializers.ModelSerializer):
    created_by = UserSerializer(read_only=True)
    tags = TagSerializer(read_only=True, many=True)

    @staticmethod
    def prefetch_queryset(queryset: QuerySet):
        return queryset.prefetch_related('comment_set')

    class Meta:
        model = Comment
        fields = ('id', 'created_by', 'post', 'content', 'tags')


class PostSerializer(serializers.ModelSerializer):
    created_by = UserSerializer(read_only=True)
    tags = TagSerializer(read_only=True, many=True)
    comment_set = CommentSerializer(read_only=True, many=True)

    class Meta:
        model = Post
        fields = ('id', 'created_by', 'tags', 'title', 'description', 'comment_set')

    @staticmethod
    def prefetch_queryset(queryset: QuerySet):
        return queryset.prefetch_related(
            Prefetch('comment_set', queryset=CommentSerializer.prefetch_queryset(Comment.objects.all())))
```

Django는 왜 망할수도 있을까?

How about FastAPI?

Non-blocking development

Blocking-NonBlocking-Synchronous-Asynchronous



WA LE STUDIO

What it takes to get Django go full async?

> A lot of young devs are constantly chomping at the bit to be on the cutting edge 24/7 but the truth is living and coding this way ultimately is counterproductive and just a stressful, security-risk-ridden shitshow.

Just to clarify - it adds an async interface to the ORM, but that is **still a wrapper** around the synchronous ORM. It's a good first step, but nowhere near async all the way down to the database interface yet.

Async is not a magic bullet. It can help with "some* things. For a lot of things, it won't make that much of a difference. In the end, someone/something has requested data. They'll always have to wait for a reply, and async won't make that data appear any quicker.

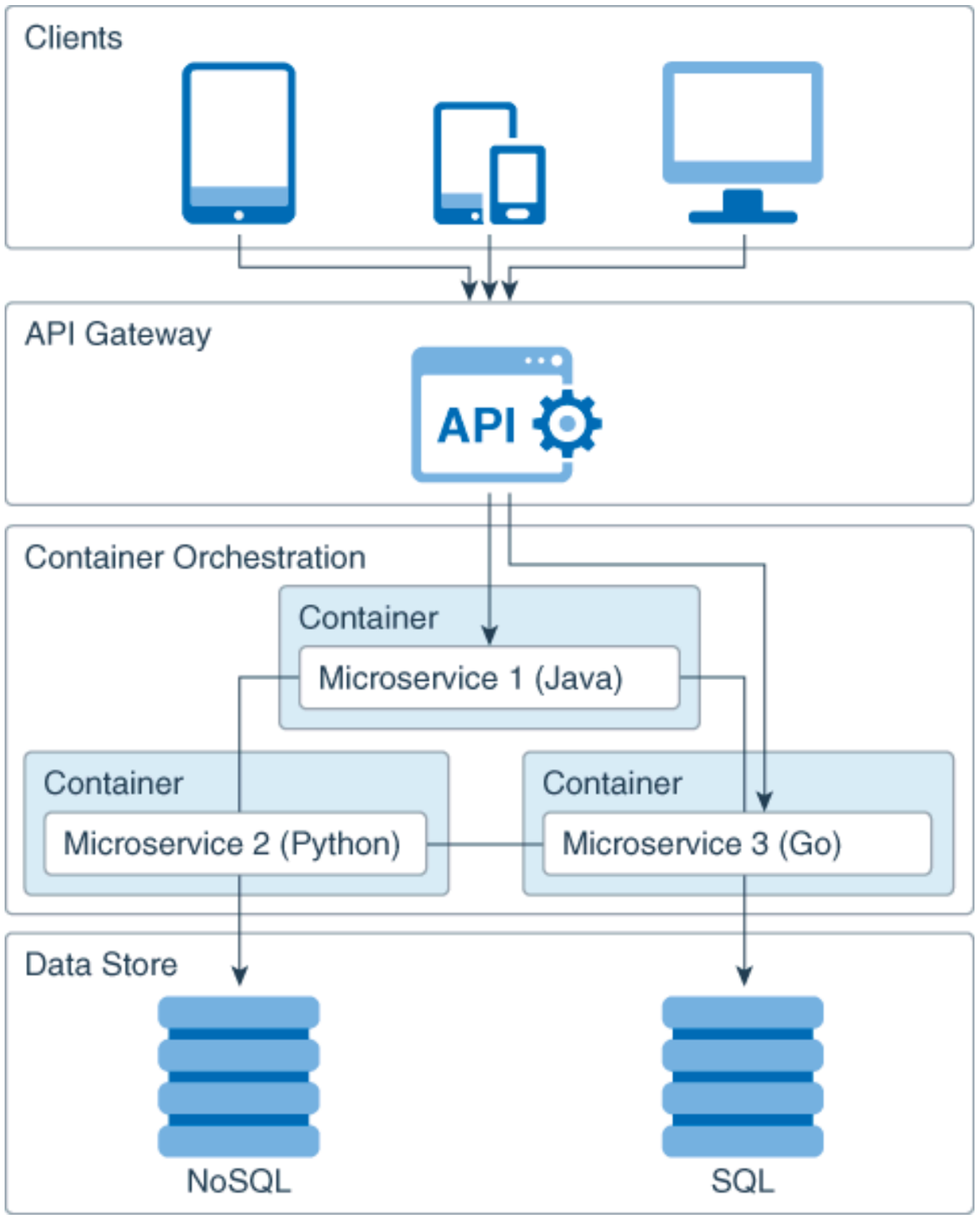
FastApi if use case is fully async and you want a Python Framework

Micro service architecture

Django Forms, Widgets,
Templates,
Authentication,
And Something just
works...

We might not need it

Especially on MSA



Type-safe development

Django is hard to typing

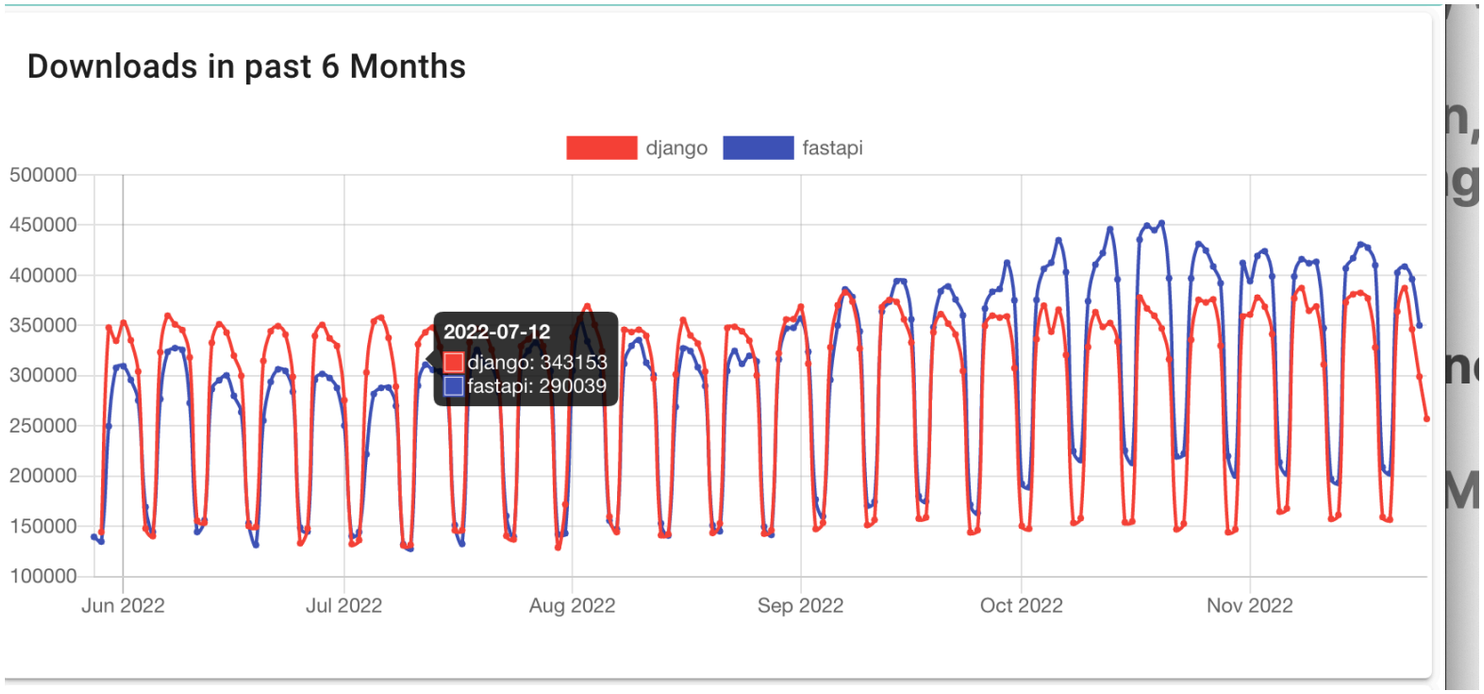
It is impossible: QuerySet[User]

Many dynamic parameters

WA LE
STUDIO

Here FastAPI Arises...

