JavaScript 비동기

13번째 세션

NEXT X LIKELION 이한주

Introduction

- HTTP란?
- Ajax
- 비동기 vs 동기
- Fetch or Axios
- JSON

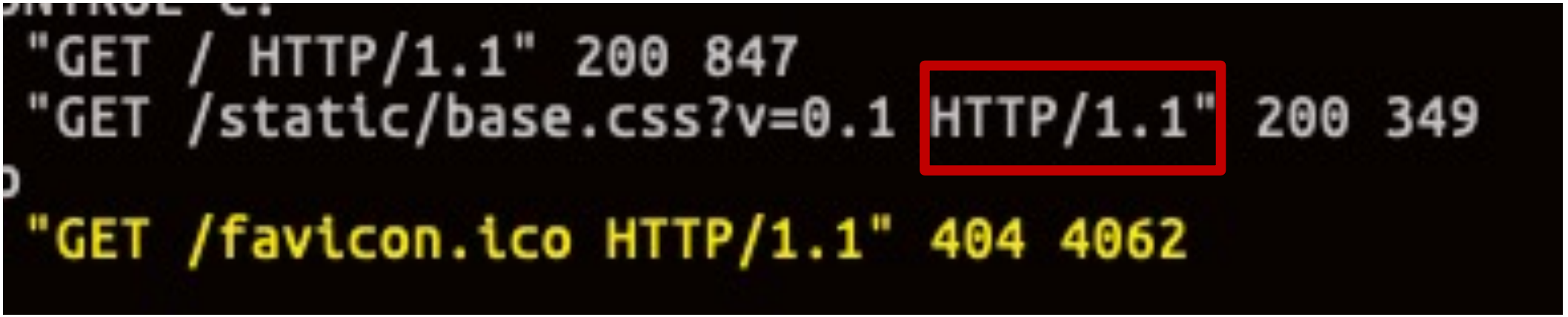
HTTP란? (Hyper Text Transfer Protocol)

- HTTP(Hyper Text Transfer Protocol)는 웹 브라우저와 웹 서버가 웹 페이지나 동영상, 음성 파일 등을 주고받기 위한 프로토콜(통신규약)

- 주로 HTML 문서를 주고받는데 쓰인다.

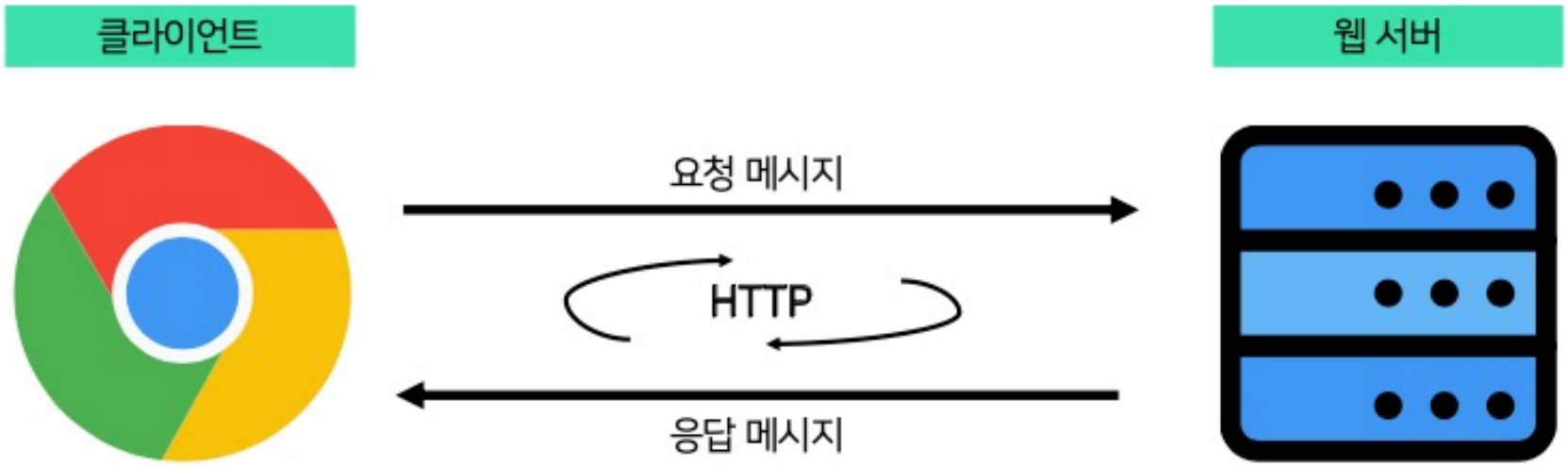
HTTP 어디서 많이 봤는데?

- 여러분들이 개발하던 Django terminal 창에서 보셨을겁니다.



HTTP 통신이란?

- HTTP에서는 클라이언트가 서버에 요청 메시지를 보내고 이에 대해 서버가 응답 메시지를 반환.



심화 (궁금하면 검색)

- HTTP에서는 전송 계층 프로토콜로 TCP를 사용하고 네트워크 계층 프로 토콜로 IP를 사용하는 것이 일반적. 이 두 계층을 합쳐서 TCP/IP라 는 이름으로 부름.

- TCP/IP에서는 IP주소를 사용해서 통신할 컴퓨터를 결정합니다. 그리고 포 트 번호를 사용해서 그 컴퓨터의 어떤 프로그램과 통신할지를 결정합니다.

- HTTP에서는 기본적으로 80번 포트를 사용합니다.

