자바스크립트(Javascript) 비동기 통신

Session 17

NEXT X LIKELION 이영찬

목차

1. 비동기 통신 개념

1. JS 비동기 처리와 비동기 통신
2. 동기 통신 vs 비동기 통신
3. AJAX란 무엇인가: 비동기 통신을 위한 기술
4. AJAX를 사용하는 3가지 방법

JS 비동기 처리와 비동기 통신

JS 비동기 처리vs 비동기 통신

비동기 처리

- 특정 작업이 완료될 때까지 기다리지 않고 다른 코드를 먼저 실행하는 프로그래밍 기법
- JavaScript에서 콜백 함수, 프로미스(Promise), setTimeout, async/await와 같은 기술을 사용하여 구현

비동기 통신

- 서버와 데이터를 주고받을 때, 페이지를 다시 로드하지 않고도 서버와 통신할 수 있는 방법을 의미(로드한다는 것을 클라이언트에서 다른 작업을 실행하지 못한다는 것)
- AJAX라는 비동기 통신 방법을 XMLHttpRequest, Fetch API, Axios 등을 통해서 구현

JS 비동기 처리와 비동기 통신

JS 비동기 처리

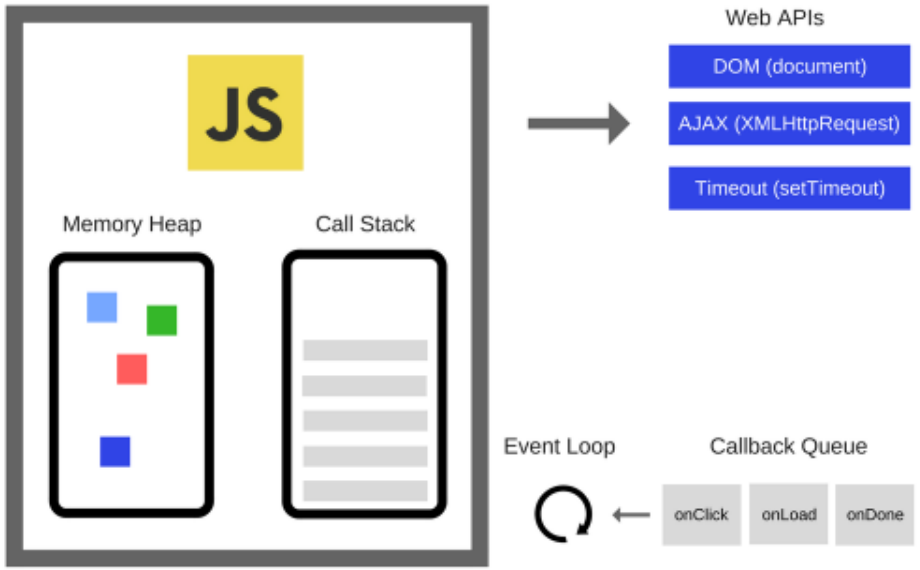
What the fxxx?

비동기 처리 :특정 작업이 완료될 때까지 기다리지 않고 다른 코드를 먼저 실행하는 프로그래밍 기법

Q. 그냥 하면 되는 거 아님? 이게 왜 중요한데? What the fxxx?

A. 자바스크립트의 동작원리를 보면, 자바스크립트는 싱글 스레드 언어이기 때문이야!

일손이 하나라는 뜻



=

JS 비동기 처리와 비동기 통신

동기처리 VS 비동기처리

동기 처리

- synchronous(동시에 발생하는)
- 요청과 결과가 동시에 일어나는 것을 말한다.
- 호출한 함수가 호출된 함수의 결과값을 계속해서 기다리고, 호출된 함수의 종료까지 직접 처리한다.

비동기 처리

- asynchronous(동시에 발생하지 않는)
- 요청과 결과가 동시에 일어나지 않는다.
- 호출한 함수가 호출된 함수의 결과값을 기다리지 않고, 다른 작업을 먼저 처리한 후 결과값이 종료된 것을 감지하고 처리한다.

간단한 비유

비동기 처리 및 통신

동기 처리

비동기 처리

주의할 점

비동기 처리 및 통신

JS 비동기 처리와 비동기 통신

JS 동기 처리

남은 집안일을 요청과 결과가 동시에 일어나는 **동기처리**로 수행한다면?

요청이 있으면 결과를 기다리는

- 빨래

손 빨래 40분 빨래 널기 10분

- 책상정리

책상정리 15분

- 바닥쓸기

바닥쓸기 20분

- 이불개기

이불개기 5분

소요 시간 총 **1시간 30**분!!

JS 비동기 처리와 비동기 통신

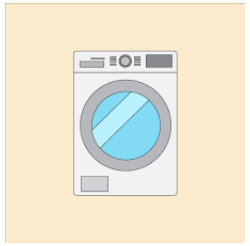
JS 비동기 처리

소요 시간 총 **50분**!!

남은 집안일을 요청과 결과가 동시에 일어나지 않는 **비동기처리**로 수행한다면?

요청이 있으면 결과를 기다리지 않고 다른 걸 먼저 실행하는

- 빨래



빨래 40분

빨래 널기 10분

- 책상정리

책상정리 15분

- 바닥쓸기

바닥쓸기 20분

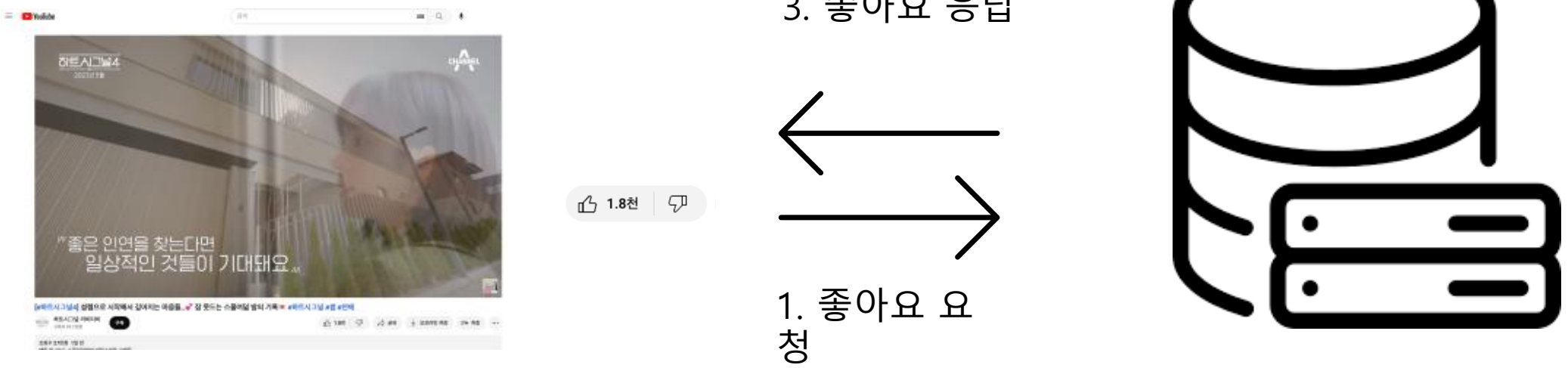
- 이불개기

이불개기 5분

동기 통신과 비동기 통신

JS 비동기 통신

비동기 통신 : 비동기 처리의 기법을 통신에서도 적용



동기 통신과 비동기 통신

JS 동기 통신

유튜브 영상의 “좋아요” 기능을 동기통신으로 처리한다면?

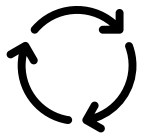
지금까지 장고에서 하던 모든 것들이 동기통신이다.

- 좋아요

좋아요 서버요청

- 영상재생

영상재생



재랜더링 영상 다시 재생

동기 통신과 비동기 통신

JS 동기 통신

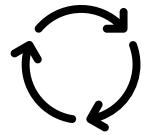
유튜브 영상의 “좋아요” 기능을 비동기통신으로 처리한다면?

