WA#LE
STUDIO

Clean software in django

Instructors:
김찬욱 @narayo9

2022.11.08 (화) 19:30

Long time no see

Django seminar 후반 소개

수업 일정 및 자료

~~드랍하지 말아주세요 ㅠㅠ (19명 남음)~~

WA LE
STUDIO

Project 1 리뷰

WA LE
STUDIO

목차

WA LE
STUDIO

Clean software: SOLID principles

요 책 part 2 기반으로 작성되었습니다

설계의 악취를 겪으셨나요

시스템을 변경하기가 어렵다. 변경하려면 시스템의 다른 부분들까지 많이 변경해야 한다.

변경을 하면 시스템에서 그 부분과 개념적으로 아무런 관련이 없는 부분이 망가진다.

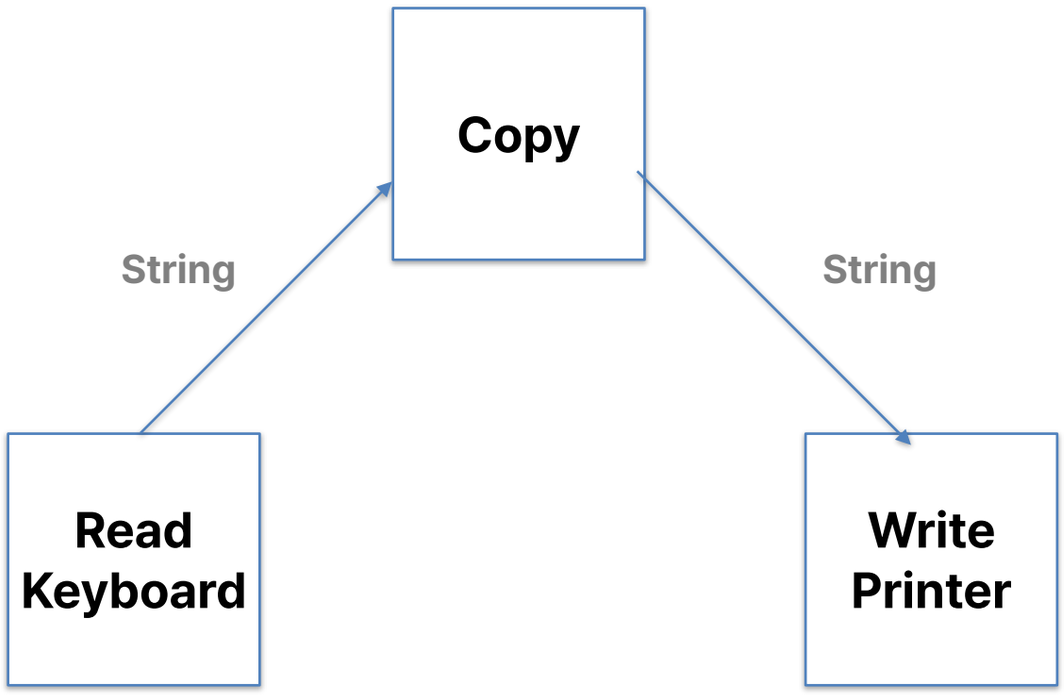
설계의 악취를 겪으셨나요(cont.)

시스템을 다른 시스템에서 재사용할 수 있는 컴포넌트로 구분하기가 어렵다.

옳은 동작을 하는 것이 잘못된 동작을 하는 것보다 더 어렵다.

예시: Copy 프로그램 만들기

팀장: 키보드로 문자를 복사하는 프로그램을 작성해주세요



WA LE
STUDIO

예시: Copy 프로그램 만들기

```
void Copy() {
    int c;
    while ((c = RdKbd()) != EOF)
        WrtPrt(c);
}
```

이 함수를 사용하는 30개의 예시가 생겼다.

요청: 키보드 말고 보이스 입력으로도
문자를 읽을 수 있도록 해주세요

예시: Copy 프로그램 만들기

```
bool voiceFlag = false; // MARK: 이 플래그를 설정하고 사용해주세요
void Copy() {
    int c;
    while ((c = (voiceFlag ? RdVoice() : RdKbd()) != EOF)
        WrtPrt(c);
}
```

요청: 보이스 output으로 문자를 쓸 수 있도록~

SOLID 원칙: 들어가기에 앞서

사실은 아름다운 OOP를 지향하기 위한 원칙

Python은 빡센 OOP 언어가 아님(no interface, duck typing)

그럼에도 불구하고 가치있다.