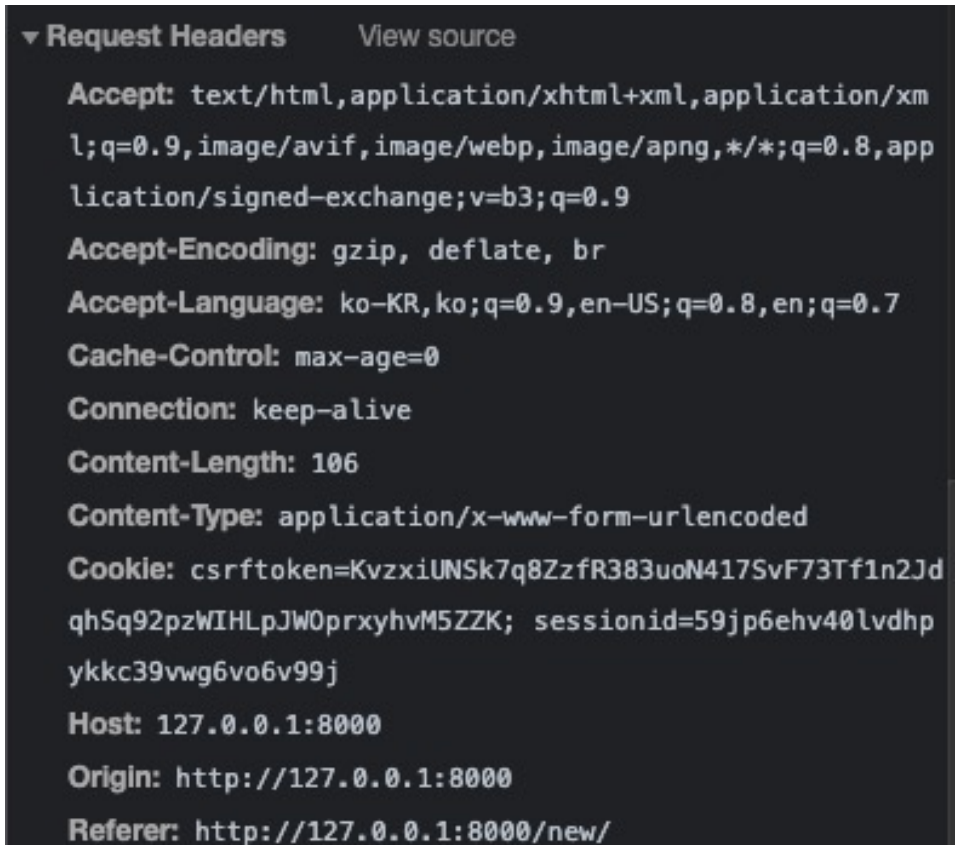
HTTP 요청 메세지와 응답 메세지

- HTTP 요청 메시지는 요청 행, 요청 헤더, 메시지 본문 이라는 세 부분으로 구성
- HTTP 응답 메시지는 응답 행, 응답 헤더, 메시지 본문 이라는 세 부분으로 구성

