좋아요로 알아보는 비동기

권중훈

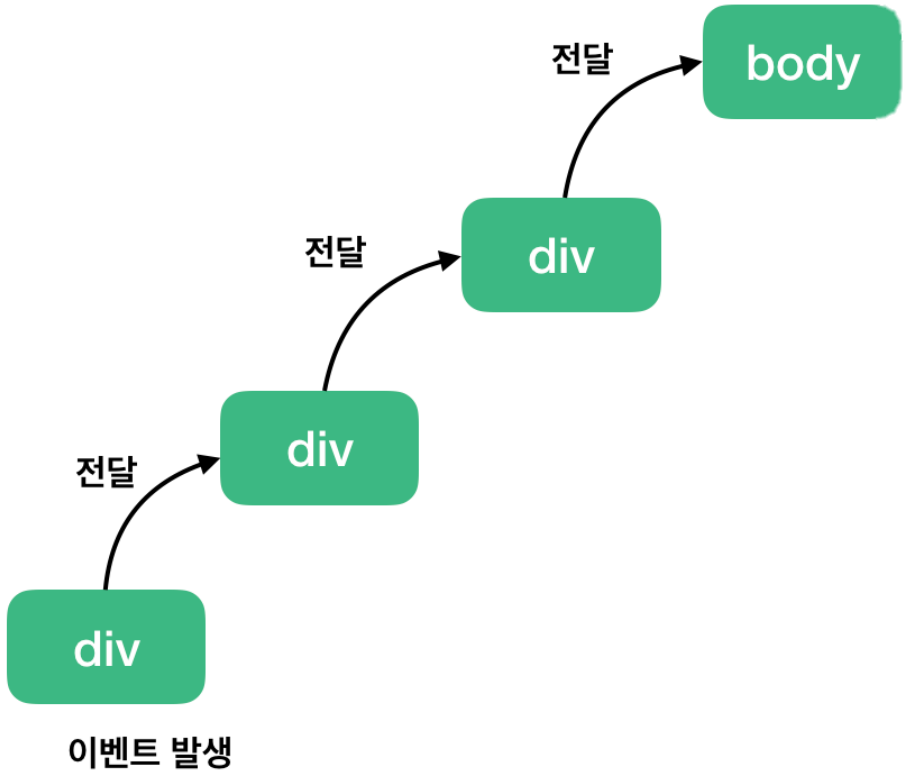
Assignment.1

```
const result = document.querySelector('#result')
const input = document.querySelector('#input')
const withSpace = document.querySelector('#countWithSpace')
const withoutSpace = document.querySelector('#countWithoutSpace')


function onKeyUp(){
  let inputValue = this.value;
  let characterNumWithSpace = inputValue.length
  let characterNumwithoutSpace = inputValue.replace(/\s/g,"").length

  withSpace.textContent = `공백 포함 글자 수 : ${ characterNumWithSpace }`
  withoutSpace.textContent = `공백 제외 글자 수 :  ${ characterNumwithoutSpace }`
}

input.addEventListener('keyup',onKeyUp)
```

정규표현식 공부합시다!

Assignment.2

Event
Bubbling



버블링

버블링(bubbling)의 원리는 간단합니다.

한 요소에 이벤트가 발생하면, 이 요소에 할당된 핸들러가 동작하고, 이어서 부모 요소의 핸들러가 동작합니다. 가장 최상단의 조상 요소를 만날 때까지 이 과정이 반복되면서 요소 각각에 할당된 핸들러가 동작합니다.

