Django view structure

> create & update view의 경우, 작성은 GET 부터 하는데,
> 왜 구조 상 조건 분기에서는 `request.method == 'POST'` 를 먼저 작성할까?

GET 먼저 분기 했을 때

CASE 1

```
if request.method == 'GET':
    form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/index.html', context)
else:
    form = ArticleForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('articles:index')
    else:
        context = {
            'form': form
        }
        return render(request, 'articles/index.html', context)
```

- 불필요한 반복

CASE 2

> 그럼 CASE 1 코드가 반복이라면 이렇게 해보면 어떨까 ?

```
if request.method == 'GET':
    form = ArticleForm()
else:
    # 하지만 이렇게 하면 여기서 POST가 아니라 PUT, PATCH, DELETE 등
    # 다른 메서드로 요청이 와도 여기가 실행되는 문제가 생김 !
    form = ArticleForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('articles:index')
context = {
    'form': form,
}
return render(request, 'articles/index.html', context)
```

- 잠깐 그건 우리가 원래 작성하던 방식대로해도 같은 이야기 아닐까?

```
if request.method == 'POST':
    form = ArticleForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('articles:index')
else:
    # 여기도 GET이 아닌 다른 메서드로 요청이 오면 동작 된다.
    form = ArticleForm()
context = {
    'form': form,
}
return render(request, 'articles/index.html', context)
```

- 둘 다 맞는 이야기 !
- 다만 if 문을 지나 else 문이 실행 될 때의 2가지 상황을 잘 생각해봅시다.
    - (`else`일 경우에 집중하세요.)

- 첫번째 코드를 먼저 살펴 봅시다.

```
if request.method == 'GET':
    form = ArticleForm()
else:
    form = ArticleForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('articles:index')
```

- 여기서 else 구문은 http method가 GET이 아닐 때(POST, PUT, DELETE 등 다른 메서드) 수행됩니다.

- 두번째 코드(우리가 배운)도 살펴 봅시다.

```
if request.method == 'POST':
    form = ArticleForm(request.POST)
    if form.is_valid():
        form.save()
        return redirect('articles:index')
else:
    form = ArticleForm()
```

- 여기서 else 구문은 http method가 POST이 아닐 때(GET, PUT, DELETE 등 다른 메서드) 수행됩니다.

- 여기서 중요한 점은 else 구문이 수행될 때의 코드의 차이입니다.
- 첫번째 코드에서의 else는 `save()`가 있는 즉, DB에 무언가 조작을 하는 코드입니다.
- 그런데 두번째 코드에서의 else는 단순히 form 인스턴스를 생성하는 코드입니다.
- 의미론적으로 생각해봅시다.

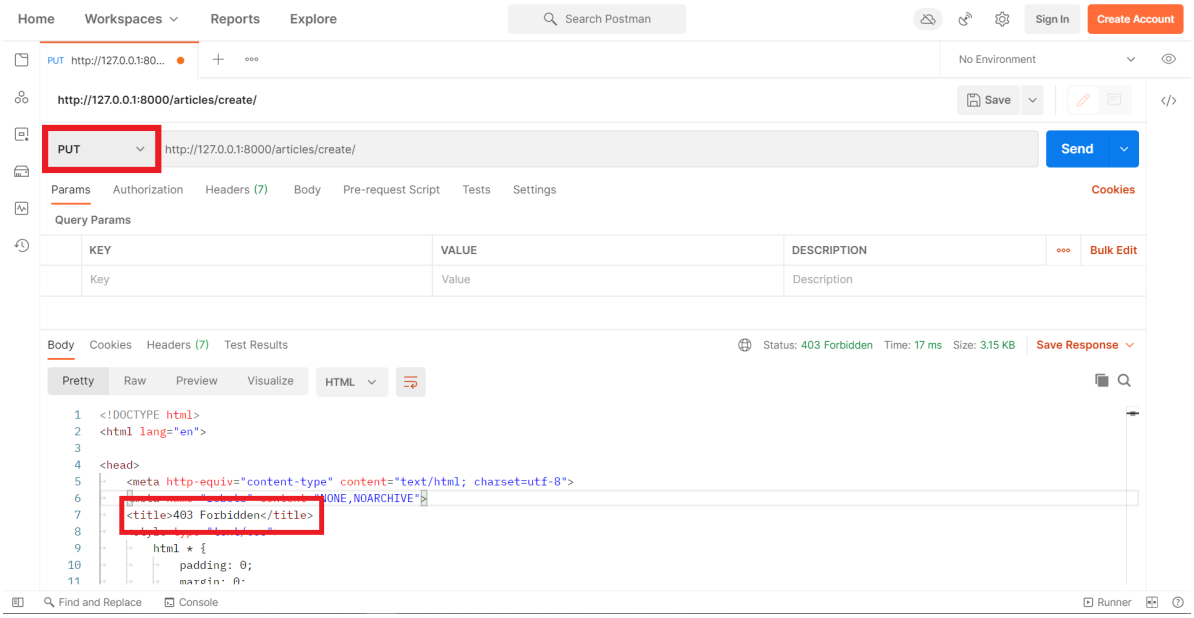
> "만약 사용자가 내가 고려한 조건과 다른 method로 요청했을 때
> 단순히 form 인스턴스를 생성하는 코드를 수행하게 하실건가요,
> 아니면 DB를 조작하는 코드를 수행하게 하실 건가요?"