좋아요 서버요청

- 좋아요

- 영상재생

영상재생



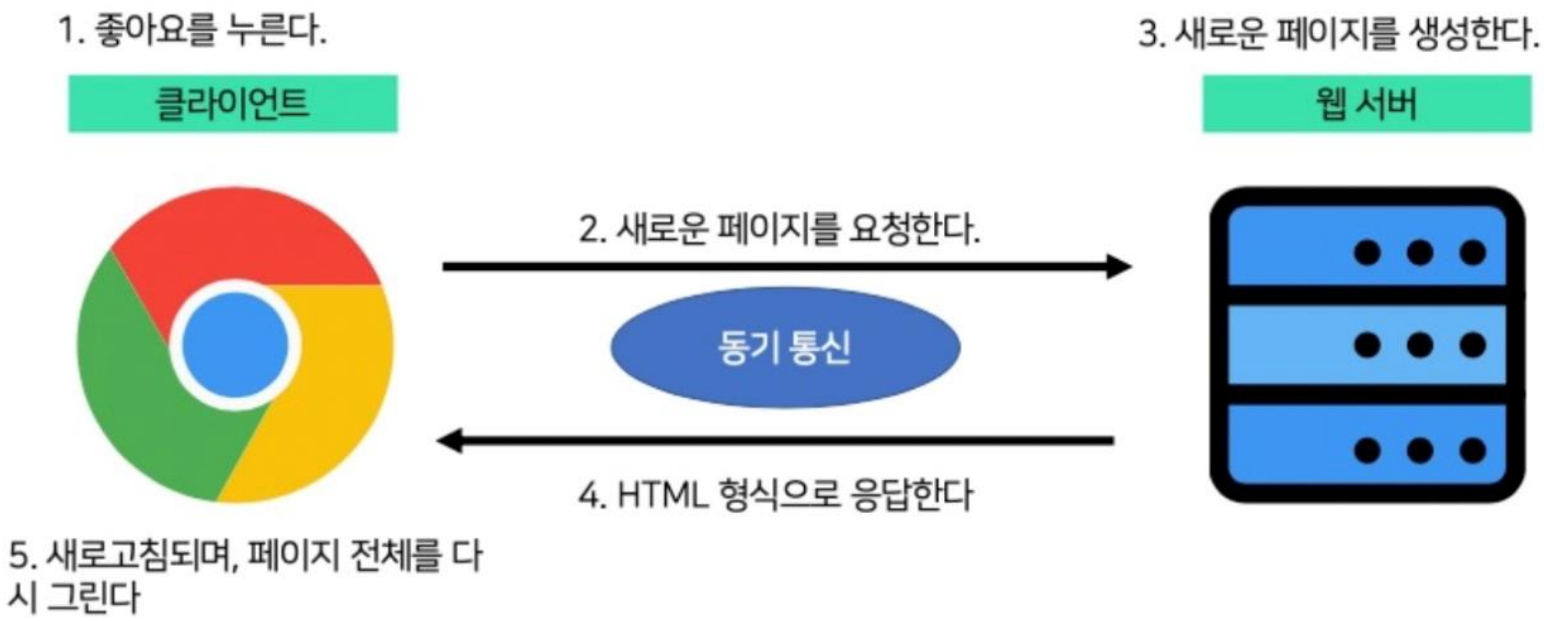
좋아요 서버 응답 받기

동기 통신과 비동기 통신

동기통신 VS 비동기통신

동기 통신

첫 렌더링 때, "좋아요"를 불러오고, "좋아요" 요청이오면, " 좋아요"를 업데이트를 하고, 새롭게 페이지를 랜더링한다.

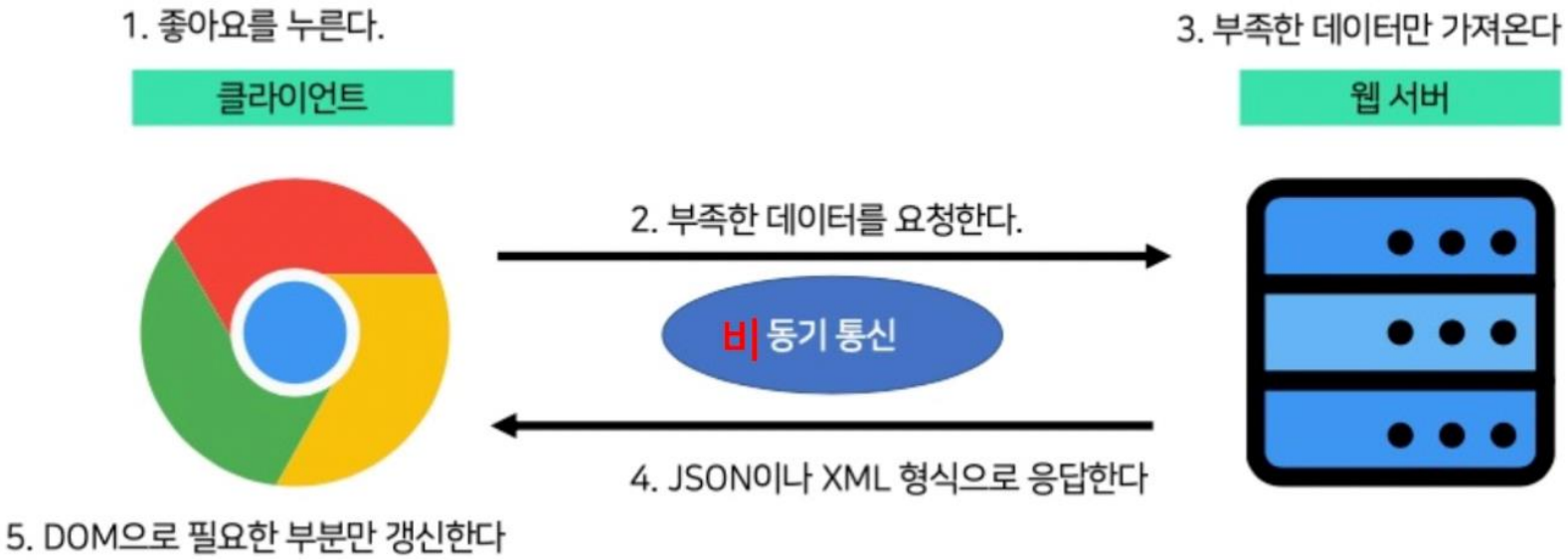


동기 통신과 비동기 통신

동기통신 VS 비동기통신

비동기 통신

"좋아요" 요청이 올 때, 새롭게 페이지를 랜더링하지 않고, "좋아요"만 업데이트한다.



동기 통신과 비동기 통신

동기통신 VS 비동기통신

동기 통신

문제점

1. 서버가 데이터를 보내기 전까지 클라이언트는 아무것도 할 수 없음
2. 웹 페이지 전체를 주고받기 때문에 불필요한 작업으로 인한 소요가 크다.

비동기 통신

특징

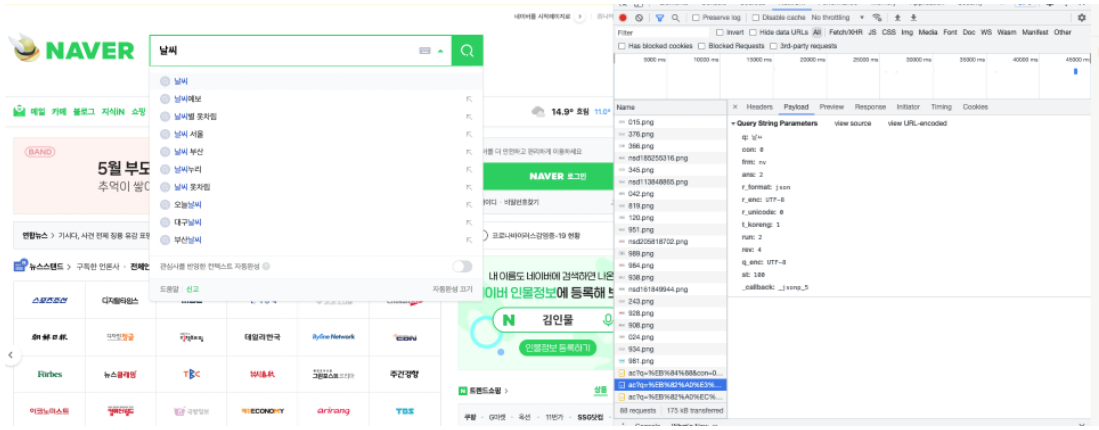
1. 필수 데이터로만 통신하기 때문에 처리 속도가 빠르고 서버 부하와 통신 트래픽 부하가 적다.
2. 비동기로 통신하기 때문에 각 영역에서 다른 작업을 처리할 수 있음.
3. 페이지를 전환하는 것이 아닌 페이지 일부만을 변경하기 때문에 빠른 렌더링이 가능함.