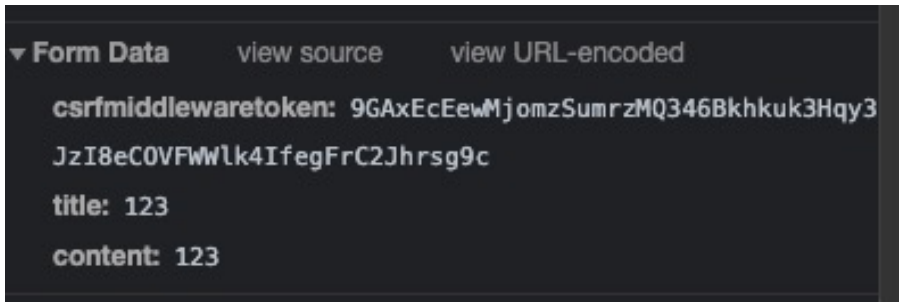
HTTP 요청 메세지 예시

- 장고에서 글 생성시

요청헤더



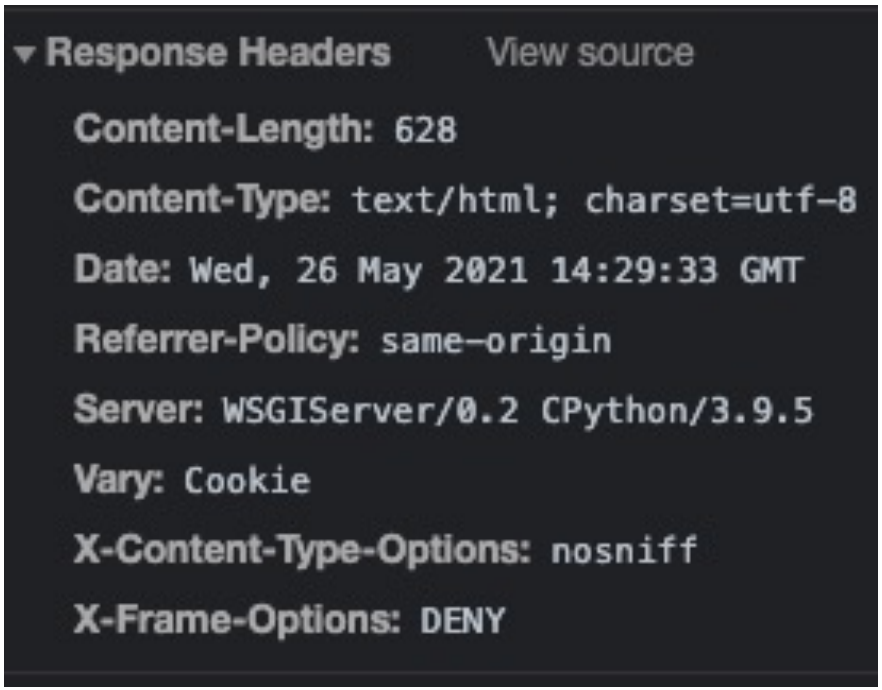
요청 메세지 본문



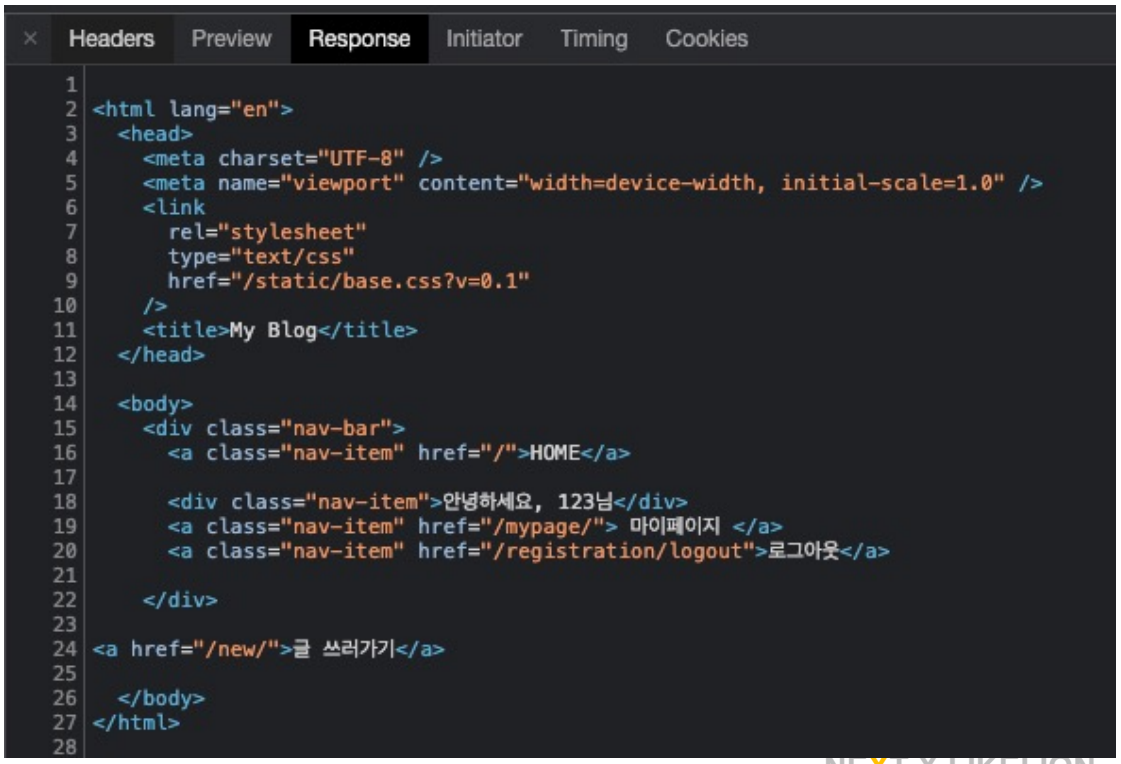
HTTP 응답 메세지 예시

- http://localhost:8000 으로 접속시

응답헤더



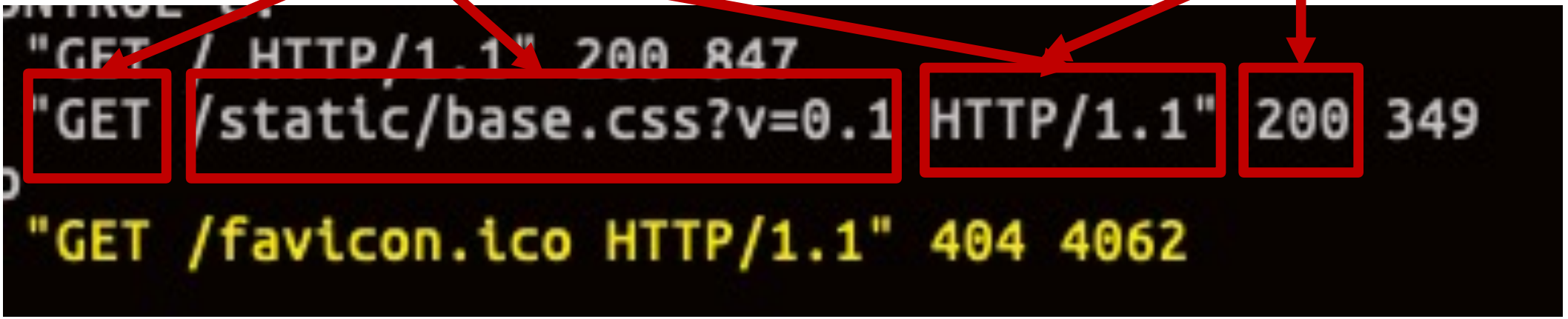
응답 메세지



HTTP 요청과 응답 메세지 예시

요청 메세지
요청 메소드 / 요청 URL / HTTP 버전

응답 메세지
HTTP 버전 / 응답 행



HTTP의 주용 상태 코드와 상태 설명 (약속)

- 참고)

분류	상태 코드	상태 설명	내용
성공	200	OK	요청을 성공함
클라이언트 오류	401	Unauthorized	인증되지 않음
클라이언트 오류	403	Forbidden	액세스가 허용되지 않음
클라이언트 오류	404	Not found	요청한 리소스를 찾지 못함
클라이언트 오류	408	Request timeout	요청 시간을 초과함
서버 오류	500	Internal server error	서버 내부에서 오류가 발생함
서버 오류	503	Service unavailable	서비스를 일시적으로 사용할 수 없음

Ajax (Asynchronous JavaScript + XML) 란

- Ajax란 XMLHttpRequest 라는 자바스크립트의 객체를 이용하여
  웹 서버와 비동기로 통신하고 DOM을 이용하여 웹 페이지를
  동적으로 갱신하는 프로그래밍 기법

- 참고로, Asynchronous란 '비동기'란 뜻.

- 편의상, XMLHttpRequest 객체를 직접 만들지 않고
  fetch API와 axios 라이브러리를 사용해 데이터를 송수신.

- jQuery ajax는 다루지 않습니다. (구시대의 유물) (몰라도됨)

참고) XML ?? JSON ??

XML 이란? (몰라도 됨, 그냥 참고, JSON은 알아야 됩니다.)

- Extensible Markup Language의 약자로 W3C에서 정의한 마크업 언 어.
- 이는 원래 Ajax에서 사용하는 데이터 형식을 뜻하는 단어.
- 하지만 현재 XML을 사용하는 경우는 매우 드물고, 주로 JSON과 텍스트 데이터를 사용.

XML vs JSON (JavaScript Object Notation)

- XML 과 JSON은 Markup language로, 태그 등을 이용하여 문서나 데이터의 구조를 명기하는 언어

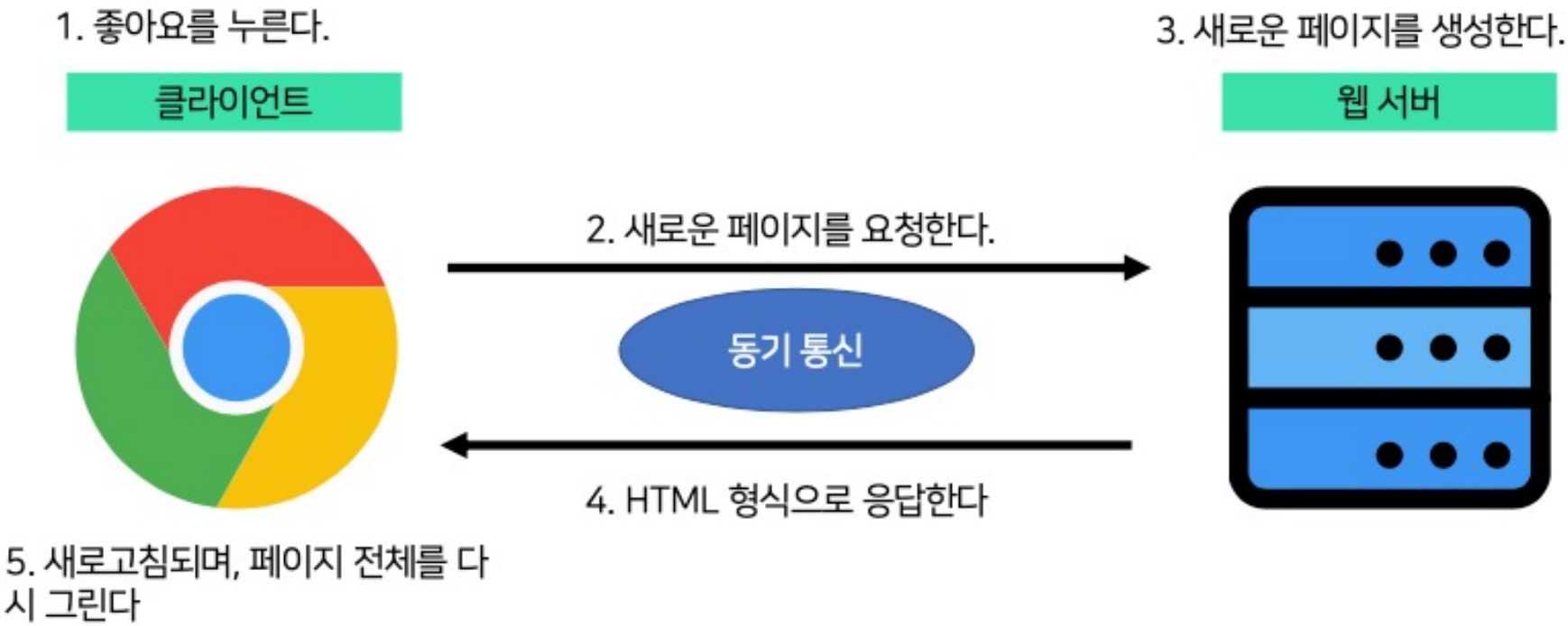
XML

```
<person>
  <age>25</age>
  <name>junghoon</name>
  <job>developer</job>
  <belongs-to>mutsa</belongs-to>
</person>
```

