# 3장. MTV 패턴

# 목 차

# url과 view

- **url 매핑 – poll/urls.py**

```
from django.urls import path
from poll import views

app_name = 'poll'  #네임 스페이스 – url별칭(소속)
urlpatterns = [
    # /polls/
    path('', views.index, name='index'),
    # /polls/2/
    path('<int:question_id>/', views.detail, name='detail'),
    # /polls/2/results
    path('<int:question_id>/results/', views.results, name='results'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

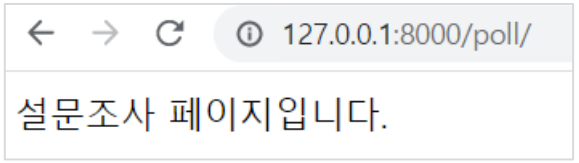
# url과 view

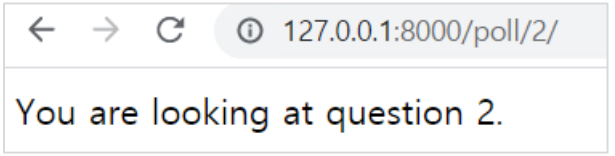
- 요청 제어 및 처리 – poll/views.py

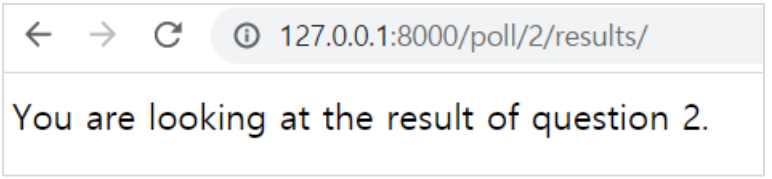
```python
from django.http import HttpResponse

def index(request):
    return HttpResponse('설문조사 페이지입니다.')

def detail(request, question_id):
    return HttpResponse("You are looking at question %s." % question_id)

def results(request, question_id):
    response = "You are looking at the result of question %s."
    return HttpResponse(response % question_id)

def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

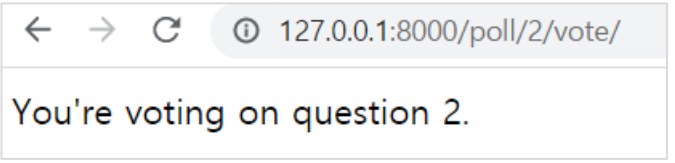
# url과 view

- 브라우저 화면 내용



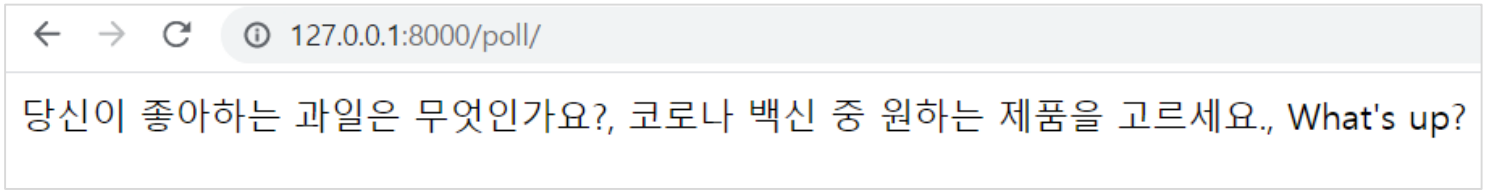



# url과 view

- **질문 목록 보기 – poll/views.py**

```
def index(request):
    question_list = Question.objects.order_by('-pub_date')[:5]
    #리스트 내포 구문 - 반복문
    list = ', '.join([q.question_text for q in question_list])
    return HttpResponse(list)
```



# 템플릿으로 질문 목록 페이지 만들기

## ◆ 템플릿(template)

1. templates 디렉터리를 루트 디렉터리 바로 밑에 만든다.

```
(mysite) C:\projects\polls>mkdir  templates
```

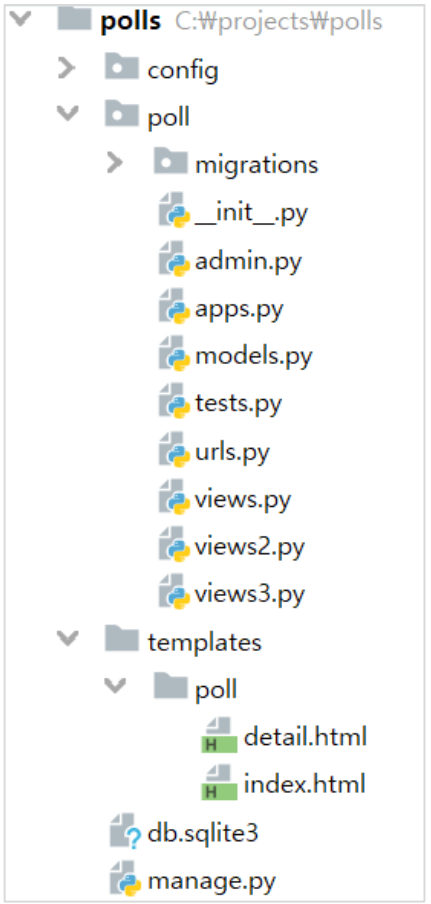
2. 템플릿 디렉터리 위치를 config/settings.py에 등록하기

```
TEMPLATES = [
    ……
    'DIRS': [BASE_DIR / 'templates'],
    ……
]
```