SRP: 단일 책임 원칙

Single responsibility principle

하나의 클래스는 하나의 이유로만 변경되어야 한다.

==

하나의 클래스는 하나의 책임만을 가져야 한다.

SRP 예시: Fat view

content의 300자만 잘라서 보내주세요

```python
class PostListView(ListAPIView):
    def list(self, request, *args, **kwargs):
        serializer = PostSerializer(instance=Post.objects.all(), many=True)

        json = serializer.data

json = {
    serialized_post['id']: {
        **serialized_post,
        'content': serialized_post['content'][:300]
    } for serialized_post in json
}

return Response(json)
```

SRP 예시: Fat view

```python
class PostSerializer(serializers.ModelSerializer):
    content = serializers.CharField()

    def to_representation(self, instance):
        ret = super().to_representation(instance)
        ret['content'] = ret['content'][:300]
        return ret

    class Meta:
        model = Post
        fields = ('id', 'content')


class PostListView(ListAPIView):
    serializer_class = PostSerializer
    queryset = Post.objects.all()
```

OCP: 개방 폐쇄 원칙

Open / closed principle

클래스는 확장에 대해 열려 있어야 하고,
수정에 대해서는 닫혀 있어야 한다.
~~이게 무슨소리야~~

OCP 예시: 다양한 알림

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE)


class NotificationSerializer(serializers.ModelSerializer):
    content = serializers.SerializerMethodField()

    def get_content(self, instance: Notification):
        return f'{instance.comment.created_by}님이 {instance.comment.post.title}에 댓글을 작성했습니다.'
    class Meta:
        model = Post
        fields = ('id', 'content')
```

확장:

글 작성자의 follower들에게 알림을 쏴주세요

OCP 예시: 다양한 알림(Bad)

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE)


class NotificationSerializer(serializers.ModelSerializer):
    content = serializers.SerializerMethodField()

    def get_content(self, instance: Notification):
        return f'{instance.comment.created_by}님이 {instance.comment.post.title}에 댓글을 작성했습니다.'

    class Meta:
        model = Post
        fields = ('id', 'content')
```

확장: 글 작성 알림도 쏴주세요

OCP 예시: 다양한 알림(Bad)

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE, null=True, blank=True)
    post = models.ForeignKey(Post, on_delete=models.CASCADE, null=True, blank=True)


class NotificationSerializer(serializers.ModelSerializer):
    content = serializers.SerializerMethodField()

    def get_content(self, instance: Notification):
        if instance.comment:
            return f'{instance.comment.created_by}님이 {instance.comment.post.title}에 댓글을 작성했습니다.'
        if instance.post:
            return f'{instance.post.created_by}님이 새로운 글: {instance.post.title}을 작성했습니다.'

    class Meta:
        model = Post
        fields = ('id', 'content')
```

확장: 글 작성 알림도 쏴주세요

OCP 예시: 다양한 알림(Better)

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)

    @property
    def content(self):
        raise NotImplementedError()


class CommentCreatedNotification(Notification):
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE)

    @property
    def content(self):
        ...
```

```python
class PostCreatedNotification(Notification):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)

    @property
    def content(self):
        ...


class NotificationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ('id', 'content')
```

확장:

만약 Notification이 추가된다면?

수정:

작성되었던 Serializer 코드를 수정할 필요가 있을까?