JSON

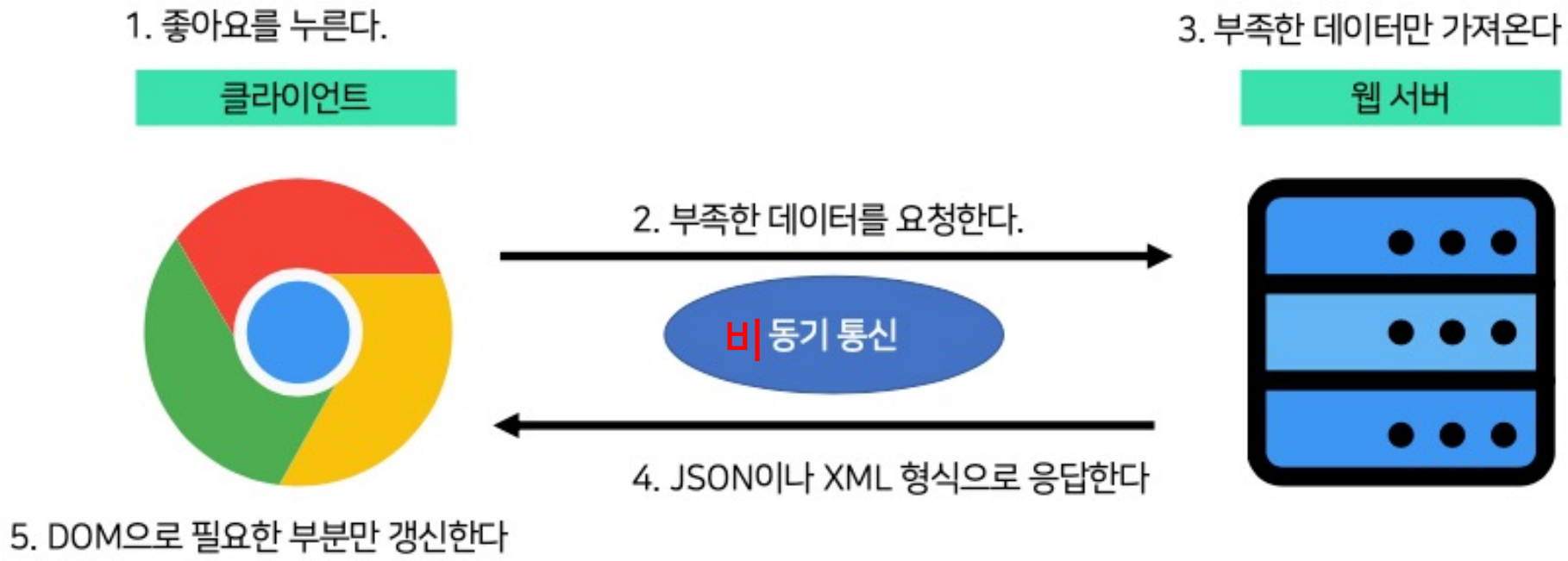
```
{
  'person': {
    'age': 25,
    'name': 'junghoon',
    'job': 'developer',
    'belongs-to': 'mutsa'
  }
}
```

동기(synchronous) 통신



현재 여러분들이 하는 장고 프로젝트, 고파스 등등이 이렇게 SSR(Server side rendering)로 구성되어있습니다.

비동기(Asynchronous) 통신



인스타에서 하트를 누르면 페이지 전환없이 하트가 증가됨을 알 수 있죠

비동기 통신의 장점

- 최소한의 데이터 통신만 하므로 처리 속도가 빠르고 서버 부하와 통신 트래픽 부하가 적음
- 비동기로 통신하므로 클라이언트 측에서 다른 작업을 할 수 있음
- 웹 페이지 갱신을 클라이언트 측이 담당 (Client side rendering)
- 페이지를 전환하는 대신에 페이지 일부분만 변경하므로 빠른 렌더링이 가능

좋아요를 구현해봅시다.

- git clone https://github.com/2-one-week/session13-Django-like-async.git

- cd djangolike
- pipenv shell
- pipenv install Django
- cd djangolike
- code .

- python manage.py makemigrations
- python manage.py migrate
- python manage.py createsuperuser // 알아서 만들어줍니다.
- python manage.py runserver

Like Model 추가

```
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
- python manage.py migrate

Admin.py에 Model 등록

```
...
from django.contrib import admin
from .models import Post, Comment, Like

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

동기 통신을 통해 먼저 구현해봅시다.

Detail.html에 like전용 form 추가해주기

```
  {% if user.is_authenticated %}
  <form method="POST">
    {% csrf_token %}
    <input type="text" name="content" placeholder="댓글을 입력하세요">
    <button type="submit">댓글 쓰기</button>
  </form>

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

Urls.py

```
urlpatterns = [
    # auth
    path('registration/signup', views.signup, name="signup"),
    path('registration/login', views.login, name="login"),
    path('registration/logout', views.logout, name="logout"),
    # social login
    path('accounts/', include('allauth.urls')),

    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"),
    # like
    path('like', views.like, name="like")
]
```

Views.py에 Like를 올려주는 view 작성