어떻게 될까요?

```
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```

이렇게 됩니다.

가장 안쪽의 <p>를 클릭하면 순서대로 다음과 같은 일이 벌어집니다.

1. <p>에 할당된 onclick 핸들러가 동작합니다.
2. 바깥의 에 할당된 핸들러가 동작합니다.
3. 그 바깥의 <form>에 할당된 핸들러가 동작합니다.
4. document 객체를 만날 때까지, 각 요소에 할당된 onclick 핸들러가 동작합니다.

Assignment. 2

```
// CONSTANT VARIABLE
const GOLD = "#fed330"
const BLUE = "#4b7bec"

// PUBLIC METHODS
const resetColor = () => {
  divs.forEach(div => div.style.backgroundColor='white')
}

const changeParentColor = (target,color) => {
  const parent = target.parentNode

  if(parent.localName !== "body"){
    parent.style.backgroundColor = color
  }
}
```

Assignment. 2

```
// version1 : stopPropagation - 모든 요소에 이벤트 할당하고 버블링 막자!
const divs = document.querySelectorAll('div')

function onClick(event){
  // 버블링 방지
  event.stopPropagation()
  // 모든 div 색상 초기화
  resetColor()
  // 자신의 색상 변경
  this.style.backgroundColor = BLUE
  // 부모 색상 변경
  changeParentColor(this, GOLD)
}
// 모든 요소에 onClick 이벤트 걸어준다
divs.forEach(div => div.addEventListener('click',onClick))
```

Event.stopPropagation( )

Event Delegation

이벤트 위임

캡처링과 버블링을 활용하면 강력한 이벤트 핸들링 패턴인 *이벤트 위임(event delegation)* 을 구현할 수 있습니다.

이벤트 위임은 비슷한 방식으로 여러 요소를 다뤄야 할 때 사용됩니다. 이벤트 위임을 사용하면 요소마다 핸들러를 할당하지 않고, 요소의 공통 조상에 이벤트 핸들러를 단 하나만 할당해도 여러 요소를 한꺼번에 다룰 수 있습니다.

공통 조상에 할당한 핸들러에서 `event.target`을 이용하면 실제 어디서 이벤트가 발생했는지 알 수 있습니다. 이를 이용해 이벤트를 핸들링합니다.

This vs Event.target

event.target

부모 요소의 핸들러는 이벤트가 정확히 어디서 발생했는지 등에 대한 자세한 정보를 얻을 수 있습니다.

이벤트가 발생한 가장 안쪽의 요소는 *타깃(target)* 요소라고 불리고, `event.target`을 사용해 접근할 수 있습니다.

`event.target`과 `this`(=`event.currentTarget`)는 다음과 같은 차이점이 있습니다.

- `event.target`은 실제 이벤트가 시작된 '타깃' 요소입니다. 버블링이 진행되어도 변하지 않습니다.
- `this`는 '현재' 요소로, 현재 실행 중인 핸들러가 할당된 요소를 참조합니다.

Assignment. 2

```
// version2 : delegation - 제일 상위 요소에 이벤트 할당하고 위임해주자!
// document 이하 제일 첫번째 div를 잡습니다
const div = document.querySelector('div')