# 템플릿으로 질문 목록 페이지 만들기

4. 템플릿 파일 만들기

– 경로 : C:/projects/polls/templates/poll/index.html




```
{% if question_list %}
    <ul>
    {% for question in question_list %}
        <li>
            <a href="/poll/{{question.id}}">
                {{ question.question_text }}
            </a>
        </li>
    {% endfor %}
    </ul>
{% else %}
    <p>설문이 없습니다.</p>
{% endif %}
```

detail.html

```
<h3>{{ question}}</h3>
```

# 템플릿으로 질문 목록 페이지 만들기

5. template.render() 사용 – poll/views.py

```
from .models import Question
from django.template import loader

def index(request):
    question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('poll/index.html')
    context = {'question_list': question_list}
    return HttpResponse(template.render(context, request))
```

# 템플릿으로 질문 목록 페이지 만들기

◆ 템플릿(template) 태그

| 템플릿 태그 | 설 명 |
|---|---|
| **{% if question_list %}** | question_list가 있다면(조건문) |
| **{% for question in question_list %}** | question_list를 반복하면 순차적으로 question에 대입(반복문) |
| **{{ question.id }}** | question 객체의 id 출력(출력문) |
| **{{ question.subject }}** | question 객체의 subject 출력 |

# 템플릿으로 질문 목록 페이지 만들기

6. shortcouts render() : 축약 사용 – poll/views.py

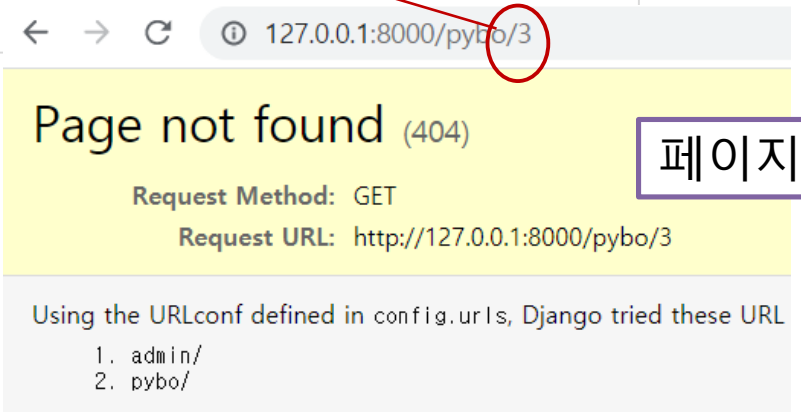
```
from django.shortcuts import render
from .models import Question

def index(request):
    question_list = Question.objects.order_by('-pub_date')
    context = {'question_list': question_list}
    return render(request, 'poll/index.html', context)
```

# 질문 상세 페이지 만들기

## ◆ 질문 상세 페이지

1. detail 페이지 URL 매핑하기 – pybo/urls.py

```
from django.urls import path
from pybo import views
urlpatterns = [
    path('', views.index),
    path('<int:question_id>/', views.detail),
]
```




페이지 만들기 전 링크 오류

# 질문 상세 페이지 만들기

## ◆ 설문 상세 페이지

2. detail 함수 구현 – poll/views.py

```
def detail(request, question_id):
    #question = Question.objects.get(id=question_id)
    #오류 화면처리
    question = get_object_or_404(Question, pk=question_id)
    context = {'question': question}
    return render(request, 'poll/detail.html', context)
```

# 질문 상세 페이지 만들기

## ◆ 설문 상세 페이지

3. poll/detail.html 작성

```
<h3>{{ question.question_text}}</h3>

<ul>
    {% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }}</li>
    {% endfor %}
</ul>
```