```
from django.shortcuts import render, redirect
from .models import Post, Comment, Like
from django.contrib.auth.models import User
from django.contrib import auth
from django.contrib.auth.decorators import login_required
```

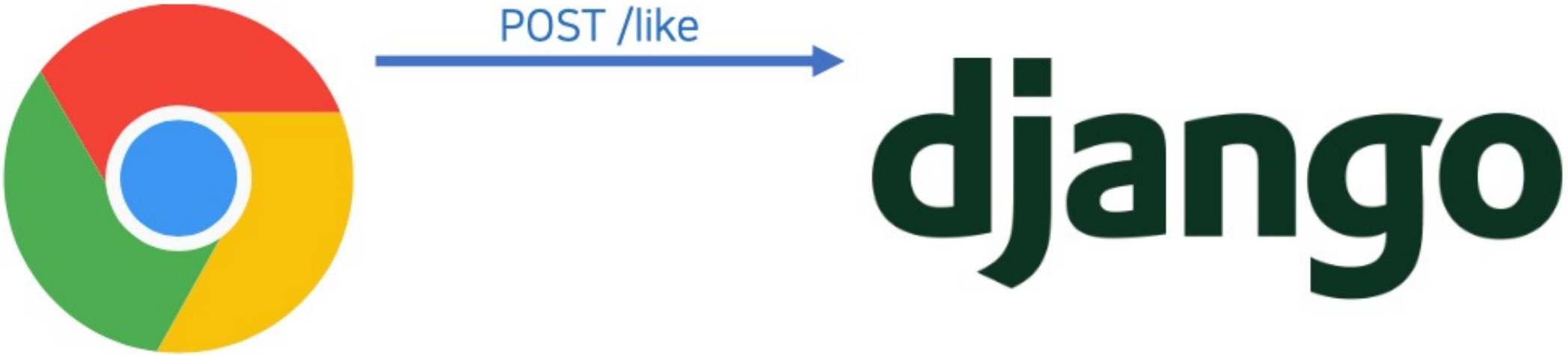
먼저 import

실습 1. 기존의 글쓰기와 동일하게 views.py를 작성해봅시다

제한 조건.
1. 좋아요를 누르면 해당 글에 좋아요를 증가시켜줍니다.
2. 이미 좋아요를 누른 글이라면, 좋아요를 취소합니다.
3. 좋아요를 눌렀다면, 해당 글의 detail page로 redirect 시켜줍니다.

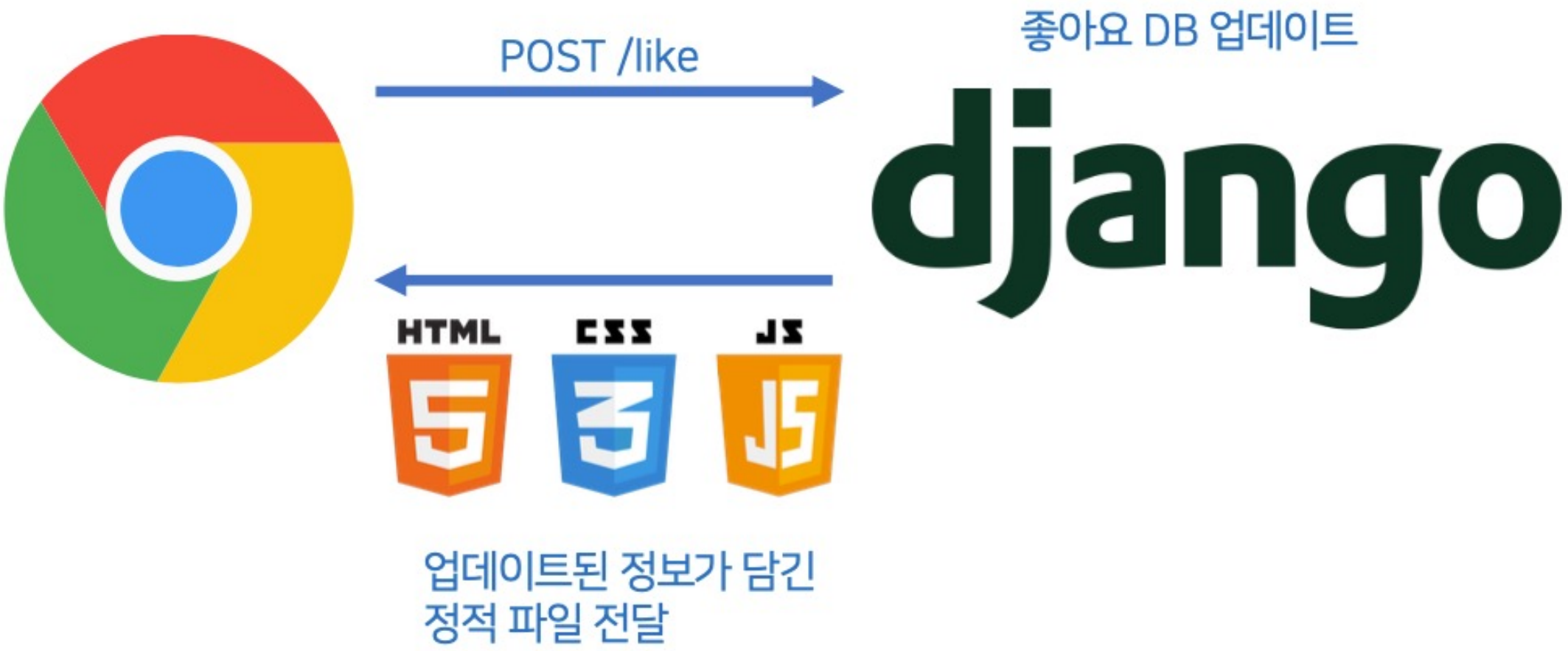
지금 작성한 view가 어떻게 작동하는지 봅시다.

1. 사용자가 장고에게 Like post 요청

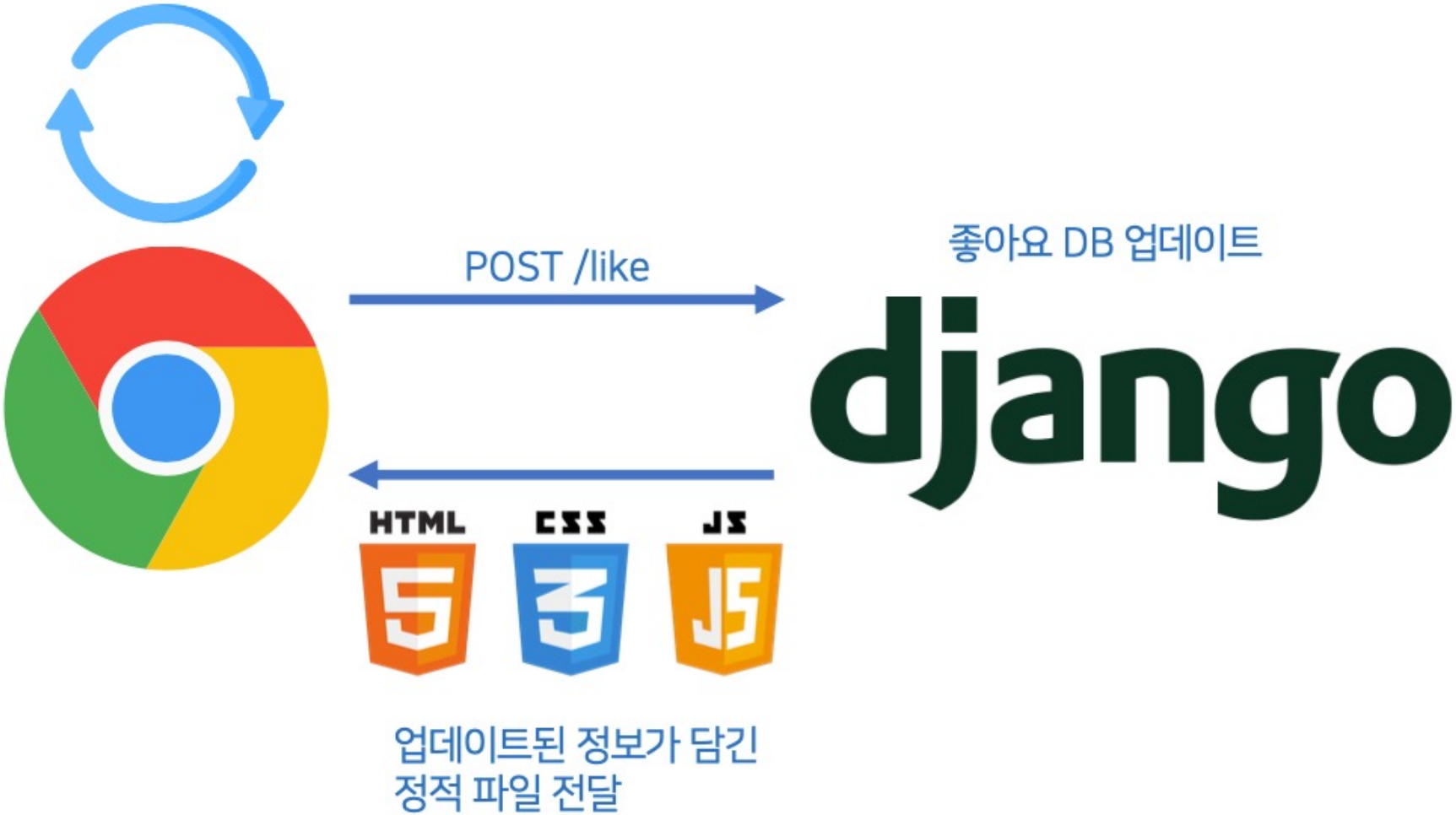


2. 장고에서 조건에 따라 Like db update

3. 장고에서 redirect detail -> render html