const onClick = (event) => {
  // 이벤트 발생시킨 요소 잡고!
  const target = event.target
  // 모든 div 색상 초기화
  resetColor()
  // 자신의 색상변경
  target.style.backgroundColor = BLUE
  // 부모 색상 변경
  changeParentColor(target, GOLD)
}
// 최상위 부모 요소에만 이벤트 걸어준다
div.addEventListener("click",onClick)
```

오늘 할 것

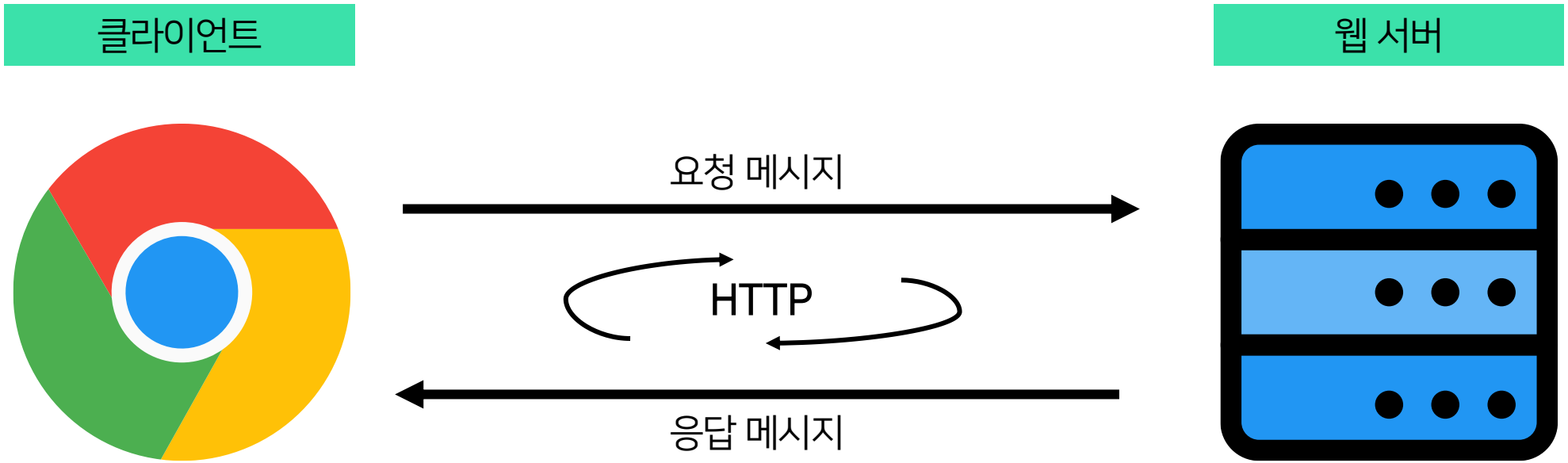
- 비동기, HTTP
- fetch, axios
- json

HTTP란?

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

HTTP 통신이란?

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



심화)

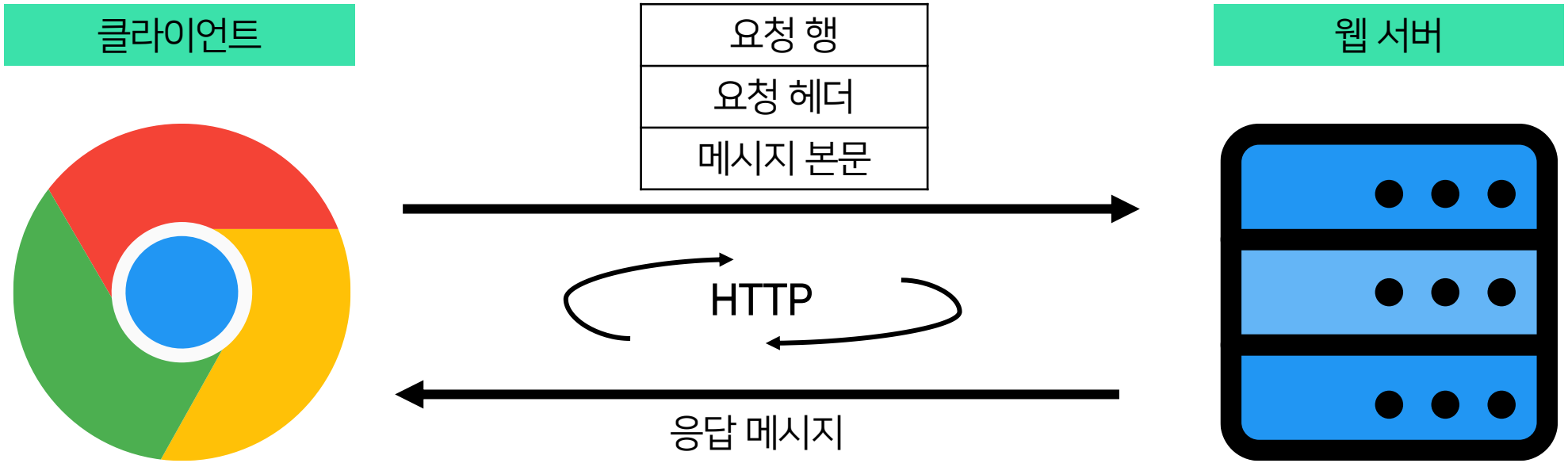