동기 통신과 비동기 통신

동기통신 VS 비동기통신

검색자동완성: 비동기통신+사용 예시

https://www.naver.com/



1. "날씨"를 검색한다고 할 때, 모든 음운마다 자동완성을 만들어줘야 한다. "ㄴ","ㅏ","ㄹ","ㅆ","ㅣ"
2. "ㄴ", "ㅏ" , "ㄹ", "ㅆ", "ㅣ" 모든 음운의 검색마다 자동완성되는 데이터를 가져와야 한다.
3. 모든 페이지를 렌더링하지 않고, 자동완성 검색에 해당하는 데이터만 업데이트되고 있다.

동기 통신과 비동기 통신

동기통신 VS 비동기통신

비동기 통신:정리

필수 데이터만 요청을 보내고, 응답을 받는 순간까지 클라이언트에서도 다른 작업을 계속해서 처리할 수 있는 것이 비동기 통신이다.

클라이언트(웹 애플리케이션)과 서버 간에 데이터를 교환하는 동안, 사용자 인터페이스와 상호작용을 계속할 수 있게 해주는 통신 방식이다

AJAX:비동기 통신을 위한 기술

AJAX개념 설명

AJAX: Asynchoronous Javascript Xml

비동기 자바스크립트 extensible Markup Language

- 자바스크립트가 **비동기 통신**을 도와주는 기술
- Ajax는 XMLHttpRequest라는 자바스크립트 객체를 활용해 웹 서버와 비동기로 통신하고 DOM을 이용하여 웹 페이지를 동적으로 갱신하는 프로그래밍 기법
- 다시말해, 웹페이지가 로딩된 후 일부 데이터를 서버로부터 요청할 수 있도록 함.
- Ajax를 위해서는 XMLHttpRequest 객체를 만들어야 하지만, 편의상 fetch API와 axios 라이브러리를 사용해 데이터를 송수신함.

AJAX:비동기 통신을 위한 기술

XML VS JSON

XML Json : 공통점

데이터를 저장하고, 교환하는데 사용되는 텍스트 기반의 포맷이다.

현재 Json이 더 많이 사용되는 이유

1. 간결성: JSON은 XML에 비해 간결한 구조를 가지고 있어, 데이터의 크기가 작아집니다. 이로 인해 데이터 전송 시 속도와 효율성이 높아집니다.
2. 가독성: JSON은 키-값 쌍으로 이루어져 있어, 읽기 쉽고 이해하기 쉽습니다.
3. 파싱 속도: JSON은 XML에 비해 빠른 파싱 속도를 가지고 있습니다. 이는 웹 서비스와 클라이언트 간의 데이터 교환에서 더 높은 성능을 제공합니다.
4. JavaScript 호환성: JSON은 JavaScript에서 원래 사용되도록 설계되었기 때문에, 웹 개발에서 JSON을 사용하면 JavaScript와의 호환성이 높아집니다.

AJAX:비동기 통신을 위한 기술

XML VS JSON

XML

확장 가능한 마크업 언어로, 데이터를 구조화하여 표현

HTML과 유사한 구조를 가지며, 사용자가 직접 태그를 정의

트리 구조로 이루어져 있으며, 각 요소는 시작 태그와 종료 태그로 구성

기계와 인간이 모두 이해할 수 있는 구조를 제공하지만, 상대적으로 복잡하고 김

```xml
<person>
    <name>John Doe</name>
    <age>30</age>
    <city>New York</city>
</person>
```

Json : Javascript Object Notation

경량의 데이터 교환 포맷으로, 주로 웹에서 데이터를 전송할 때 사용

객체 리터럴 문법을 기반으로 하지만, 여러 프로그래밍 언어에서 사용가능

키-값 쌍으로 구성된 데이터 구조를 사용하며, 배열과 객체를 중첩

간결하고 읽기 쉬운 구조를 가지며, 파싱 및 처리 속도 빠름

```json
{
    "person": {
        "name": "John Doe",
        "age": 30,
        "city": "New York"
    }
}
```

AJAX 사용하는 3가지 방법

XMLHTTPRequest()/Fetch Api/Axios Api

AJAX:

XMLHttpRequest, Fetch API, Axios 등 다양한 도구와 기술을 사용하여 구현 가능

XMLHttpRequest

AJAX 사용하는 3가지 방법

XMLHTTPRequest()

XMLHttpRequest

XMLHttpRequest

웹 브라우저에서 제공

XMLHttpRequest 객체를 통해 서버와 통신을 주고 받을 수 있도록 함

XMLHttpRequest는 HTTP 비동기 통신을 위한 메서드와 프로퍼티를 제공

AJAX 사용하는 3가지 방법

Fetch API

Fetch API

HTTP 통신을 위해서 사용

Javascript 내장 함수

XMLHttpRequest와 같은 역할을 수행

프로미스 기반의 문법

AJAX 사용하는 3가지 방법

AXIOS

AXIOS

HTTP 통신을 위해서 사용

서드 파티 라이브러리

프로미스 기반의 문법

브라우저와 Node js 환경에서 모두 사용할 수 있음

2. 비동기 통신으로 좋아요 구현하기

1. 동기 통신으로 “좋아요” 구현하기
2. 비동기 통신으로 “좋아요” 구현하기

동기 통신으로 “좋아요” 구현하기

환경세팅

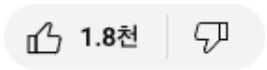
압축 푼 폴더에서 진행

```
$ pipenv shell
$ pipenv install django
$ cd project
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py createsuperuser
$ python manage.py runserver
```

"no such table" exception error 뜨는 경우:
db.splite3 제거 후
python manage.py migrate --run-syncdb 실행
이후 createsuperuser 이하로 이동

동기 통신으로 “좋아요” 구현하기

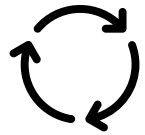
- 좋아요



좋아요 서버요청

- 영상재생

영상재생



재랜더링 영상 다시 재생

동기 통신으로 "좋아요" 구현하기

Like Model 추가

models.py

4 app/models.py

```
@@ -1,21 +1,25 @@
from django.contrib.auth.models import User
from django.db import models


class Post(models.Model):
    title = models.CharField(max_length=50)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts', default=1)


    def __str__(self):
        return self.title

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='comments')


    def __str__(self):
        return self.content
```

```
from django.contrib.auth.models import User
from django.db import models


class Post(models.Model):
    title = models.CharField(max_length=50)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts', default=1)


    def __str__(self):
        return self.title

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='comments')


    def __str__(self):
        return self.content

+
+ class Like(models.Model):
+     post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name="likes")
+     user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="likes")
```

$ python manage.py makemigrations app

$ python manage.py migrate

https://github.com/dhapdhap123/NEXT-session17/commit/6a9abc148e5b1823c046e03fed87685176222aed

동기 통신으로 "좋아요" 구현하기

Admin.py에 Model 등록

admin.py

3 app/admin.py

```
@@ -1,7 +1,8 @@
 from django.contrib import admin

-from .models import Comment, Post
+from .models import Comment, Post, Like

 # Register your models here.
 admin.site.register(Post)
 admin.site.register(Comment)
+admin.site.register(Like)
```

https://github.com/dhapdhap123/NEXT-session17/commit/6a9abc148e5b1823c046e03fed87685176222aed

