Ask Company

리액트와 함께 장고 시작하기 / 장고 Views

클래스 기반 뷰 시작하기

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

View

호출가능한 객체 (Callable Object)

함수 기반 뷰 (Function Based View)

View 구현의 기본 !!! → FBV로 구현할 줄 알아야 응용이 가능합니다.

공통 기능들은 장식자 문법으로 적용

```
@api_view(['GET'])
@throttle_classes([OncePerDayUserThrottle])
def my_view(request):
    return Response({"message": "Hello for today!"})
```

클래스 기반 뷰 (Class Based View)

공통 기능들은 상속 문법으로 적용

```
class MyView(APIView):
    throttle_classes = [OncePerDayUserThrottle]

    def get(self, request):
        return Response({"message": "Hello for today!"})
```

Class Based View

View 함수를 만들어주는 클래스

as_view() 클래스 함수를 통해, View 함수를 생성

상속을 통해, 여러 기능들을 믹스인.

장고 기본 CBV 팩키지

django.views.generic

https://github.com/django/django/tree/3.0.2/django/views/generic

써드파티 CBV

django-braces

https://django-braces.readthedocs.io

https://docs.djangoproject.com/en/3.0/topics/class-based-views/

CBV 컨셉 구현해보기

#1. FBV

```python
from django.shortcuts import get_object_or_404, render

def post_detail(request, id):
    post = get_object_or_404(Post, id=id)
    return render(request, 'blog/post_detail.html', {
        'post': post,
    })

def article_detail(request, id):
    article = get_object_or_404(Article, id=id)
    return render(request, 'blog/article_detail.html', {
        'article': article,
    })
```

```python
urlpatterns = [
    path('post/<int:id>/', post_detail),
    path('article/<int:id>/', article_detail),
]
```

#2. 함수를 통해, 동일한 View 함수 생성

```
def generate_view_fn(model):
    def view_fn(request, id):
        instance = get_object_or_404(model, id=id)
        instance_name = model._meta.model_name
        template_name = '{}/{}_detail.html'.format(model._meta.app_label, instance_name)
        return render(request, template_name, {
            instance_name: instance,
        })
    return view_fn


post_detail = generate_view_fn(Post)

article_detail = generate_view_fn(Article)
```

#3. Class로 동일한 View 함수 구현

```python
class DetailView:
    def __init__(self, model):
        self.model = model

    def get_object(self, *args, **kwargs):
        return get_object_or_404(self.model, id=kwargs['id'])

    def get_template_name(self):
        return '{}/{}_detail.html'.format(
            self.model._meta.app_label,
            self.model._meta.model_name)

    def dispatch(self, request, *args, **kwargs):
        object = self.get_object(*args, **kwargs)
        return render(request, self.get_template_name(), {
            self.model._meta.model_name: object,
        })

    @classmethod
    def as_view(cls, model):
        def view(request, *args, **kwargs):
            self = cls(model)
            return self.dispatch(request, *args, **kwargs)
        return view
```

```python
post_detail = DetailView.as_view(Post)

article_detail = DetailView.as_view(Article)
```

#4. 장고 기본 제공 CBV 활용

```
from django.views.generic import DetailView

post_detail = DetailView.as_view(model=Post, pk_url_kwarg='id')
article_detail = DetailView.as_view(model=Article, pk_url_kwarg='id')
```

pk_url_kwarg 인자를 "pk"로 지정했다면~

```
post_detail = DetailView.as_view(model=Post)
article_detail = DetailView.as_view(model=Article)
```

```
urlpatterns = [
    path('post/<int:pk>/', post_detail),
    path('article/<int:pk>/', article_detail),
]
```

상속을 통한 CBV 속성 정의

```
from django.views.generic import DetailView

class PostDetailView(DetailView):
    model = Post
    pk_url_kwarg = 'id'

post_detail = PostDetailView.as_view()
```

CBV는 ~

CBV가 정한 관례대로 개발할 경우, 아주 적은 양의 코드로 구현

그 관례에 대한 이해가 필요 → FBV를 통한 개발경험이 큰 도움.

필요한 설정값을 제공하거나, 특정 함수를 재정의하는 방식으로 커스텀 가능

하지만, 그 관례를 잘 이해하지 못하고 사용하거나, 그 관례에서 벗어난 구현을 하고자 할 때에는 복잡해지는 경향이 있습니다.

CBV를 제대로 이해할려면 ~

코드를 통한 이해가 지름길

파이썬 클래스에 대한 이해가 필요 (특히 상속, 인자 packing/unpacking)

https://github.com/django/django/tree/2.1/django/views/generic

CBV 코드를 동일하게 동작하는 FBV로 구현해보는 연습을 추천

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

- Ask Company