4. 브라우저에서 받은 파일을 통해 전체 재랜더링



그렇다면 동기 통신의 단점은 무엇인가?

- 서버가 데이터를 보내기 전까지 클라이언트는 아무 것도 할 수 없는 상태가 된다.
- 웹 페이지 전체를 주고받아야 하기 때문에 그만큼 시간이 걸린다.

이러한 단점을 해결하기 위해 비동기 통신이 나왔습니다.

비동기 통신으로 좋아요를 구현해봅시다.

그전에! 알아야될 개념들

- Django와 브라우저에서 JSON 다루기

- 저번 시간에 단비님이 했던 DOM control

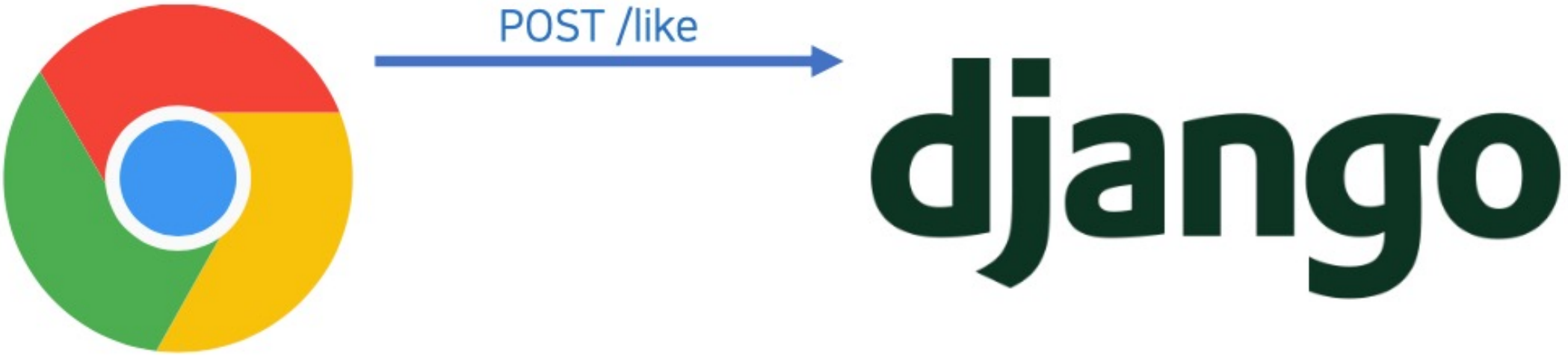
알아야될 개념 JSON

- 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

비동기는 이렇게 구현될 겁니다.

1. 사용자가 장고에게 Like post 요청

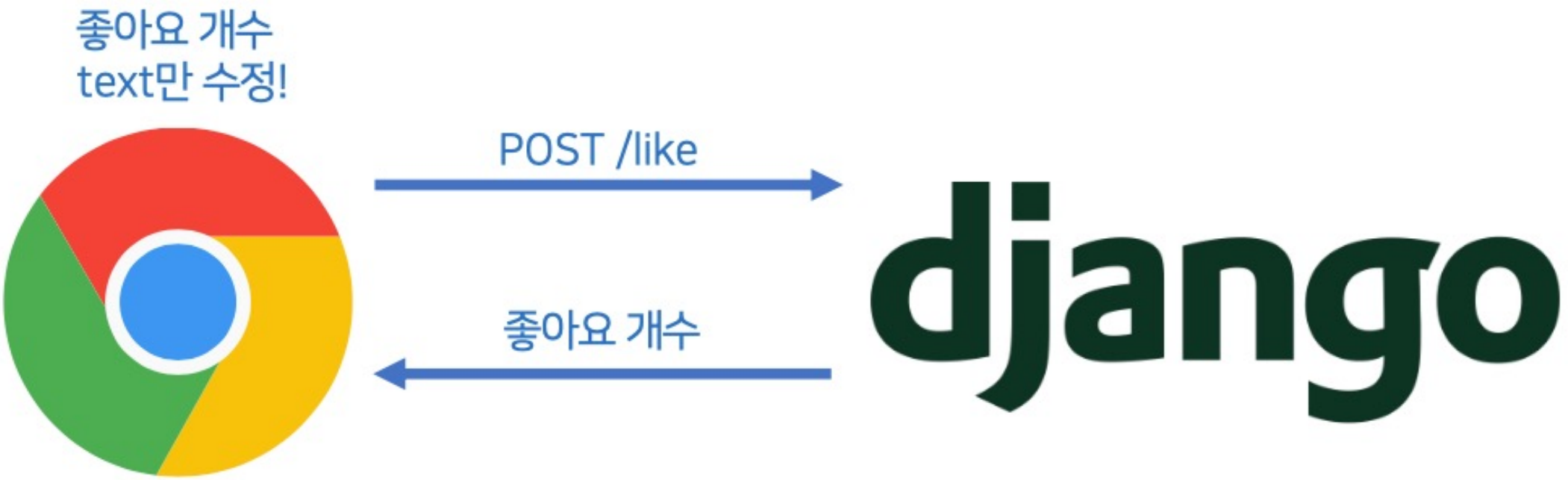


2. 장고에서 조건에 따라 Like db update

여기까지는 같죠

3. 장고에서 좋아요 개수를 담은 Response

4. 브라우저에서 받은 Response로 좋아요 개수만 수정



그래서 어떻게 하는데?

Detail.html 에 좋아요 버튼 추가 (Form아님)