동기 통신으로 "좋아요" 구현하기

detail.html에 좋아요 보여주기 + 좋아요 등록 form 추가

detail.html

6 ■■■■■ app/templates/detail.html

```
@@ -11,12 +11,18 @@
```

```
11      {% if user.is_authenticated and user.pk == post.author.pk %}
12      <a href="{% url 'edit' post.pk %}">수정하기</a>
13      <a href="{% url 'delete' post.pk %}">삭제하기</a>{% endif %}
14    </div>
15    <form action="" method="POST">
16      {% csrf_token %}
17      <input type="text" name="content" />
18      <button type="submit">댓글 작성</button>
19    </form>
20    {% for comment in post.comments.all %}
21    <li>
22      <span>{{comment.content}} </span>
```

```
11      {% if user.is_authenticated and user.pk == post.author.pk %}
12      <a href="{% url 'edit' post.pk %}">수정하기</a>
13      <a href="{% url 'delete' post.pk %}">삭제하기</a>{% endif %}
14  +   <span>좋아요 수 {{ like_length }}</span>
15    </div>
16    <form action="" method="POST">
17      {% csrf_token %}
18      <input type="text" name="content" />
19      <button type="submit">댓글 작성</button>
20    </form>
21  + <form action="{% url 'like' post.pk%}" method="POST">
22  +   {% csrf_token %}
23  +   <input type="hidden" name="post_pk" value="{{post_pk}}" />
24  +   <button type="submit">좋아요</button>
25  + </form>
26    {% for comment in post.comments.all %}
27    <li>
28      <span>{{comment.content}} </span>
```

https://github.com/dhapdhap123/NEXT-session17/commit/6a9abc148e5b1823c046e03fed87685176222aed

동기 통신으로 “좋아요” 구현하기

Url.py추가 url.py

project/urls.py

```
@@ -8,25 +8,26 @@
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from app import views
from django.contrib import admin
from django.urls import path

urlpatterns = [
   path('admin/', admin.site.urls),
   path('', views.home, name='home'),
   path('new/', views.new, name='new'),
   path('detail/<int:post_pk>', views.detail, name='detail'),
   path('edit/<int:post_pk>', views.edit, name='edit'),
   path('delete/<int:post_pk>', views.delete, name='delete'),
   path('delete-comment/<int:post_pk>/<int:comment_pk>',
views.delete_comment, name='delete_comment'),
   path("registration/signup/", views.signup, name="signup"),
   path("registration/login/", views.login, name="login"),
   path("registration/logout/", views.logout, name="logout"),
+    path("like/<int:post_pk>", views.like, name="like"),
]
```

https://github.com/dhapdhap123/NEXT-session17/commit/6a9abc148e5b1823c046e03fed87685176222aed

동기 통신으로 “좋아요” 구현하기

View.py

views.py

```
@@ -3,7 +3,7 @@
from django.contrib.auth.models import User
from django.shortcuts import redirect, render

- from .models import Comment, Post
+ from .models import Comment, Post, Like


def signup(request):
@@ -64,6 +64,7 @@ def new(request):
@login_required(login_url="/registration/login/")
def detail(request, post_pk):
    post = Post.objects.get(pk=post_pk)
+   likes = Like.objects.filter(post=post)

    if request.method == 'POST':
        content = request.POST['content']
@@ -72,9 +73,9 @@ def detail(request, post_pk):
            content=content,
            author=request.user
        )
-       return redirect('detail', post_pk);
+       return redirect('detail', post_pk)

-   return render(request, 'detail.html', {'post':post})
+   return render(request, 'detail.html', {'post':post, 'like_length': len(likes)})


def edit(request, post_pk):
@@ -103,4 +104,16 @@ def delete(request, post_pk):
def delete_comment(request, post_pk, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    comment.delete()
-   return redirect('detail', post_pk)
+   return redirect('detail', post_pk)
+
+def like(request, post_pk):
+   post = Post.objects.get(pk=post_pk)
+   user_like = Like.objects.filter(user=request.user, post=post)
+   if (len(user_like) > 0):
+       user_like.delete()
+       return redirect('detail', post_pk)
+   Like.objects.create(
+   post=post,
+   user=request.user
+   )
+   return redirect('detail', post_pk)
```

https://github.com/dhapdhap123/NEXT-session17/commit/6a9abc148e5b1823c046e03fed87685176222aed

동기 통신으로 "좋아요" 구현하기

View.py

views.like

like 함수는 좋아요를 눌렀을 때, 발생한다.

- 좋아요를 하지 않았으면, 좋아요를 생성
- 좋아요가 없으면, 좋아요를 취소

```
def like(request, post_pk):
        post = Post.objects.get(pk=post_pk)
        user_like = Like.objects.filter(user=request.user, post=post)
        if (len(user_like) > 0):
                user_like.delete()
                return redirect('detail', post_pk)
        Like.objects.create(
        post=post,
        user=request.user
        )
return redirect('detail', post_pk)
```

Like DB 중에서
- 현재 게시글과 현재 user의 Like 객체를 가져와

현재 존재하는 좋아요가 있어? 없어?

있으면 삭제해

없으면 생성해

동기 통신으로 “좋아요” 구현하기

View.py

views.detail

```
def detail(request, post_pk):
        post = Post.objects.get(pk=post_pk)
        likes = Like.objects.filter(post=post)

        if (request.method == 'POST'):
                Comment.objects.create(
                post=post,
                content=request.POST['content'],
                author=request.user
                )
        return redirect('detail', post_pk)
return render(request, 'detail.html', {'post': post, 'like_length': len(likes)})
```

→ Post에 있는 좋아요 갯수를 모두 가져오기

동기 통신으로 "좋아요" 구현하기

코드 개선하기

post.likes.count로 좋아요 개수 불러오기

detail.html

4 app/templates/detail.html

```
@@ -11,14 +11,14 @@
11      {% if user.is_authenticated and user.pk == post.author.pk %}
12      <a href="{% url 'edit' post.pk %}">수정하기</a>
13      <a href="{% url 'delete' post.pk %}">삭제하기</a>{% endif %}
14  -   <span>좋아요 수 {{ like_length }}</span>
```

```
@@ -11,14 +11,14 @@
11      {% if user.is_authenticated and user.pk == post.author.pk %}
12      <a href="{% url 'edit' post.pk %}">수정하기</a>
13      <a href="{% url 'delete' post.pk %}">삭제하기</a>{% endif %}
14  +   <span>좋아요 수 {{ post.likes.count }}</span>
```

views.py

9 app/views.py

```
@@ -63,9 +63,7 @@ def new(request):
63
64  @login_required(login_url="/registration/login/")
65  def detail(request, post_pk):
66  -   post = Post.objects.get(pk=post_pk)
67  -   likes = Like.objects.filter(post=post)
68  -
69      if request.method == 'POST':
70          content = request.POST['content']
71          Comment.objects.create(
@@ -75,7 +73,7 @@ def detail(request, post_pk):
75          )
76          return redirect('detail', post_pk)
77
78  -   return render(request, 'detail.html', {'post':post, 'like_length': len(likes)})
```

```
@@ -63,9 +63,7 @@ def new(request):
63
64  @login_required(login_url="/registration/login/")
65  def detail(request, post_pk):
66  +   post = Post.objects.get(pk=post_pk)
67      if request.method == 'POST':
68          content = request.POST['content']
69          Comment.objects.create(
@@ -75,7 +73,7 @@ def detail(request, post_pk):
73          )
74          return redirect('detail', post_pk)
75
76  +   return render(request, 'detail.html', {'post':post})
```


https://github.com/dhapdhap123/NEXT-session17/commit/3696ff76f6ce01f2dfdca6cc129347fdd8df971d