**코로나 백신 중 원하는 제품을 고르세요.**

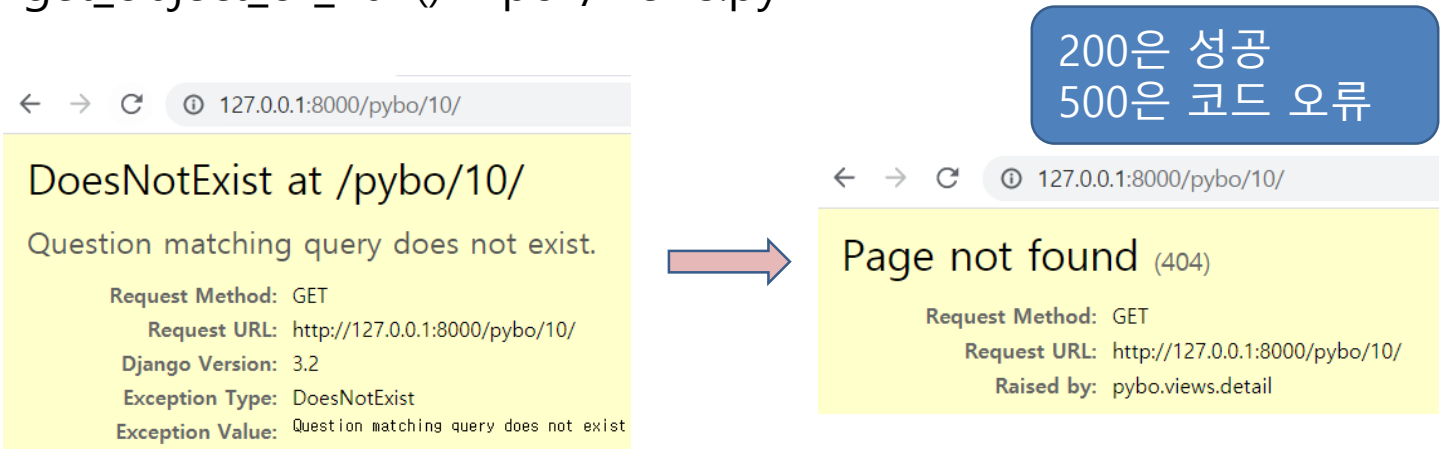
- 1. 얀센 2. 화이자 3. 모더나

# 오류 화면 처리

## ◆ 오류 화면 처리

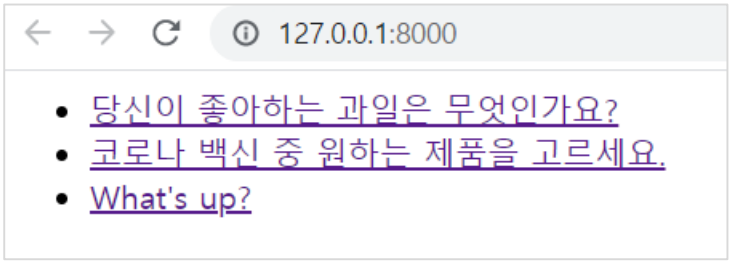
- 존재하지 않는 페이지에 접속하면 오류 대신 404 페이지를 출력함

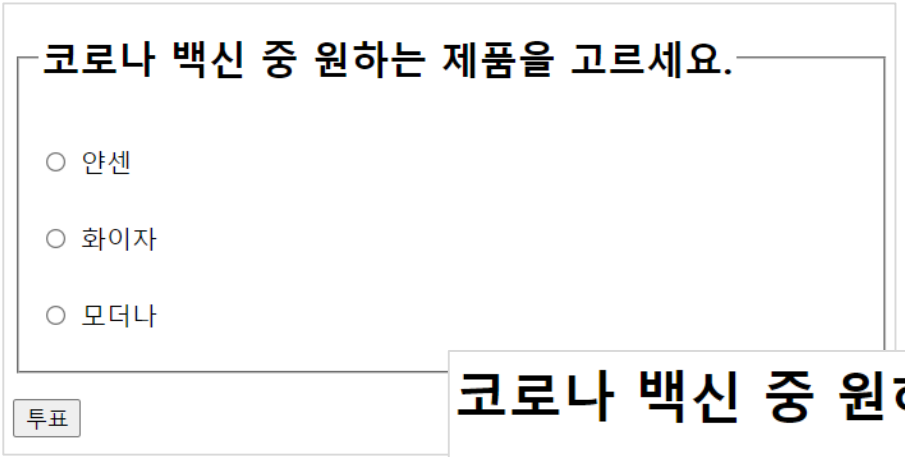
get_object_or_404() – poll/views.py




```python
def detail(request, question_id):
    #question = Question.objects.get(id=question_id)
    #오류 화면처리
    question = get_object_or_404(Question, pk=question_id)
    context = {'question': question}
    return render(request, 'poll/detail.html', context)
```

# 설문 투표하기


127.0.0.1:8000
당신이 좋아하는 과일은 무엇인가요?
코로나 백신 중 원하는 제품을 고르세요.
What's up?


코로나 백신 중 원하는 제품을 고르세요.
얀센
화이자
모더나
투표


코로나 백신 중 원하는 제품을 고르세요.
얀센--2 votes
화이자--3 votes
모더나--4 votes
다시 투표 하시겠습니까?