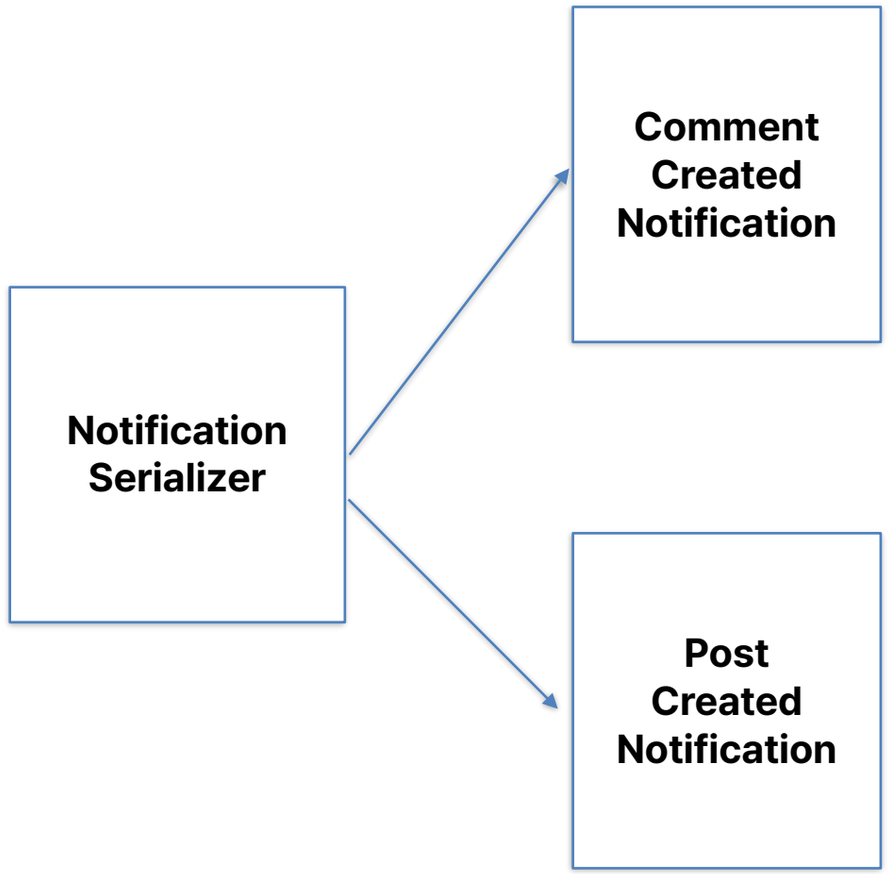
DIP: 의존관계 역전 원칙

Dependency inversion principle

상위 수준의 모듈은 하위 수준의 모듈에 의존해서는 안 된다. 