동기 통신으로 “좋아요” 구현하기

코드 개선하기

post.pk를 value로 보내서 값 변경하기

detail.html

```
19        <button type="submit">댓글 작성</button>
20      </form>
21  -   <form action="{% url 'like' post.pk%}" method="POST">
22        {% csrf_token %}
23        <input type="hidden" name="post_pk" value="{{post_pk}}" />
24        <button type="submit">좋아요</button>
```

```
19        <button type="submit">댓글 작성</button>
20      </form>
21  +   <form action="{% url 'like' %}" method="POST">
22        {% csrf_token %}
23        <input type="hidden" name="post_pk" value="{{post_pk}}" />
24        <button type="submit">좋아요</button>
```

views.py

```
109 - def like(request, post_pk):
110       post = Post.objects.get(pk=post_pk)
111       user_like = Like.objects.filter(user=request.user, post=post)
112       if (len(user_like) > 0):
```

```
107 + def like(request):
108 +     post_pk = request.POST.get('post_pk')
109       post = Post.objects.get(pk=post_pk)
110       user_like = Like.objects.filter(user=request.user, post=post)
111       if (len(user_like) > 0):
```

urls.py

project/urls.py

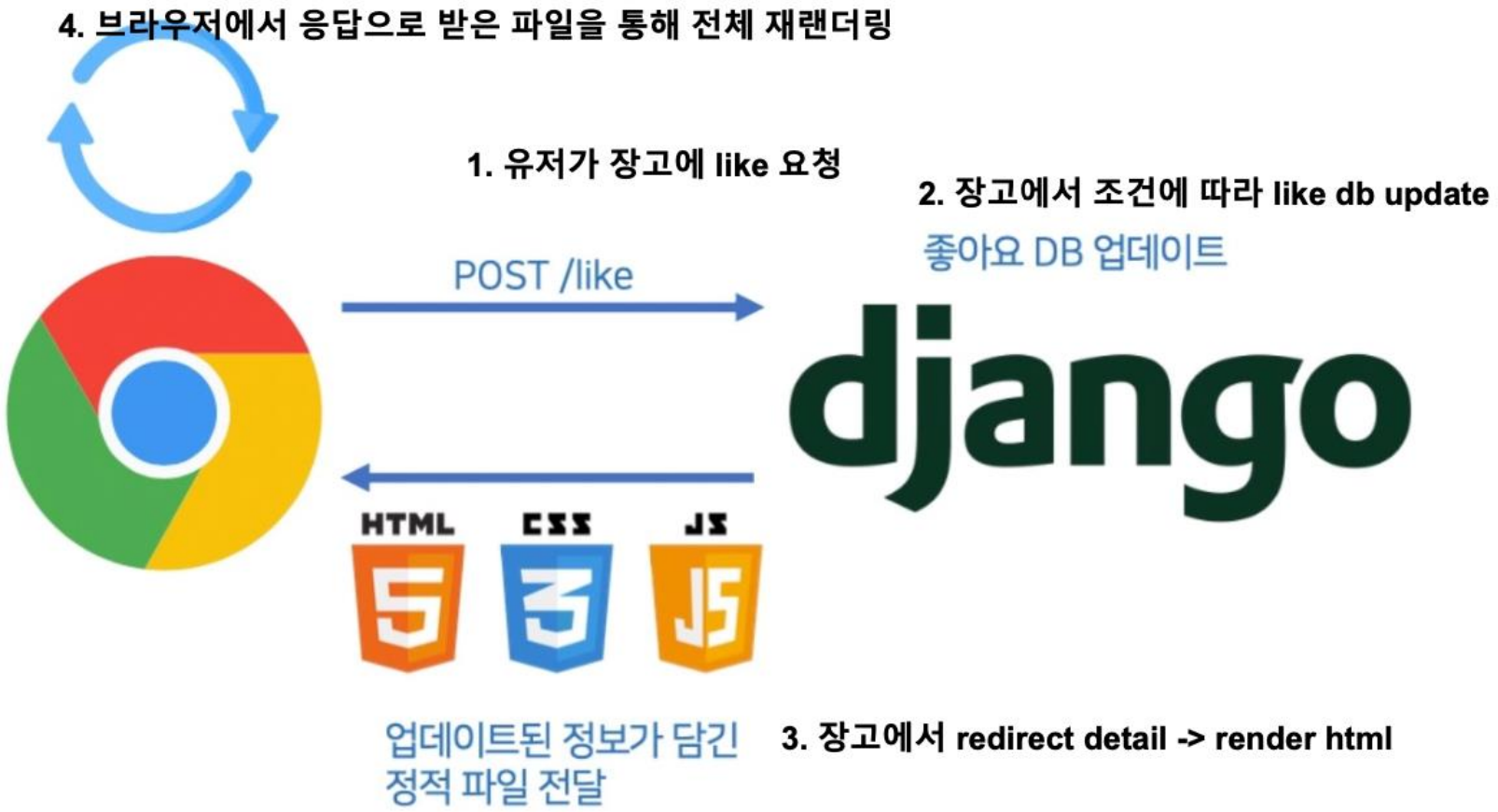
```
@@ -28,6 +28,6 @@
28      path("registration/signup/", views.signup, name="signup"),
29      path("registration/login/", views.login, name="login"),
30      path("registration/logout/", views.logout, name="logout"),
31  -   path("like/<int:post_pk>", views.like, name="like"),
32    ]
33
```

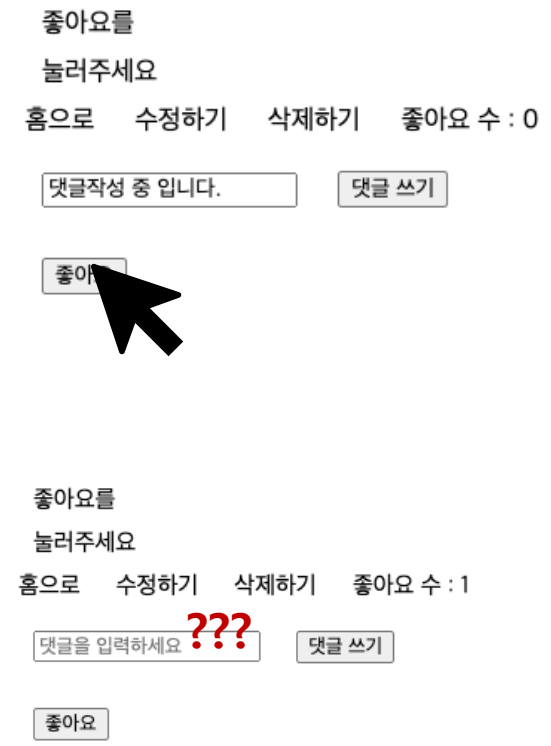
```
28      path("registration/signup/", views.signup, name="signup"),
29      path("registration/login/", views.login, name="login"),
30      path("registration/logout/", views.logout, name="logout"),
31  +   path("like", views.like, name="like"),
32    ]
33
```

https://github.com/dhapdhap123/NEXT-session17/commit/3696ff76f6ce01f2dfdca6cc129347fdd8df971d

동기 통신으로 “좋아요” 구현하기

동기 통신의 동작 방식





동기 통신으로 “좋아요” 구현하기

View.py

동기 통신의 단점

- 서버가 데이터를 보내기 전까지 클라이언트는 아무것도 할 수 없음
- 클라이언트의 요청에 서버가 응답하면 웹 페이지를 렌더링할 때, 모든 데이터를 새롭게 받아옴

좋아요만 빠르게 업데이트하고 싶은데?

이러한 단점을 해결하기 위해 **비동기 통신**이 등장함

비동기 통신으로 “좋아요” 구현하기

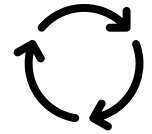
비동기 통신으로 “좋아요” 구현하기

- 좋아요

좋아요 서버요청

- 영상재생

영상재생



좋아요 서버 응답 받기

비동기 통신으로 “좋아요” 구현하기

알아야 할 사전 개념

비동기 통신을 위한 사전 개념

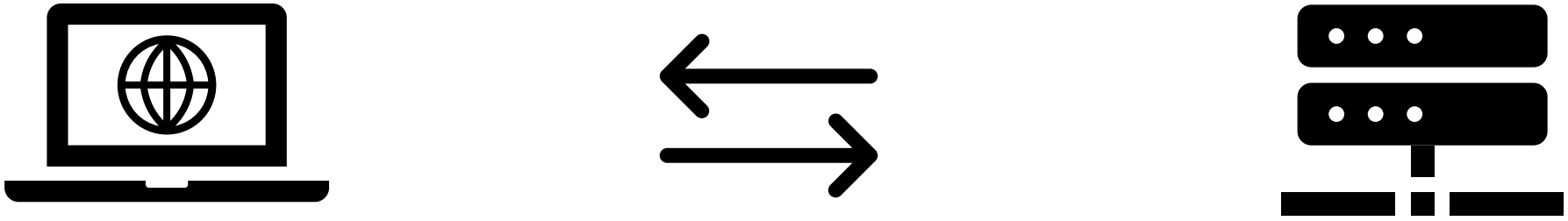
1. Django와 Javascript에서 **JSON** 다루기
2. 저번 세션에서 다루었던 **DOM** Control

비동기 통신으로 “좋아요” 구현하기

View.py

다시한번 JSON 이 왜 필요해??