# URL 네임 스페이스

## ◆ URL 별칭 사용하기 〉 네임 스페이스

1. 템플릿에서 URL 네임 스페이스 사용하기 – poll/index.html

```
{% if question_list %}
    <ul>
    {% for question in question_list %}
        <li>
            <a href="{% url 'poll:detail' question.id %}">
                {{ question.question_text }}
            </a>
        </li>
    {% endfor %}
    </ul>
{% else %}
    <p>설문이 없습니다.</p>
{% endif %}
```

# 설문 투표하기

1. 투표 양식 만들기 – poll/detail.html

```
<form action="{% url 'poll:vote' question.id %}" method="post">
    {% csrf_token %}
    <fieldset >
        <legend><h2>{{ question.question_text}}</h2></legend>
        {% if error_message %}
            <p><strong>{{ error_message }}</strong></p>
        {% endif %}
        {% for choice in question.choice_set.all %}
            <input type="radio" name="choice" id="choice{{ forloop.counter }}"
                   value="{{ choice.id }}">
            <label for="choice{{ forloop.counter }}" style="line-height: 3rem">
                {{ choice.choice_text }}
            </label><br>
        {% endfor %}
    </fieldset>
    <p><input type="submit" value="투표"></p>
</form>
```

**forloop.counter – for문의 순서로 1부터 표시**

# 설문 투표하기

2. URL 매핑 추가하기 – poll/urls.py

```
urlpatterns = [
    ...
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

# 설문 투표하기

2. vote 함수 정의하기 – poll/views.py

```
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        sel_choice = question.choice_set.get(pk=request.POST['choice'])
    except KeyError:
        return render(request, 'poll/detail.html', {
                        'question': question,
                        'error_message': '선택을 하세요!'
        })
    else:
        sel_choice.votes += 1
        sel_choice.save()
        return HttpResponseRedirect(reverse('poll:results', args=(question.id,)))
```

# 설문 투표하기

2. vote 함수 정의하기 – poll/views.py

request.POST 는 키로 전송된 자료에 접근할 수 있도록 해주는 사전과 같은 객체입니다.
이 경우, request.POST['choice'] 는 선택된 설문의 ID를 문자열로 반환합니다.
request.POST 의 값은 항상 문자열들입니다.

Django는 같은 방법으로 GET 자료에 접근하기 위해 request.GET 를 제공합니다.
그러나 POST 요청을 통해서만 자료가 수정되게하기 위해서, 명시적으로 코드에 request.POST 를 사용하고 있습니다.

만약 POST 자료에 choice 가 없으면, request.POST['choice'] 는 KeyError 가 일어납니다.
위의 코드는 KeyError 를 체크하고, choice가 주어지지 않은 경우에는 에러 메시지와 함께 설문조사 폼을 다시보여줍니다.

설문지의 수가 증가한 이후에, 코드는 일반 HttpResponse 가 아닌 HttpResponseRedirect 를 반환하고,
HttpResponseRedirect 는 하나의 인수를 받습니다: 그 인수는 사용자가 재전송될 URL 입니다.

# 설문 투표하기

2. vote 함수 정의하기 – poll/views.py

HttpResponseRedirect 생성자 안에서 reverse() 함수를 사용하고 있습니다.
이 함수는 뷰 함수에서 URL을 하드코딩하지 않도록 도와줍니다.
제어를 전달하기 원하는 뷰의 이름을, URL패턴의 변수부분을 조합해서 해당 뷰를 가리킵니다.
이 reverse() 호출은 아래와 같은 문자열을 반환할 것입니다.

'/polls/3/results/'
여기서 3 은 question.id 값입니다. 이렇게 리디렉션된 URL은 최종 페이지를 표시하기 위해 'results' 뷰를 호출합니다.

request 는 HttpRequest 개체입니다.
어떤 이가 설문조사에 설문을 하고난 뒤에는, vote() 뷰는 설문조사 결과 페이지로 리다이렉트합니다

# 설문 투표하기

2. vote 함수 정의하기 – poll/views.py

## `django.urls` utility functions

### `reverse()`

If you need to use something similar to the `url` template tag in your code, Django provides the following function:

**`reverse`**(*viewname*, *urlconf=None*, *args=None*, *kwargs=None*, *current_app=None*)

**`viewname`** can be a URL pattern name or the callable view object. For example, given the following **`url`**:

```
from news import views

path('archive/', views.archive, name='news-archive')
```

you can use any of the following to reverse the URL:

```
# using the named URL
reverse('news-archive')

# passing a callable object
# (This is discouraged because you can't reverse namespaced views this way.)
from news import views
reverse(views.archive)
```

# 설문 투표하기

1. 투표 결과 URL 매핑 추가하기 – poll/urls.py

```
urlpatterns = [
    ...
    path('<int:question_id>/results/', views.results, name='results'),
]
```

2. result 함수 정의하기 – poll/views.py

```
def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    context = {'question': question}
    return render(request, 'poll/results.html', context)
```

# 설문 투표하기

3. 투표 결과 페이지 – poll/results.html

```
<h1>{{ question.question_text }}</h1>

<ul>
    {% for choice in question.choice_set.all %}
        <li>
            {{ choice.choice_text }}--{{ choice.votes }} votes
        </li>
    {% endfor %}
</ul>