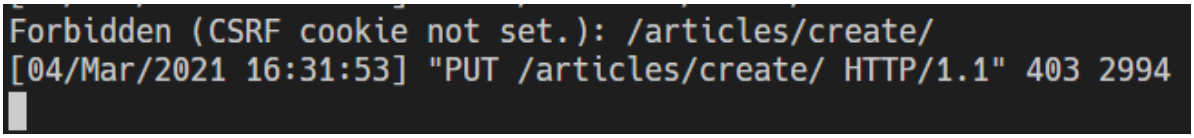
실제 응답 확인

- 예시 코드

```
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('articles:index')
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

- POSTMAN 이라는 프로그램을 통해 CREATE URL에 PUT 메서드로 요청을 보내보겠습니다.





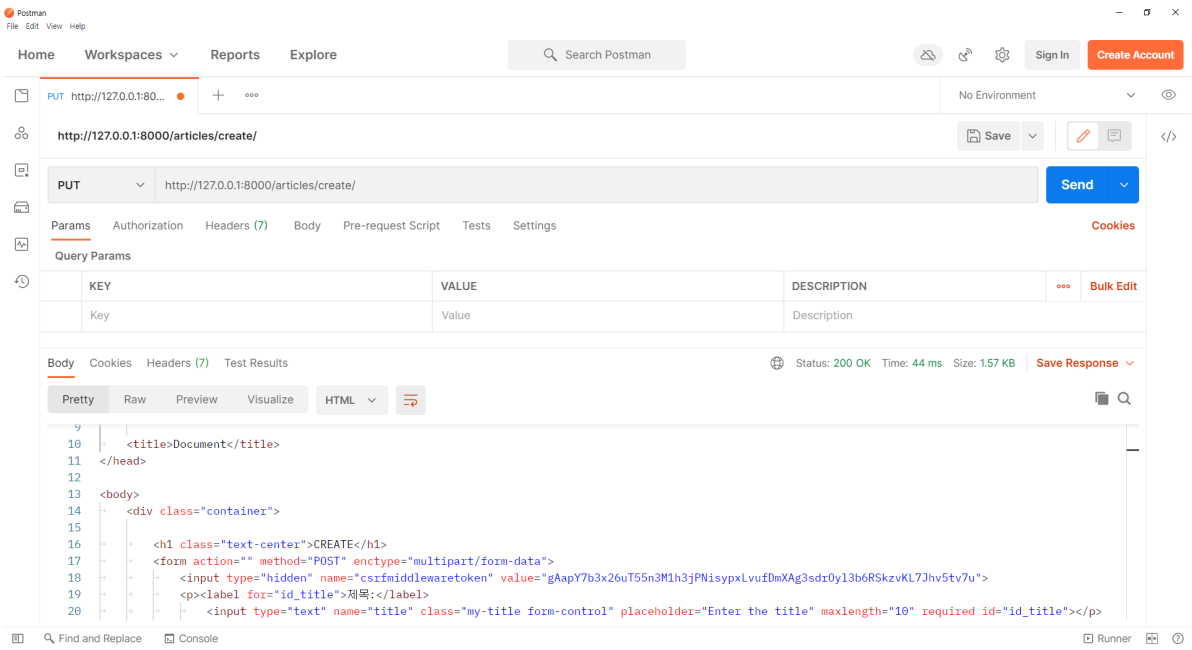
- 403 Forbidden이라는 title과 함께 body 태그내용을 보면
  "CSRF 검증에 실패했습니다. 요청을 중간하였습니다." 라는 문구를 확인할 수 있습니다.
- django는 GET method가 아니면 언제나 요청에서 csrf 검증을 시도합니다.
- 이 검증에 관한 코드는 settings.py에 작성되어 있습니다.

```
# settings.py

MIDDLEWARE = [
    ...,
    'django.middleware.csrf.CsrfViewMiddleware',
    ...,
]
```

  - PUT 메서드 요청은 코드 상으로 else 구문이 실행되기는 하겠지만,
    애초에 csrf 검증을 통과하지 못하기 때문에 create view 함수는 동작조차 하지 못합니다.

- 그렇다면 미들웨어 해당 코드를 주석처리하고 다시 요청을 보내면 어떻게 될까요?



- csrf 검증이 동작하지 않게 되고 코드상으로 else 구문이 실행되기 때문에 우리가 GET으로 요청을 보낸 것처럼 게시글 작성 페이지가 응답으로 오게됩니다.

허용되지 않은 HTTP method 처리하기

- 그런데 우연히 csrf_token을 맞춰서 통과를 하게 된다면 어떻게 될까요?
- 마찬가지로 게시글 작성 페이지가 렌더링 될 것입니다.
- 그래서 이것조차 방지한다고 한다면 django가 제공하는 View decorators를 사용해 완성할 수 있습니다.

```
from django.views.decorators.http import require_http_methods


@require_http_methods(['GET', 'POST'])
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('articles:index')
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

- 이렇게 하면 혹여나 csrf_token을 맞춰서 뚫었다고해도 create 함수는 실행되지 않고 405 Method Not Allowed 를 응답합니다.

> **django가 현재 구조를 선택한 것과 그럴 수 밖에 없는 이유**
>
> 1. POST가 아니면(GET, PUT, DELETE 등) DB 조작과 관련되지 않은 코드가 실행되도록 하기 위함

> 2. 검증을 통과하더라도 완벽하고 올바르게 처리할 수 있도록 데코레이터까지 사용해 적절한 응답 상태 코드까지 전달하면 끝 !