No features at all < Right in the middle < Too much features (for most use cases)



WA#LE STUDIO

FastAPI: Ad

*"[...] 저는 요즘 **FastAPI**를 많이 사용하고 있습니다. [...] 사실 우리 팀의 **마이크로소프트 ML 서비스** 전부를 바꿀 계획입니다. 그중 일부는 핵심 **Windows**와 몇몇의 **Office** 제품들이 통합되고 있습니다."*

Kabir Khan - **마이크로소프트** (ref)

---

*"**FastAPI** 라이브러리를 채택하여 **예측**을 얻기 위해 쿼리를 실행 할 수 있는 **REST** 서버를 생성했습니다. [Ludwig을 위해]"*

Piero Molino, Yaroslav Dudin 그리고 Sai Sumanth Miryala - **우버** (ref)

---

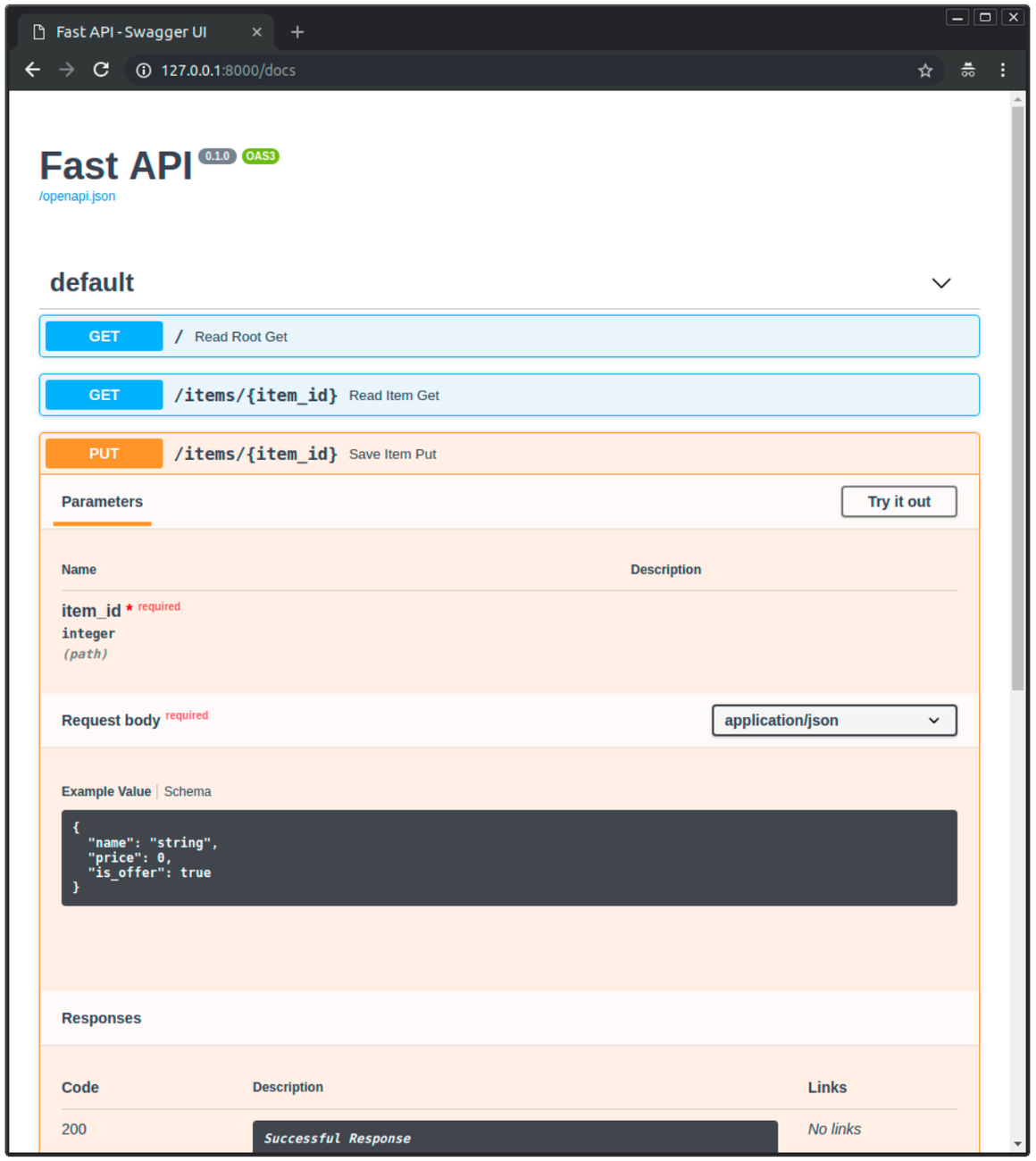
*"**Netflix**는 우리의 오픈 소스 배포판인 **위기 관리** 오케스트레이션 프레임워크를 발표할 수 있어 기쁩니다: 바로 **Dispatch**입니다! [**FastAPI**로 빌드]"*