<a href="{% url 'poll:detail' question.id %}">다시 투표 하시겠습니까?</a>
```

# CSRF(cross-site request forgery)

## ◆ csrf

### 사이트 간 요청 위조

20개 언어

위키백과, 우리 모두의 백과사전.

**사이트 간 요청 위조**(또는 **크로스 사이트 요청 위조**, 영어: Cross-site request forgery, **CSRF**, **XSRF**)는 웹사이트 취약점 공격의 하나로, 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 하는 공격을 말한다.

유명 경매 사이트인 옥션에서 발생한 개인정보 유출 사건에서 사용된 공격 방식 중 하나다.

사이트 간 스크립팅(XSS)을 이용한 공격이 사용자가 특정 웹사이트를 신용하는 점을 노린 것이라면, 사이트간 요청 위조는 특정 웹사이트가 사용자의 웹 브라우저를 신용하는 상태를 노린 것이다. 일단 사용자가 웹사이트에 로그인한 상태에서 사이트간 요청 위조 공격 코드가 삽입된 페이지를 열면, 공격 대상이 되는 웹사이트는 위조된 공격 명령이 믿을 수 있는 사용자로부터 발송된 것으로 판단하게 되어 공격에 노출된다.

#### 공격 과정 [ 편집 ]

1. 이용자는 웹사이트에 로그인하여 정상적인 쿠키를 발급받는다
2. 공격자는 다음과 같은 링크를 이메일이나 게시판 등의 경로를 통해 이용자에게 전달한다.

   http://www.geocities.com/attacker
3. 공격용 HTML 페이지는 다음과 같은 이미지태그를 가진다.