```
{% if user.is_authenticated %}
<form method="POST">
  {% csrf_token %}
  <input type="text" name="content" placeholder="댓글을 입력하세요">
  <button type="submit">댓글 쓰기</button>
</form>

<button id="like-button">좋아요</button>
{% endif %}
<div id="like-count">좋아요{{ post.likes.count }}개</div>
```

그전에 onClick 이벤트 추가하는 방법 3가지를 알아보자

onclick 이벤트 추가 방법 1

- 버튼에 직접 attribute 추가하기

```
  {% if user.is_authenticated %}
  <form method="POST">
    {% csrf_token %}
    <input type="text" name="content" placeholder="댓글을 입력하세요">
    <button type="submit">댓글 쓰기</button>
  </form>
  <button id="like-button" onclick="alert('좋아요 버튼이 눌렸습니다')">좋아요</button>
  {% endif %}
  <div>좋아요 {{ post.likes.count }}개</div>
</div>


{% endblock content %}
```

onclick 이벤트 추가 방법 2

- script을 이용하여, DOM으로 onclick event를 지정해줄 element를 지정하고, onclick시 실행할 함수를 직접 선언한다.

```
  {% if user.is_authenticated %}
  <form method="POST">
    {% csrf_token %}
    <input type="text" name="content" placeholder="댓글을 입력하세요">
    <button type="submit">댓글 쓰기</button>
  </form>
  <button id="like-button">좋아요</button>
  {% endif %}
  <div>좋아요 {{ post.likes.count }}개</div>
</div>

<script>
  const likeButton = document.getElementById('like-button')

  likeButton.onclick = function () {
    alert('좋아요 버튼이 눌렸습니다')
  }
</script>
```

onclick 이벤트 추가 방법 3

- script에서 함수를 선언하고, HTML단에서 선언한 함수를 가져와 쓴다.

```
  {% if user.is_authenticated %}
  <form method="POST">
    {% csrf_token %}
    <input type="text" name="content" placeholder="댓글을 입력하세요">
    <button type="submit">댓글 쓰기</button>
  </form>
  <button id="like-button" onclick="alertLikeButtonClicked()">좋아요</button>
  {% endif %}
  <div>좋아요 {{ post.likes.count }}개</div>
</div>
<script>
  function alertLikeButtonClicked() {
    alert('좋아요 버튼이 눌렸습니다')
  }
</script>
{% endblock content %}
```

Fetch API

- Using Fetch (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)
- Ex)

```
const data = { username: 'example' };

fetch('https://example.com/profile', {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

Fetch API 적용하기

```
  {% if user.is_authenticated %}
  <form method="POST">
    {% csrf_token %}
    <input type="text" name="content" placeholder="댓글을 입력하세요">
    <button type="submit">댓글 쓰기</button>
  </form>

  <button id="like-button" onclick="like()">좋아요</button>
  {% endif %}
  <div id="like-count">좋아요{{ post.likes.count }}개</div>

</div>

<script>
  const like = () => {
    fetch('/like', {
      method: "POST",
      body: JSON.stringify({ post_pk: "{{ post.pk }}" })
    })
      .then(response => response.json())
      .then(res => document.getElementById('like-count').innerHTML = '좋아요' + res.like_count + '개')
      .catch(error => console.error(err))
  }
</script>
```

Django에서 요청 받기

Views.py 필요한 거 import

```python
from django.shortcuts import render, redirect
from .models import Post, Comment, Like
from django.contrib.auth.models import User
from django.contrib import auth
from django.contrib.auth.decorators import login_required
from django.views.decorators.csrf import csrf_exempt
from django.http import HttpResponse
import json
```

```python
@csrf_exempt
def like(request):
    if request.method == "POST":
        post_pk = request.POST['post_pk']
```

Views.py request body 받기

```python
@csrf_exempt
def like(request):
    if request.method == "POST":
        request_body = json.loads(request.body)
        post_pk = request_body['post_pk']

        existing_like = Like.objects.filter(
            post = Post.objects.get(pk=post_pk),
            user = request.user
        )

        # 좋아요 취소
        if existing_like.count() > 0:
            existing_like.delete()

        # 좋아요 생성
        else:
            Like.objects.create(
                post = Post.objects.get(pk=post_pk),
                user = request.user
            )
```

Views.py response 보내기

```python
# 좋아요 취소
if existing_like.count() > 0:
    existing_like.delete()

# 좋아요 생성
else:
    Like.objects.create(
        post = Post.objects.get(pk=post_pk),
        user = request.user
    )

post_likes = Like.objects.filter(
    post = Post.objects.get(pk=post_pk)
)

response = {
    'like_count': post_likes.count()
}

return HttpResponse(json.dumps(response))
```

HttpResponse를 왜 사용해야하는가?

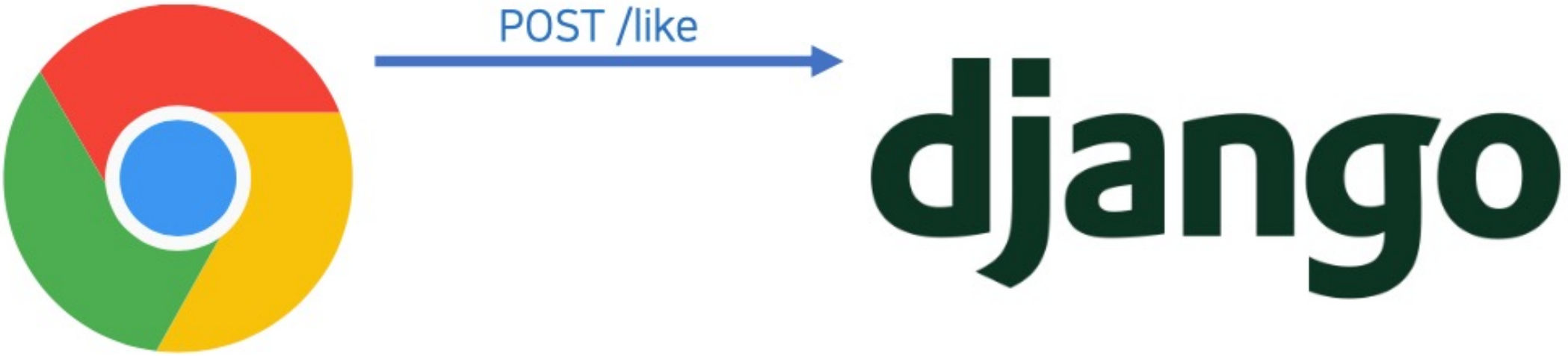
```
response = {
    'like_count': post_likes.count()
}

return (json.dumps(response))

# 이런식으로 통신하는 것은 불가능합니다. 왜?
# 요청에 대한 응답 메시지는 HTTP response 형식에 맞아야 합니다.
# = 응답 행, 응답 헤더, 메시지 본문이 모두 포함된 정보를 주어야 합니다.
# json.dumps(response) 자체는 응답 상태 코드(200, 400, ...), 응답 헤더(Content-type, etc..)를 포함하지 않습니다.
# 그래서 위 정보들을 자동으로 반환해주는 HttpResponse라는 모듈을 사용해야 합니다.
```

그래서 이제는?

1. 사용자가 장고에게 Like post 요청



2. 장고에서 조건에 따라 Like db update

여기까지는 같죠

3. 장고에서 좋아요 개수를 담은 Response

4. 브라우저에서 받은 Response로 좋아요 개수만 수정

Fetch 대신 Axios??