Kevin Glisson, Marc Vilanova, Forest Monsen - **넷플릭스** (ref)

실습: newEra 프로젝트 켜보면서 확인하기

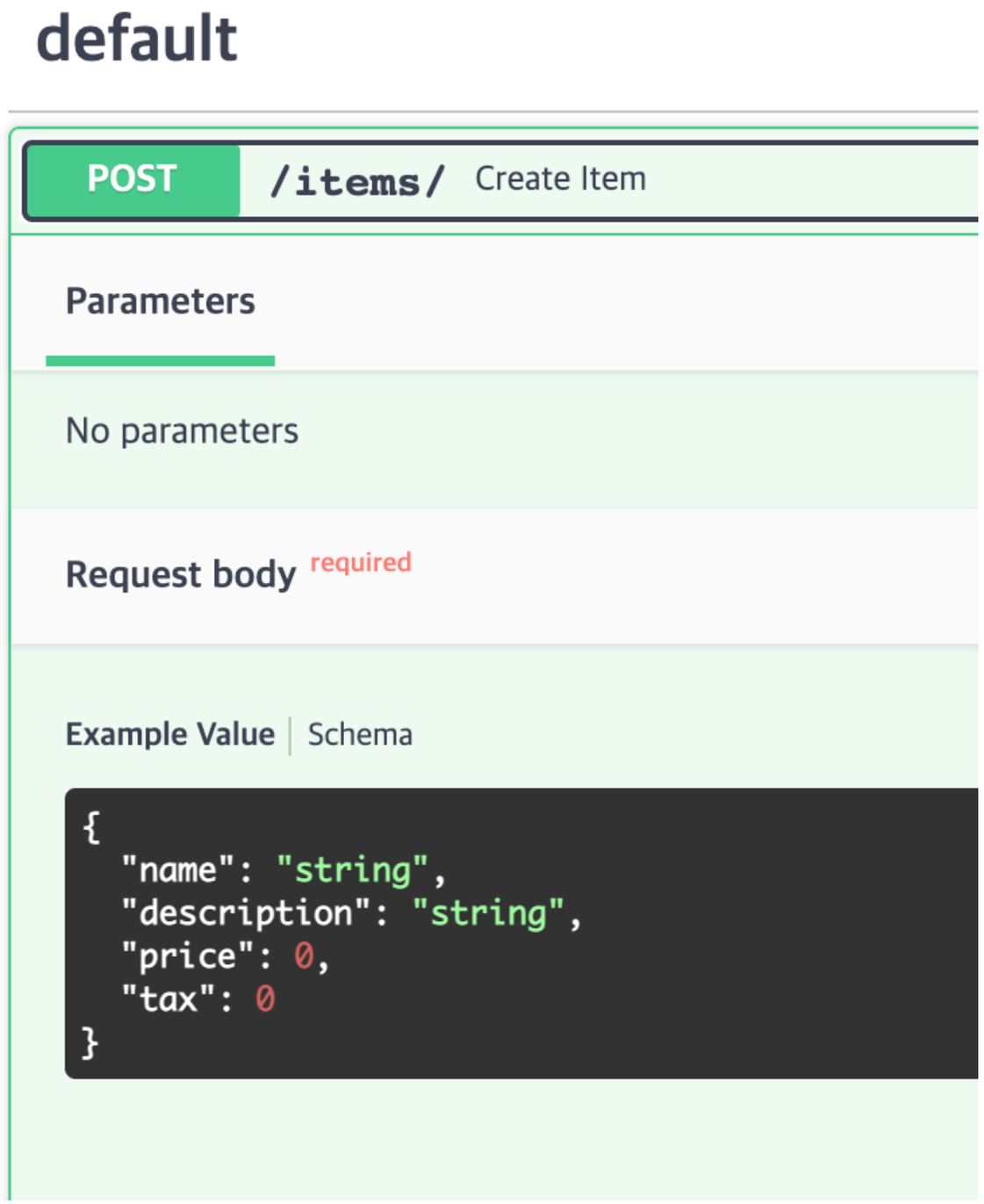
Feature: open api

/docs, /redoc 확인



Feature: type-safe validation with pydantic

pydantic_validation.py



Feature: background tasks

background_tasks.py

Feature: tests with pytest

test_main.py

No orms?

Business logic layer를 pydantic으로 구성하고 db layer 없이도 테스트가 가능하도록

Feature: async / non-blocking?

Already works!

Uvicorn: asgi web server

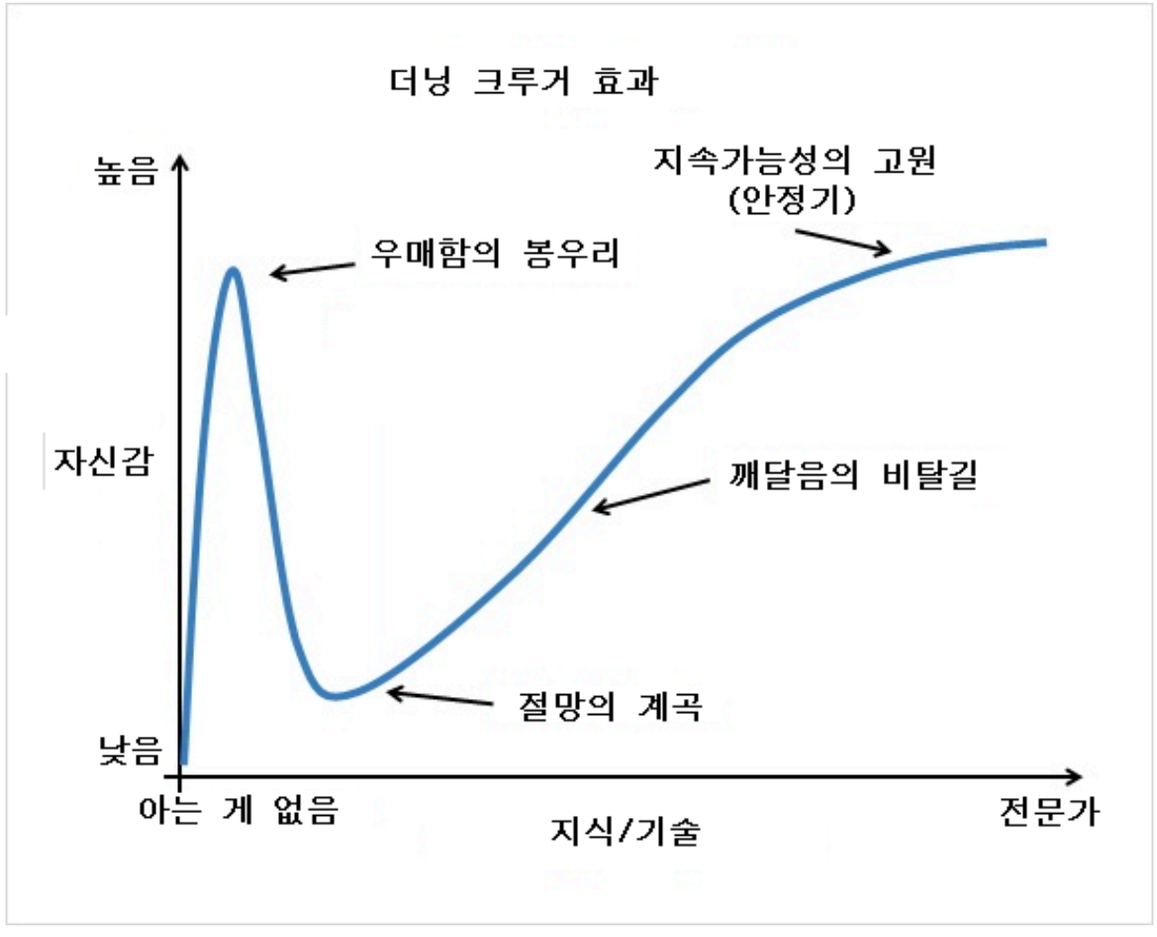