```
<img src= "https://travel.service.com/travel_update?.src=Korea&.dst=Hell">
```

   해당 링크는 클릭시 정상적인 경우 출발지와 도착지를 등록하기위한 링크이다. 위의 경우 도착지를 변조하였다.
4. 이용자가 공격용 페이지를 열면, 브라우저는 이미지 파일을 받아오기 위해 공격용 URL을 연다.
5. 이용자의 승인이나 인지 없이 출발지와 도착지가 등록됨으로써 공격이 완료된다. 해당 서비스 페이지는 등록 과정에 대해 단순히 쿠키를 통한 본인확인 밖에 하지 않으므로 공격자가 정상적인 이용자의 수정이 가능하게 된다.

# csrf_token

## ◆ csrf_token

django는 csrf 공격에 대한 방어로 csrf_token을 발급 체크한다.

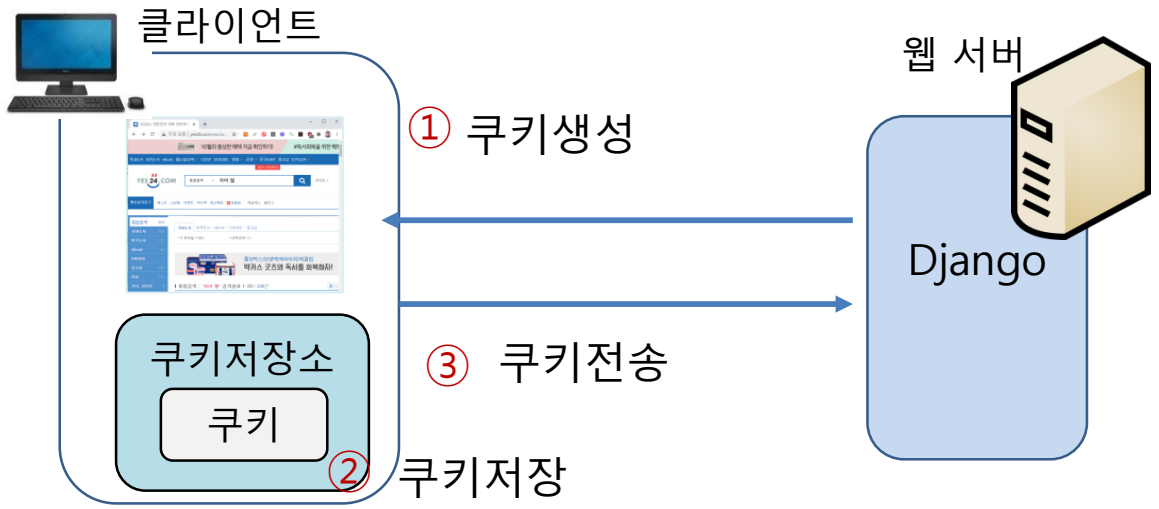
### 동작 과정

1. 사용자가 해당 페이지에 접속하면 Django에서 자동으로 csrf_token을 클라이언트로 보내어 cookie에 저장
2. 사용자가 form을 모두 입력한 후 제출버튼을 클릭한다.
3. form과 cookie의 csrf_token을 함께 POST로 전송한다.