- 클라이언트와 서버가 데이터를 주고 받기 위한 포맷이 필요함
- django에서는 컨텍스트 변수에 데이터를 넣어주면, Django 템플릿 엔진에서 자체적으로 변수를 처리하고, HTML 코드에 적합한 형태로 삽입해줌 `return render(request, ‘home.html’, {“post”:post})`
- Javascript로 통신을 하거나 일반적인 통신 형태에는 데이터를 주고 받기 위한 포맷인 json이 필요함



비동기 통신으로 “좋아요” 구현하기

View.py

JSON 변환하기

Json은 텍스트 기반의 포맷이므로 언어에 맞게 변환이 필요함

JSON

- JSON.stringify() : 자바스크립트 객체를 JSON 문자열로 변환
- JSON.parse() : JSON 문자열을 자바스크립트 객체로 환원

- json.dumps() : dict 형식을 JSON 문자열로 변환
- json.loads() : JSON 문자열을 dict 형식으로 환원

```
{
"person":
        {
                "name": "John Doe",
                "age": 30,
                "city": "New York"
        }
 }
```

비동기 통신으로 “좋아요” 구현하기

비동기 통신의 동작 방식

비동기 통신으로 “좋아요” 구현하기

detail.html

좋아요 form 삭제

```
<form action="{% url 'like' post.pk %}" method="POST">
  {% csrf_token %}
  <input type="hidden" name="post_pk" value="{{post.pk}}" />
  <button type="submit">좋아요</button>
</form>
```

<form></form>은 입력된 데이터를 한 번에 서버로 전송함

<form>내부에 있는 <button type=“submit”/>을 누르면 데이터가 서버에 전송됨

좋아요 button 추가

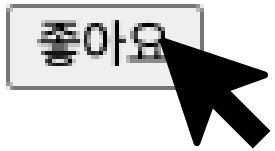
```
<button class="like-button">좋아요</button>
```

이제 button을 클릭하여 통신할 수 있도록 변경할 것이다.

비동기 통신으로 “좋아요” 구현하기

그냥 button

button 을 클릭해보자!



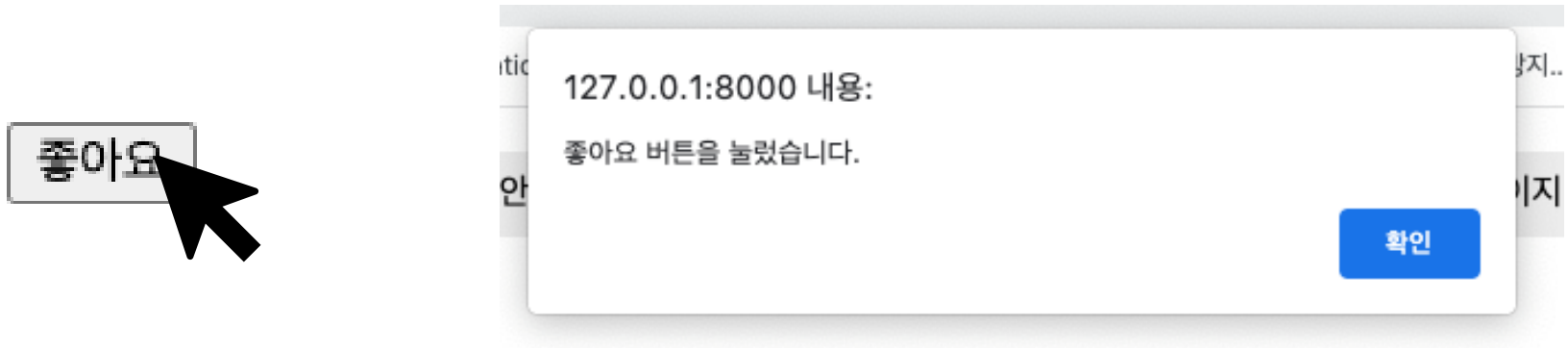
Dom 요소를 조작하여
button에 onClick이벤트를 등록해보자!

비동기 통신으로 “좋아요” 구현하기

버튼에 직접 onClick 추가하기

button에 inline 방식으로 onClick 이벤트 등록

```
<button class="like-button" onClick="alert('좋아요 버튼을 눌렀습니다.')">좋아요</button>
```



비동기 통신으로 “좋아요” 구현하기

View.py

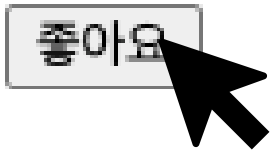
button에 addEventListener를 사용하여 onClick 이벤트 등록

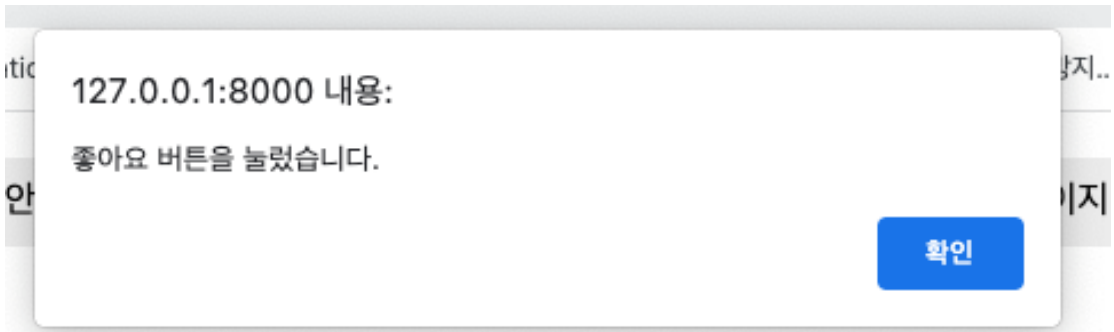
```
<button class="like-button">좋아요</button>
```

```
<script>
        const likeButton = document.querySelector(".like-button");

        const handleLike = () => {
                alert("좋아요 버튼을 눌렀습니다.");
        };

        likeButton.addEventListener("click", handleLike);
</script>
```





비동기 통신으로 “좋아요” 구현하기

Fetch api

이제 버튼 클릭은 되니까, **Fetch API**로 통신하는 함수를 작성해 보자.

뭐였지?

HTTP 통신을 위해서 사용

Javascript 내장 함수

XMLHttpRequest와 같은 역할을 수행

프로미스 기반의 문법

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

비동기 통신으로 “좋아요” 구현하기

Fetch api vs form 통신 비교하기


```
<form action="{% url 'like' post.pk %}" method="POST">
  {% csrf_token %}
  <input type="hidden" name="post_pk" value="{{post.pk}}" />
  <button type="submit">좋아요</button>
</form>
```

```
<form action="{% url 'like' %}"
```

=

```
fetch('https://example.com/api/data',
```

```
method="POST">
```

=

```
method: 'POST',
```

```
fetch('https://example.com/api/data',{
        method: 'POST',
        headers: {
                'Content-Type': 'application/json'
        },
        body: JSON.stringify({
                key1: 'value1'
                })
        })
        .then(response => {
         if (!response.ok) {
                throw new Error('Network response was not ok');
        }
        return response.json();
        })
        .then(data => {
                console.log(data);
        })
        .catch(error => {
                console.error('There was a problem with the fetch operation:', error);
});
```

비동기 통신으로 “좋아요” 구현하기

Fetch api vs form 통신 비교하기


```
<form action="{% url 'like' post.pk %}" method="POST">
  {% csrf_token %}
  <input type="hidden" name="post_pk" value="{{post.pk}}" />
  <button type="submit">좋아요</button>
</form>
```

```
<input type="hidden" name="post_pk" value="{{post.pk}}" />
```

=

```
body: JSON.stringify({
        key1: 'value1'
        })
})
```

```
fetch('https://example.com/api/data',{
        method: 'POST',
        headers: {
                'Content-Type': 'application/json'
        },
        body: JSON.stringify({
                key1: 'value1'
                })
        })
        .then(response => {
         if (!response.ok) {
                throw new Error('Network response was not ok');
        }
        return response.json();
        })
        .then(data => {
                console.log(data);
        })
        .catch(error => {
                console.error('There was a problem with the fetch operation:', error);
});
```