둘 모두 추상화에 의존해야 한다.
추상화는 구체적인 사항에 의존해서는 안 된다.

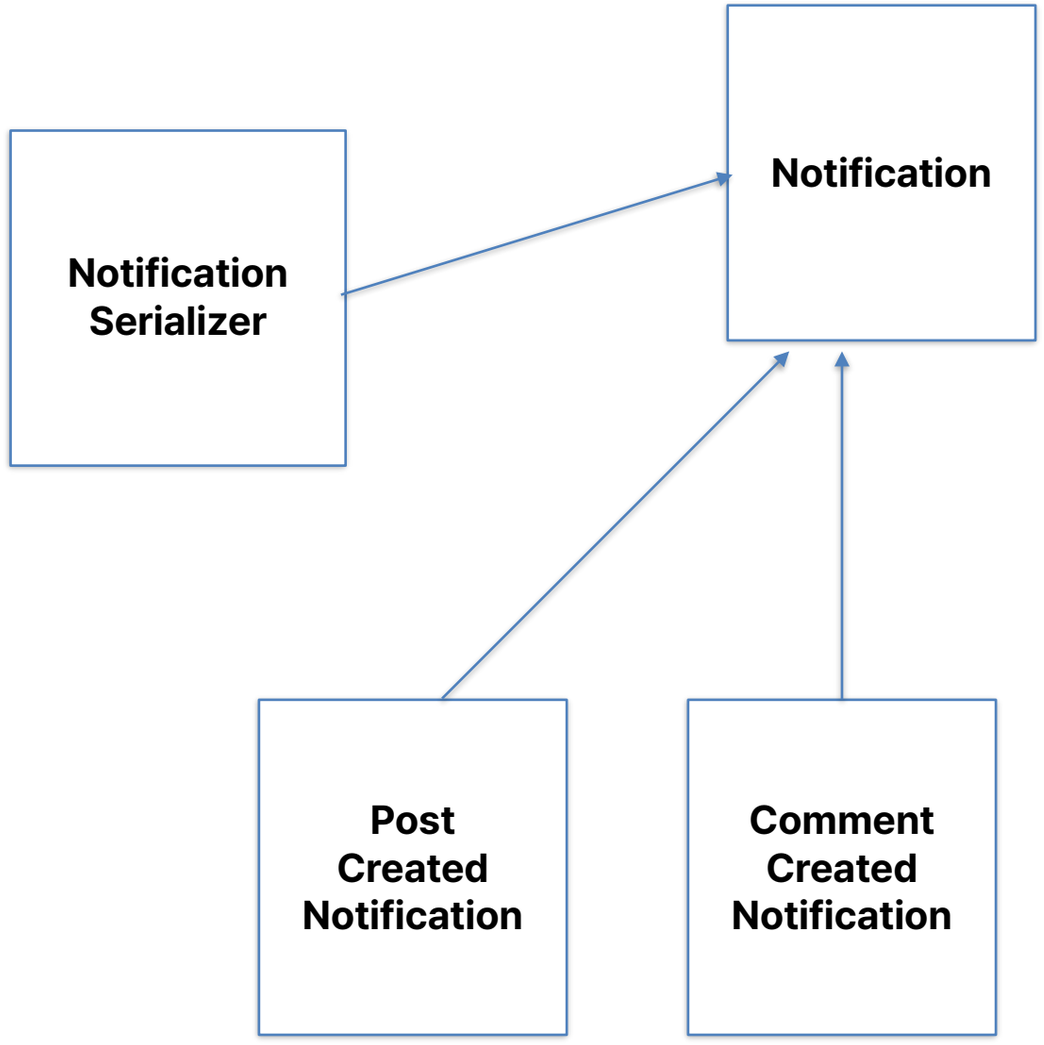
DIP 예시: 다양한 알림(Bad)