4. 전송된 token의 유효성을 검증
5. 유효한 요청이면 요청을 처리
    1. token이 유효하지 않거나(없거나 값이 잘못된 경우) 검증 오류 시에는 403 Forbidden Response 반환

# 쿠키(cookie)

## 쿠키(cookie)

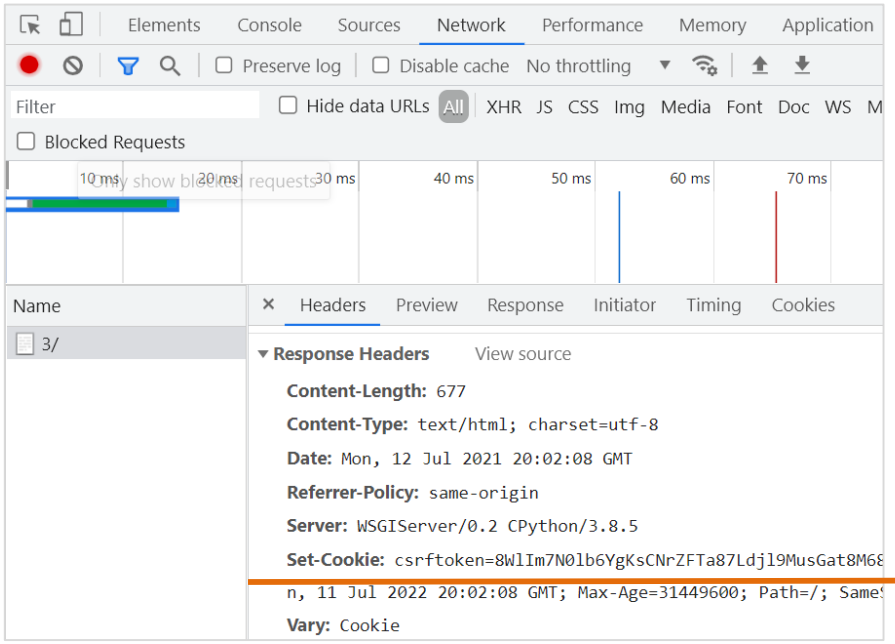
- 클라이언트와 웹 서버간의 상태를 지속적으로 유지하는 방법으로 쿠키와 세션이 있다.
- 쿠키는 세션과 달리 상태 정보를 웹 서버가 아닌 클라이언트에 저장한다.
  예) 어떤 웹 사이트를 처음 방문한 사용자가 로그인 인증을 하고 나면 아이디와 비밀번호를 기록한 쿠키가 만들어지고, 그 다음부터 사용자가 그 사이트에 접속하면 별도의 절차를 거치지 않고 쉽게 접속할 수 있다.
- 쿠키는 클라이언트의 일정 폴더에 정보를 저장하므로 웹 서버의 부하를 줄일 수 있으나 개인 정보 기록이 남기 때문에 보안에 문제가 있다.



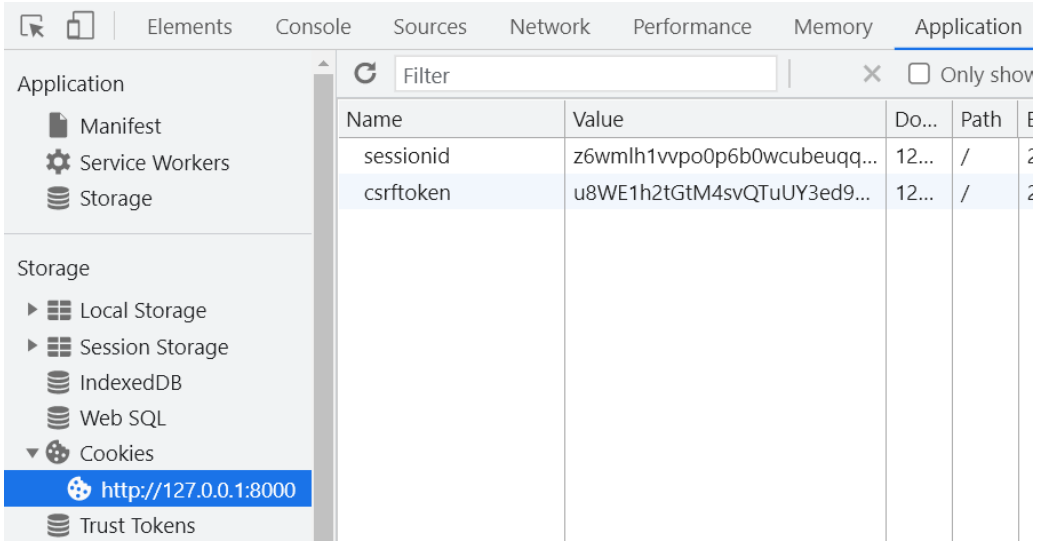
# Cookie(쿠키)

◆ 쿠키

개발자도구 > Network




개발자도구 > Application