비동기 통신으로 “좋아요” 구현하기

Fetch Api 사용하기 / detail.html

18 app/templates/detail.html

```
@@ -11,7 +11,7 @@
      {% if user.is_authenticated and user.pk == post.author.pk %}
      <a href="{% url 'edit' post.pk %}">수정하기</a>
      <a href="{% url 'delete' post.pk %}">삭제하기</a>{% endif %}
-     <span>좋아요 수 {{ post.likes.count }}</span>
+     <span class="like-count">좋아요 수 {{ post.likes.count }}</span>
    </div>
    <form action="" method="POST">
      {% csrf_token %}
@@ -31,8 +31,22 @@
  </div>
  <script>
    const likeButton = document.querySelector(".like-button");
+   const likeCount = document.querySelector(".like-count");
+
    const handleLike = () => {
-     alert("좋아요 버튼을 눌렀습니다.");
+     fetch("/like", {
+       method: "POST",
+       body: JSON.stringify({
+         post_pk: "{{ post.pk }}",
+       }),
+       headers: {
+         "Content-Type": "application/json",
+       }
+     })
+     .then((response) => response.json())
+     .then((data) =>
+       likeCount.innerHTML = `좋아요 ${data.like_count}개`
+     )
    };
    likeButton.addEventListener("click", handleLike);
  </script>
```

https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

Fetch Api 사용하기 / detail.html

Fetch API 적용하기

요청보내기

url: “/like” 어디에 보냄?

method: “post” 어떻게 보냄?

Javascript 객체를 json으로 변경

무엇을 보냄?

Body: JSON.stringify({post_pk:“{{post.pk}}”,

요청받기

respons.json() JSON을 javascript로 변환

innerHtml text 써주기

likeCount.innerHtml = ’좋아요 ${data.like-count}개`;

data로 담은 걸 가져오기

```
28 + <script>
29 +   const likeButton = document.querySelector(".like-button");
30 +   const likeCount = document.querySelector(".like-count");
31 +
32 +   const handleLike = () => {
33 +     fetch("/like", {
34 +       method: "POST",
35 +       body: JSON.stringify({
36 +         post_pk: "{{ post.pk }}",
37 +       }),
38 +       headers: {
39 +         "Content-Type": "application/json",
40 +       },
41 +     })
42 +       .then((response) => response.json())
43 +       .then((data) => {
44 +         likeCount.innerHTML = `좋아요 ${data.like_count}개`;
45 +       });
46 +   };
47 +
48 +   likeButton.addEventListener("click", handleLike);
49 + </script>
50 +
```

```
<span class="like-count">좋아요 수 : {{ post.likes.count }} </span>
```


https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

views.py 작성하기

34 app/views.py

```
@@ -2,6 +2,9 @@
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.shortcuts import redirect, render
+from django.http import HttpResponse
+import json
+from django.views.decorators.csrf import csrf_exempt

from .models import Comment, Post, Like

@@ -104,15 +107,24 @@ def delete_comment(request, post_pk, comment_pk):
    comment.delete()
    return redirect('detail', post_pk)

+@csrf_exempt
def like(request):
-    post_pk = request.POST.get('post_pk')
-    post = Post.objects.get(pk=post_pk)
-    user_like = Like.objects.filter(user=request.user, post=post)
-    if (len(user_like) > 0):
-        user_like.delete()
-        return redirect('detail', post_pk)
-    Like.objects.create(
-    post=post,
-    user=request.user
-    )
-    return redirect('detail', post_pk)
+    if request.method == 'POST':
+        request_body = json.loads(request.body)
+        post_pk = request_body['post_pk']
+        post = Post.objects.get(pk=post_pk)
+        user_like = Like.objects.filter(user=request.user, post=post)
+
+
+        if (len(user_like) > 0):
+            user_like.delete()
+        else:
+            Like.objects.create(
+                post=post,
+                user=request.user
+            )
+
+        response = {
+            'like_count': post.likes.count(),
+        }
+        return HttpResponse(json.dumps(response))
```

https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

View.py

Import 하기

json

- json.dumps() : dict 형식을 JSON 문자열로 변환
- json.loads() : JSON 문자열을 dict 형식으로 환원

```
1   import json
2
3   from django.contrib import auth
4   from django.contrib.auth.decorators import login_required
5   from django.contrib.auth.models import User
6 + from django.http import HttpResponse
7   from django.shortcuts import redirect, render
8 + from django.views.decorators.csrf import csrf_exempt
9
```

httpResponse

Django의 HTTP 응답 클래스로, 웹 브라우저가 요청한 데이터를 담아 돌려주는 객체

@crsf_exempt

특정 뷰 함수에서 CSRF 보호를 비활성화하는 데 사용됩니다. 이 데코레이터를 사용하면 해당 뷰 함수는 CSRF 토큰 확인 과정을 건너뛰고 요청을 처리

```
120 + @csrf_exempt
121   def like(request):
122 +     if request.method == 'POST':
123 +         request_body = json.loads(request.body)
```

```
<form method="POST">
  {% csrf_token %}
  <input type="text"
  <button type="subm
```

csrf_token은 템플릿 태그로, Django 템플릿에서 HTML <form> 요소에 CSRF 토큰을 삽입하는 데 사용됩니다. 이 태그를 사용하면, 클라이언트가 폼을 제출할 때 자동으로 CSRF을 포함함

https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

View.py

views 작성하기

요청받기

무엇을 받음?

request_body = json. loads(request.body),

Json을 딕셔너리로 변환

응답보내기

무엇을 보낼 준비?

response= {'like_count': post.likes.count()},

무엇을 보낼 준비?

return HttpResponse(json.dumps.(response))

딕셔너리를 json으로 변환

Body: JSON.stringify({post_pk:"{{post.pk}}"

```
+ @csrf_exempt
  def like(request):
+     if request.method == 'POST':
+         request_body = json.loads(request.body)
+         post_pk = request_body['post_pk']
+         post = Post.objects.get(pk=post_pk)
+         user_like = Like.objects.filter(user=request.user, post=post)
+
+         if (len(user_like) > 0):
+             user_like.delete()
+         else:
+             Like.objects.create(
+                     post=post,
+                     user=request.user
+             )
+
+         response = {
+             'like_count': post.likes.count(),
+         }
+         return HttpResponse(json.dumps(response))
```


https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

View.py

views 작성하기

응답보내기

return HttpResponse(json.dumps.(response))

JSON or 텍스트를 담아서 클라이언트에게 응답

return render(request, ‘foo.html', { “data_name": data })

Html과 컨텍스트 데이터를 사용하여 HTML 문서를 생성하고, HttpResponse 객체로 래핑하여 반환함

return redirect('home')

함수는 클라이언트를 다른 URL로 리다이렉션하는 응답

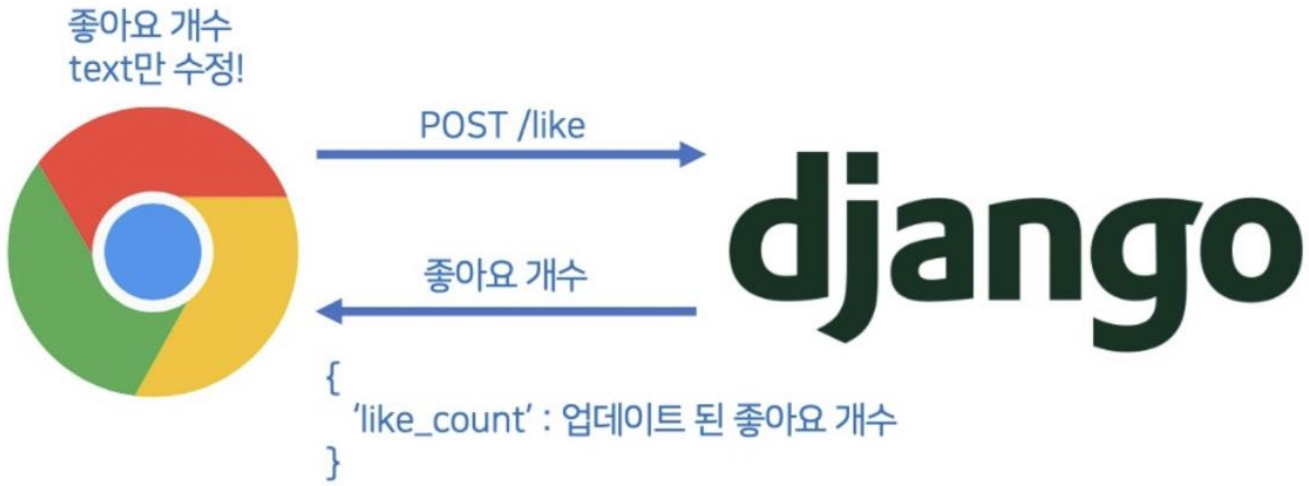
```
+ @csrf_exempt
  def like(request):
+     if request.method == 'POST':
+         request_body = json.loads(request.body)
+         post_pk = request_body['post_pk']
+         post = Post.objects.get(pk=post_pk)
+         user_like = Like.objects.filter(user=request.user, post=post)
+
+         if (len(user_like) > 0):
+             user_like.delete()
+         else:
+             Like.objects.create(
+                     post=post,
+                     user=request.user
+             )
+
+         response = {
+             'like_count': post.likes.count(),
+         }
+         return HttpResponse(json.dumps(response))
```