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)


class CommentCreatedNotification(Notification):
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE)


class PostCreatedNotification(Notification):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)


class NotificationSerializer(serializers.ModelSerializer):
    content = serializers.SerializerMethodField()

    def get_content(self, instance: Notification):
        if isinstance(instance, CommentCreatedNotification):
            return f'{instance.comment.created_by}님이
{instance.comment.post.title}에 댓글을 작성했습니다.'
        if isinstance(instance, PostCreatedNotification):
            return f'{instance.post.created_by}님이 새로운 글:
{instance.post.title}을 작성했습니다.'

    class Meta:
        model = Post
        fields = ('id', 'content')
```



DIP 예시: 다양한 알림(Better)

```python
class Notification(models.Model):
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)

    @property
    def content(self):
        raise NotImplementedError()


class CommentCreatedNotification(Notification):
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE)

    @property
    def content(self):
        ...


class PostCreatedNotification(Notification):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)

    @property
    def content(self):
        ...


class NotificationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ('id', 'content')
```



ISP: 인터페이스 분리 원칙

Interface segregation principle

클라이언트가 자신이 사용하지 않는 메소드에 의존하도록 강제되어서는 안된다.

ISP 예시: ViewSet

```
class ModelViewSet(mixins.CreateModelMixin,
                   mixins.RetrieveModelMixin,
                   mixins.UpdateModelMixin,
                   mixins.DestroyModelMixin,
                   mixins.ListModelMixin,
                   GenericViewSet):
    """
    A viewset that provides default `create()`, `retrieve()`, `update()`,
    `partial_update()`, `destroy()` and `list()` actions.
    """
    pass
```

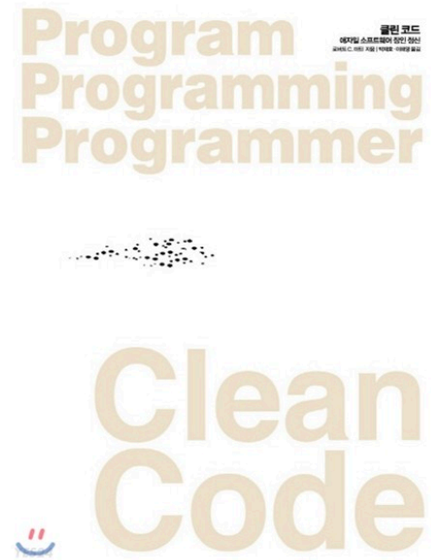
굳이...?

읽을거리

클린 코드

변수명, 함수명 등 좀 더 미시적인

단위에서의 클린 코드를 다룸



[로버트 C 마틴 blog]

[인물 소개: 로버트 C 마틴, 마틴 파울러]

[Martin fowler blog]

Multi-model logic:
Django가 잘 해주지 못하는 일

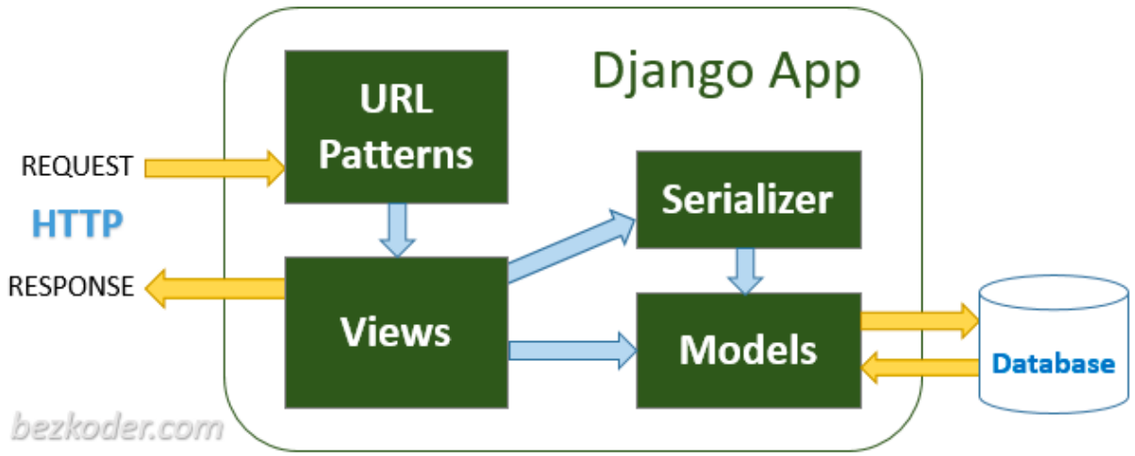
Singleton service pattern

Django가 제일 잘 작동하는 경우

View layer의 data representation이 model layer와 흡사할 때

Django ORM은 의도된 경우에 매우 잘 작동함:

- Excellent performance
- Easy to use
- Great Experience



What if: 교과서에 나오지 않는

두 개의 모델을 합쳐서 보여주고 싶은 경우

ModelSerializer?
Pagination?
ModelViewSet

```
[
  {
    "type": "post",
    "instance": {
      "title": "와플스튜디오 사랑해요",
      "content": "하지만 없죠"
    }
  },
  {
    "type": "friends-recommendation",
    "instance": {
      "friends": [
        "김찬욱",
        "김지호",
        "정대용",
        "이민규",
        "박준영"
      ]
    }
  },
  {
    "type": "ad",
    "instance": {
      "title": “le sserafim",
      "content": "antifragile",
      "link": "https://www.youtube.com/watch?v=p_XdZdg9oGc"
    }
  }
]
```


WA LE STUDIO

Fat View보단 나은 Fat Model

```python
class TransferView(APIView):
    @transaction.atomic
    def post(self, request, *args, **kwargs):
        amount = int(request.body['amount'])
        send = Send.objects.create(amount=amount)
        account = Account.objects.get(id=request.body['account_id'])
        account.amount -= amount
        request.user.amount += amount
        account.save()
        request.user.save()
```

VS

```python
class User(AbstractUser):
    ...

    def transfer(self, send: Send, account: Account, amount: number):
        account.amount -= amount
        self.amount += amount
        account.save()
        self.save()
```

Fat View보단 나은 Fat Model

```
class User(AbstractUser):
    ...
    def transfer(self, send: Send, account:
Account, amount: number):
        account.amount -= amount
        self.amount += amount
        account.save()
        self.save()
```

여기 있는 게 맞을까?

Account, send 수정 시...

테스트하기 어려운 구조

Context

Service pattern

transfer/services.py