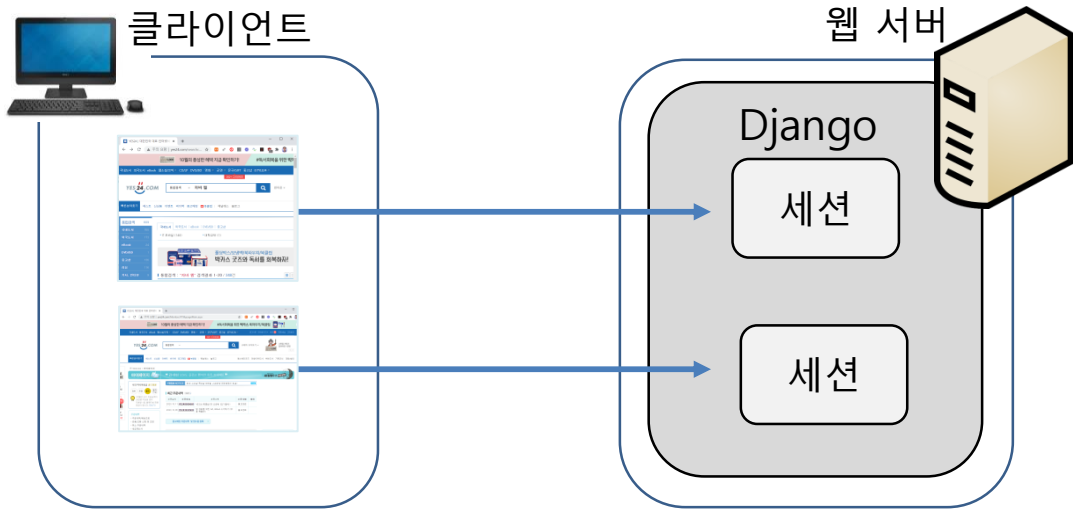


# 세션(session)

## 세션(session)

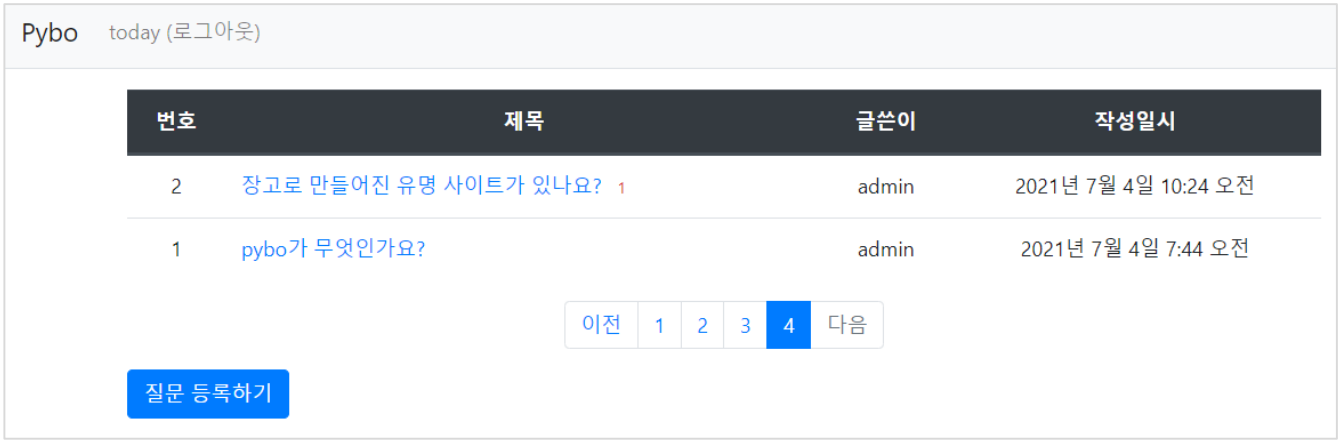
- 클라이언트와 웹 서버 간의 상태를 지속적으로 유지하는 방법을 말한다.
- 사용자 인증을 통해 특정 페이지를 사용할 수 있도록 권한 상태 유지.

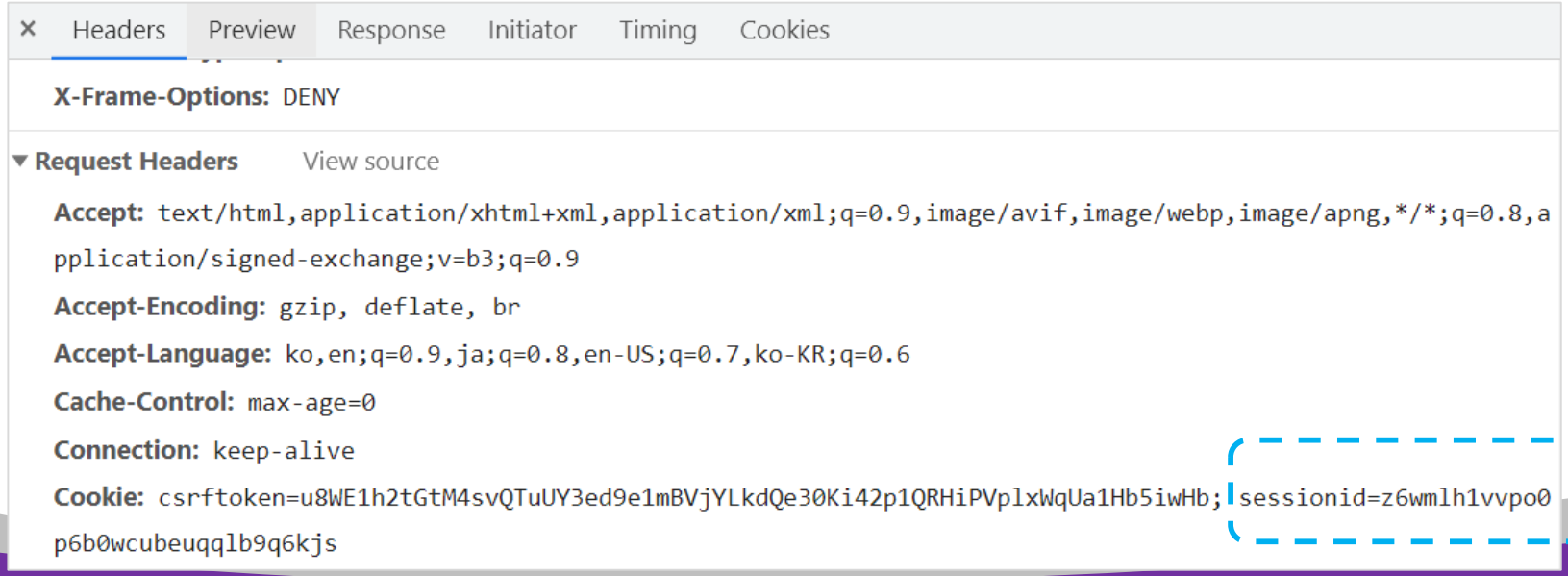
  예) 웹 쇼핑몰 – 장바구니나 주문 처리와 같은 회원 전용 페이지 로그인 후 다른 웹 페이지에 갔다가 돌아와도 로그인 상태 유지됨
- 세션은 오직 웹 서버에 존재하는 객체로 웹 브라우저마다 하나씩 존재하므로 브라우저를 닫기 전까지 웹 페이지를 이동하더라도 사용자 정보가 유지된다.



# 세션(session)

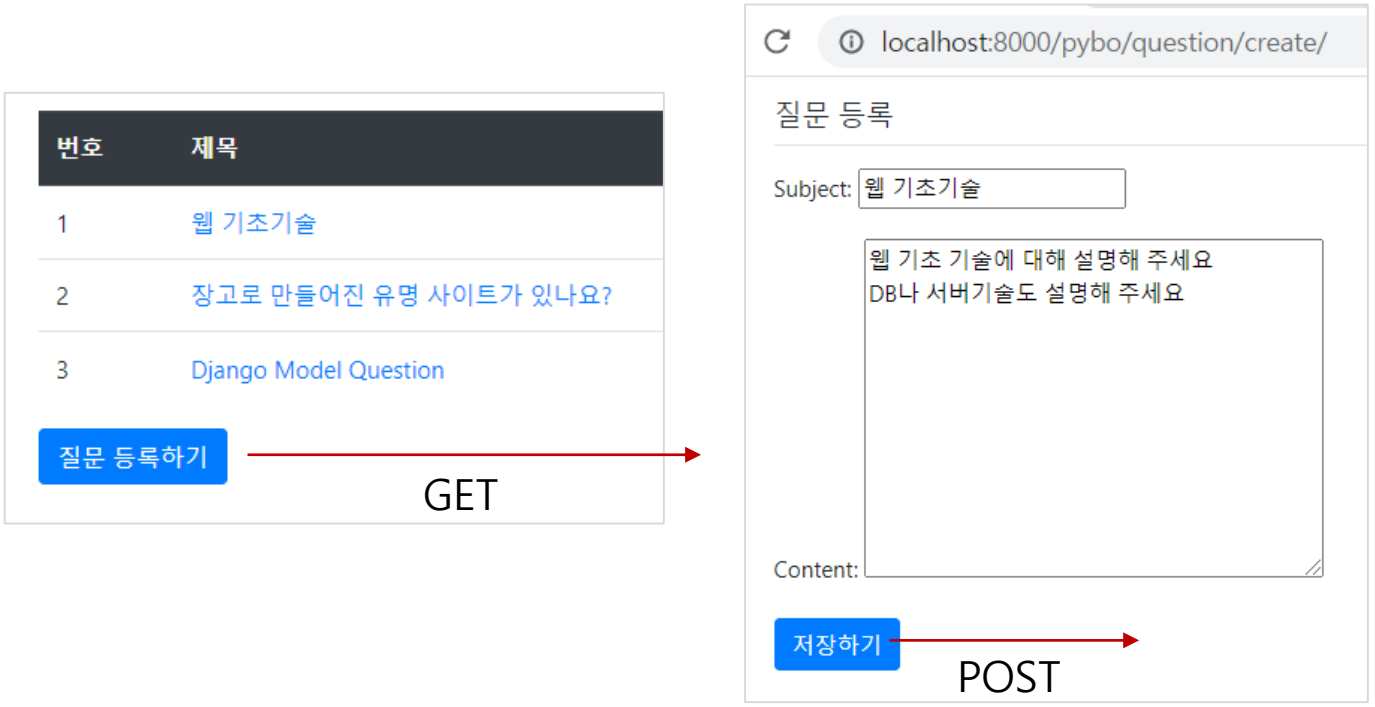
## 세션(session)



# GET 방식과 POST 방식 비교

➢ GET 방식과 POST 방식 비교




GET 방식 요청 : 질문 등록하기 -> question/create 페이지

POST 방식 요청: 폼에 내용 입력 -> 저장하기