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

- Axios의 장점

1. JSON data를 자동으로 변환해줌
2. Node js에서도 사용이 가능
3. 문법이 간결함

Axios 사용하기

스크립트 추가하기

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

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const like = () => {
    axios('/like', {
      method: "POST",
      data: { post_pk: "{{ post.pk }}" }
    })
      .then(response => document.getElementById('like-count').innerHTML = '좋아요' + response.data.like_count + '개')
      .catch(error => console.error(error))
  }
</script>
{% endblock content %}
```

Fetch vs Axios

편의에 따라 사용합시다.

```
<script>
  const like = () => {
    fetch('/like', {
      method: "POST",
      body: JSON.stringify({ post_pk: "{{ post.pk }}" })
    })
      .then(response => response.json())
      .then(res => document.getElementById('like-count').innerHTML = '좋아요' + res.like_count + '개')
      .catch(error => console.error(err))
  }
```

```
<script>
  const like = () => {
    axios('/like', {
      method: "POST",
      data: { post_pk: "{{ post.pk }}" }
    })
      .then(response => document.getElementById('like-count').innerHTML = '좋아요' + response.data.like_count + '개')
      .catch(error => console.error(error))
  }
```

Axios 더 간결하게 쓰기

axios.post(url, data, headers)

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const like = () => {
    axios.post('/like', { post_pk: "{{ post.pk }}" })
      .then(response => document.getElementById('like-count').innerHTML = '좋아요' + response.data.like_count + '개')
      .catch(error => console.error(error))
  }
</script>
```

심화) Promise.then 대신 async/await

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const like = async () => {
    try {
      const response = await axios.post('/like', { post_pk: "{{ post.pk }}" })
      document.getElementById('like-count').innerHTML = '좋아요' + response.data.like_count + '개'
    } catch (error) {
      console.error(error)
    }
  }
</script>
```

심화) ES6 비구조화 할당 1

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const like = async () => {
    try {
      const { data } = await axios.post('/like', { post_pk: "{{ post.pk }}" })
      document.getElementById('like-count').innerHTML = '좋아요' + data.like_count + '개'
    } catch (error) {
      console.error(error)
    }
  }
</script>
```

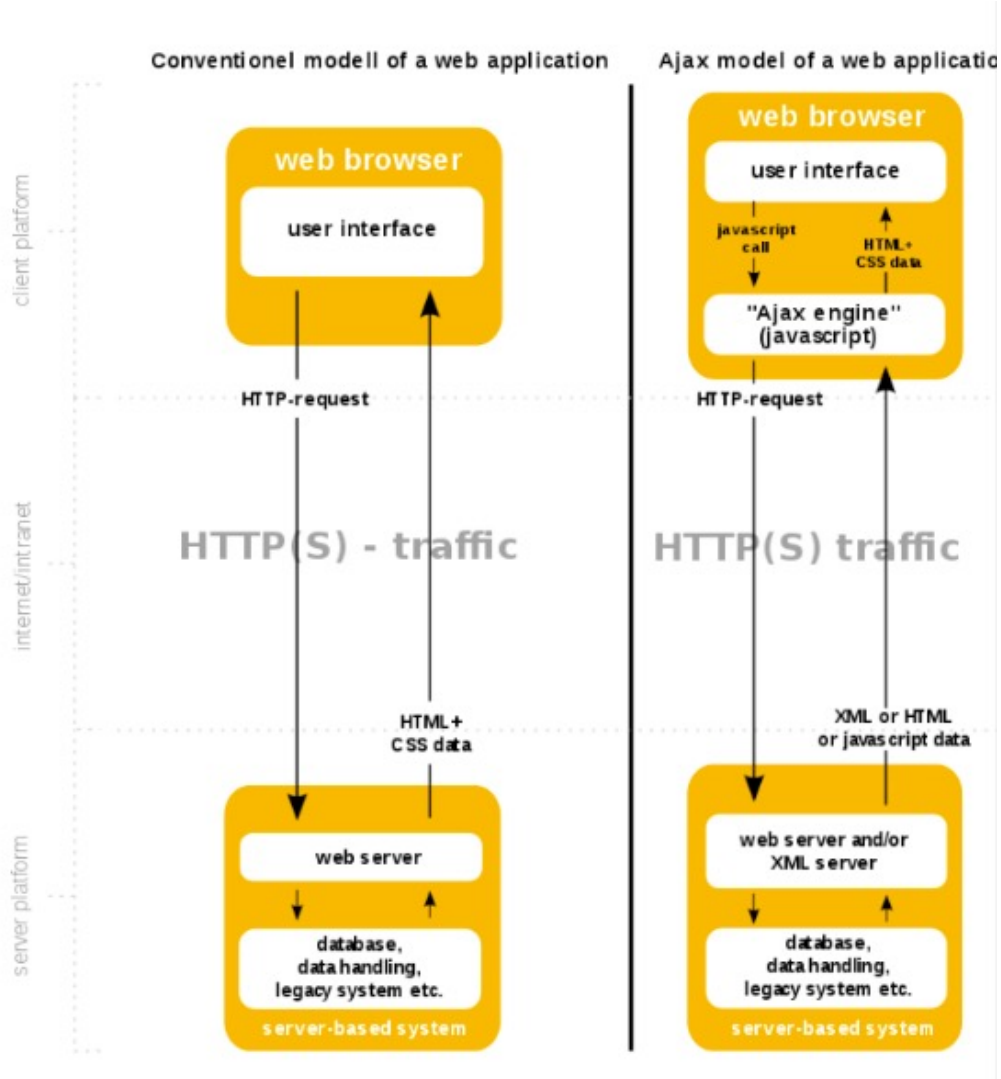
심화) ES6 비구조화 할당 2

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const like = async () => {
    try {
      const { data: { like_count } } = await axios.post('/like', { post_pk: "{{ post.pk }}" })
      document.getElementById('like-count').innerHTML = '좋아요' + like_count + '개'
    } catch (error) {
      console.error(error)
    }
  }
</script>
{% endblock content %}
```

총 정리

- 동기 통신
  1) HTML 정적 파일 수신
  2) 매번 새로 고침

- 비동기 통신
  1) JSON 송수신
  2) 필요한 부분만 DOM 수정



그럼 나도 오늘부터 비동기 마스터?

ㄴㄴ

- 비동기의 개념은 이보다 심오하고 깊습니다
- 콜백지옥
- Promise의 개념

에 대해 검색해보세요.

그래도 이제 에타 정도는 만들 수 있게 되었습니다.

과제

1. 게시글에 접속했을 때, 유저가 좋아요를 누른 게시글이면 좋아요가
   빨간색으로 표시되게 해주세요. 최초 접속 시에도 가능해야 합니다.
2. 찜하기 기능을 만들어주세요. 찜한 목록을 마이페이지에서 따로 확인할 수 있게 해주세요.
3. 마이페이지를 만들어서, 좋아요를 누른 게시글 제목만 모아볼 수 있게 만들어 주세요.
4. 마이페이지를 만들어서, 찜하기를 누른 게시글 제목만 모아볼 수 있게 만들어 주세요.

Hint) style 이용하면 됩니다.
response 처리 연습입니다.