Assignment 5 소개

기말고사 잘 보시고 크리스마스 전까지!
~~양이 아주 적습니다~~
링크

Outro: 우리가 아직 모르는 것들

해치웠나...?

그동안 굉장히 깔끔하고
아름다운 것만 다뤄왔음



우리가 아직 모르는 것들

사실은 정말 중요한 많은 모델링 경험
    현실의 문제를 해결 가능한 문제로 바꾸는 것

짜증나는 troubleshooting을 해결하는 노하우
    예: 첫 날 postgres docker

적당히 legacy code와 타협하는 법

많은 infra / devops 이슈
    AWS: 토이 프로젝트에서 쓰셔야 됨. 문서 화이팅!
    Kubernetes

수업에서 꼭 다루고 싶었던

Celery: django와 함께 많이 사용하는 task queue (ex: background tasks)

Docker: 어느 환경 / OS에서도 application을 실행할 수 있도록 해줌. You must know how to write Dockerfile. What is container? Image?

Django's cache backend: memcached, redis

Django's transactions:

```python
from django.db import transaction

@transaction.atomic
def viewfunc(request):
    # This code executes inside a transaction.
    do_stuff()
```

하지만 압도되지는 마세요

사실 저도 잘 모릅니다

취업하고 멋져보이는 시니어 개발자도 사실 잘 모릅니다

왜 필요한지 인덱스 항상 늘려놓기 +
보통 문제가 생겼을 때 하나씩 적용해봐도 늦지 않다

주니어 개발자의 가장 큰 실수: Overengineering!

그래도...

"겨우" 동아리에 일주일에 10시간씩 6번씩이나,

"그" 대학에서 경쟁률이 2:1을 넘는 server programmers 전형 통과

고생하셨습니다...

연락주세요!

Instagram @narayo9

이력서 / 면접...
토이프로젝트 기술적 고민
기타 그냥 궁금한 것들

세미나 불만 / 발전적인 의견: 마지막 익명 피드백

WA LE
STUDIO

Questions?