https://github.com/dhapdhap123/NEXT-session17/commit/32be87f9e6f0559a153634ef27c34d51688c3c3f

비동기 통신으로 “좋아요” 구현하기

View.py

좋아요를 비동기 통신으로 구현



비동기 통신으로 “좋아요” 구현하기

View.py

Fetch 대신 axios?

- **https://github.com/axios/axios**

- **Axios의 장점**
  1. **JSON data를 자동으로 변환해줌**
  2. **Nodejs에서도 사용할 수 있음**
  3. **문법이 간결함**

비동기 통신으로 “좋아요” 구현하기

View.py

Axios 활용하기

스크립트 추가

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const likeButton = document.querySelector(".like-button");
  const likeCount = document.querySelector(".like-count");

  const handleLike = () => {
    axios('/like', {
      method: 'POST',
      data: {post_pk: "{{post.pk}}"}
    }).then(res => likeCount.innerHTML = "좋아요 수 " + res.data.like_count);
  }

  likeButton.addEventListener("click", handleLike);
</script>
```

비동기 통신으로 “좋아요” 구현하기

View.py

Axios 더 간결하게 쓰기

Axios.[method](url, data, headers)

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const likeButton = document.querySelector(".like-button");
  const likeCount = document.querySelector(".like-count");

  const handleLike = () => {
    axios.post('/like', {post_pk: "{{post.pk}}"})
    .then(res => likeCount.innerHTML = "좋아요 수 " + res.data.like_count);
  }

  likeButton.addEventListener("click", handleLike);
</script>
```

비동기 통신으로 “좋아요” 구현하기

View.py

Promise.then 대신 async/await

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const likeButton = document.querySelector(".like-button");
  const likeCount = document.querySelector(".like-count");

  const handleLike = async () => {
    try {
      const res = await axios.post('/like', {post_pk: "{{post.pk}}"})
      likeCount.innerHTML = "좋아요 수 " + res.data.like_count
    } catch (err) {
      console.error(err);
    }
  }

  likeButton.addEventListener("click", handleLike);
</script>
```

- Promise는 비동기 작업의 결과를 나타내는 객체로, 성공한 경우(resolve)나 실패한 경우(reject)에 대한 처리를 간결하게 작성할 수 있음.
- Promise를 사용하면 코드가 중첩되고 복잡해질 수 있음

이에 더 나은 해결책으로 async/await가 도입

- async/await는 비동기 작업을 처리할 때 코드를 동기적인 방식처럼 작성할 수 있게 해주는 문법
- async/await로 Promise를 간결하고 가독성있게 작성가능

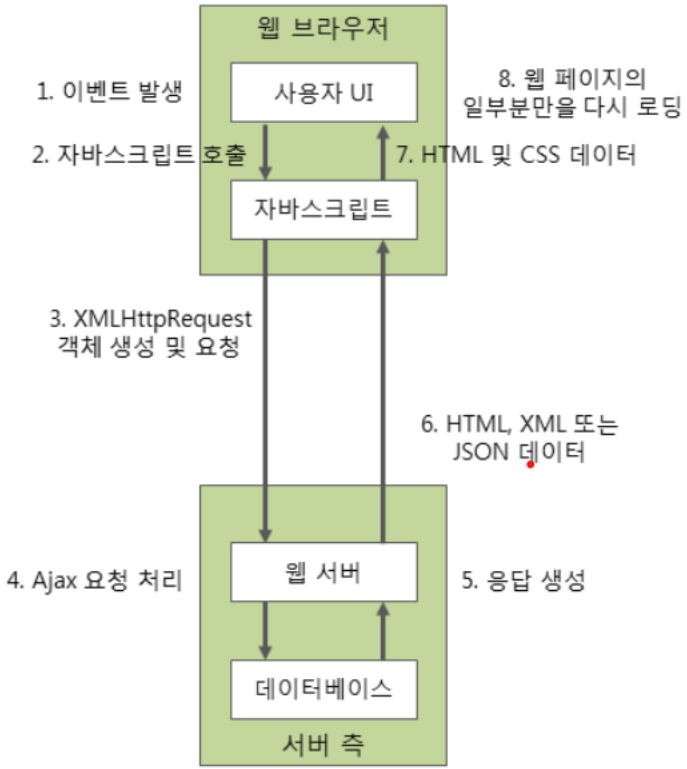
비동기 통신으로 “좋아요” 구현하기

View.py

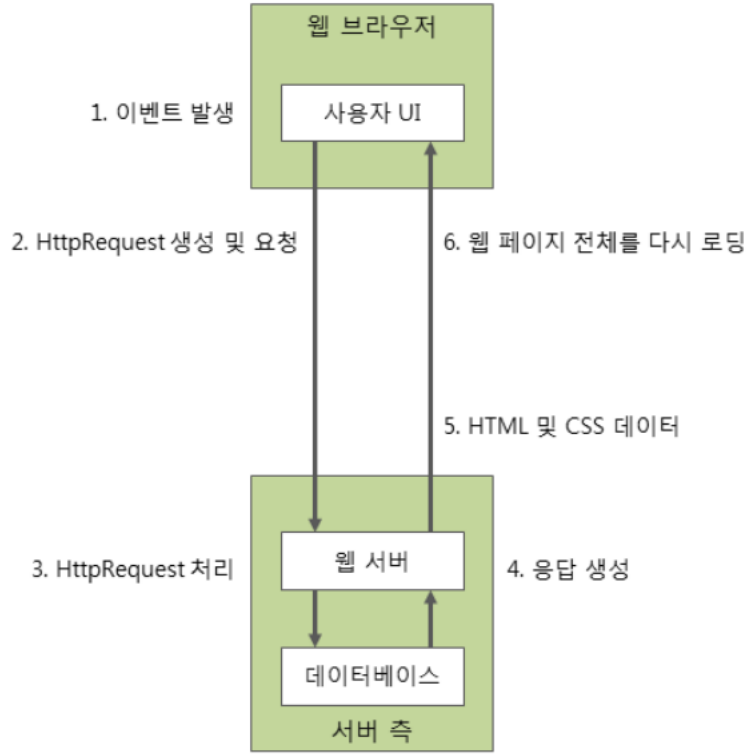
동기 통신
1. Html 정적 파일 수신
2. 새로운 요청시 매번 새로 고침

비동기 통신
1. JSON 송수신
2. 필요한 부분만 DOM 수정

< Ajax를 이용한 웹 응용 프로그램의 동작 원리 >



< 기존 웹 응용 프로그램의 동작 원리 >



과제
할 수 있다!

비동기 통신 마스터하기

과제 내용

필수

댓글을 비동기 통신으로 수행하기

Axios를 활용해서 비동기 통신해보기

선택

좋아요를 누른 게시글은 빨간색

좋아요를 누르지 않은 게시글은 검은색으로 하기

제출 방법

이름

본인 레포지토리에 git push한 링크

영상

제출 기한

다음 세션까지 => 5월 16일(목)