```
class TransferService:
    def send(self, user: User, account: Account, amount:
int):
        account.amount -= amount
        user.amount += amount
        account.save()
        user.save()
        Send.objects.create(amount=amount)


transfer_service = TransferService()
```

transfer/views.py

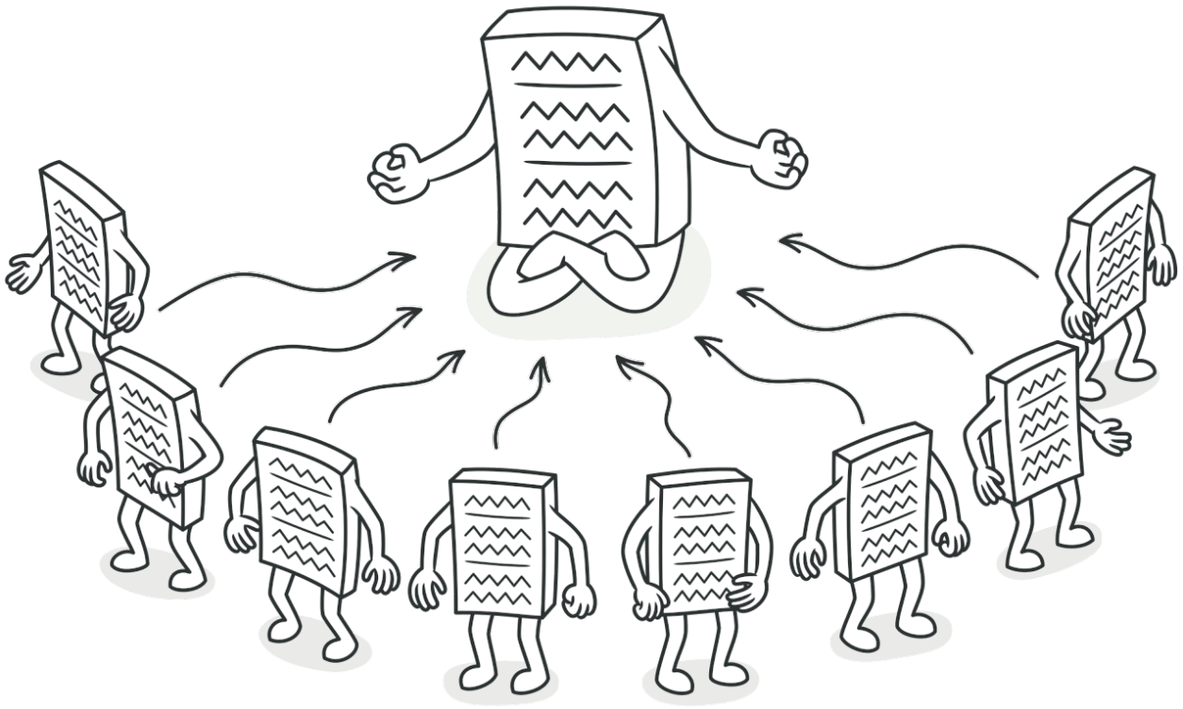
```
class TransferView(APIView):
    def post(self, request, *args, **kwargs):
        account = get_object_or_404(Account,
id=request.body['account_id'])
        transfer_service.transfer(request.user,
account, amount=int(request.body['amount']))
```

Service pattern: test

```python
class TransferTestCase(TestCase):
    def setUp(self):
        self.transfer_service = TransferService()
        self.chanuk = User.objects.create()

    def test_transfer_good(self):
        account = Account.objects.create(user=self.chanuk)
        self.transfer_service.transfer(user=user, account=account, amount=1000)
        self.assertEqual(Send.objects.count(), 1)
        self.assertEqual(account.amount, 1000)
```

테스트를 작성하지 않더라도,
항상 테스트하기 쉬운 코드를
작성하려 노력하세요

Singleton service pattern



```
transfer/services.py

class TransferService:
    def send(self, user: User, account: Account, amount:
int):
        …


transfer_service = TransferService()
```

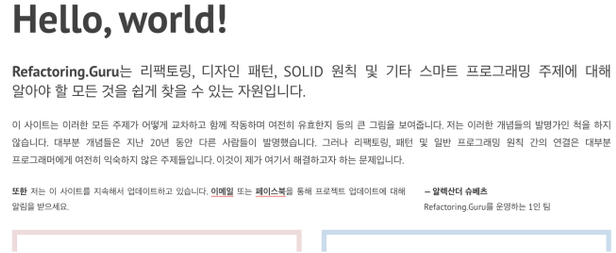
상태와 맥락이 없음

Class가 App에 딱 하나의 instance만 존재

WA LE STUDIO

읽을거리

Refactoring.guru -> 추천!



Django best practices: Refactoring fat models

Clean architecture in django (djangocon 2021)

WA LE
STUDIO

Assignment 3

Questions?