- HTTP에서는 전송 계층 프로토콜로 TCP를 사용하고 네트워크 계층 프로토콜로 IP를 사용하는 것이 일반적입니다. 이 두 계층을 합쳐서 TCP/IP라는 이름으로 부릅니다.

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

- HTTP에서는 기본적으로 80번 포트를 사용합니다.

HTTP 요청 메시지

- HTTP 요청 메시지는 요청 행, 요청 헤더, 메시지 본문 이라는 세 부분으로 구성됩니다.



HTTP 요청 메시지 – 요청 행

- 다음은 요청 행의 구체적인 예입니다.

- GET http://www.naver.com HTTP/1.1

- GET: 요청 메서드
- http://www.naver.com: URL
- HTTP/1.1: HTTP의 버전

HTTP 요청 메시지 – 요청 헤더

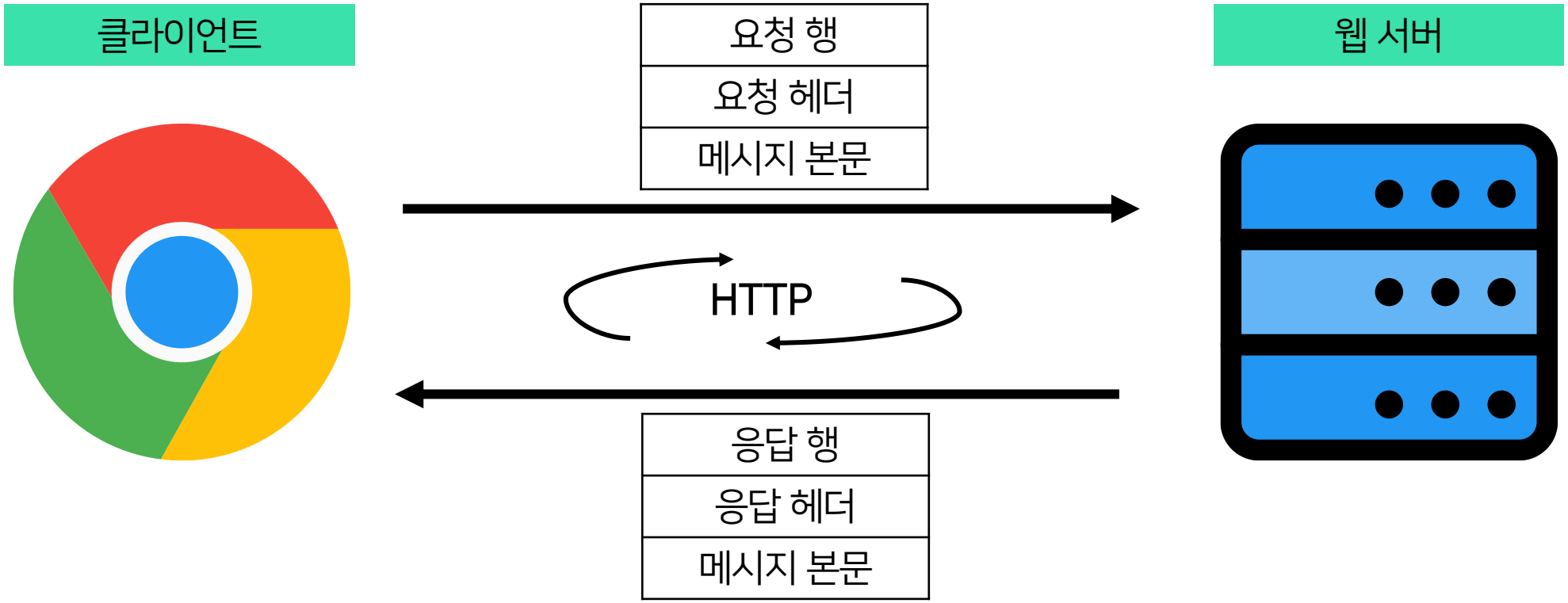
- 요청 헤더에는 메시지의 제어 정보, 메시지 본문에 저장한 데이터 정보(데이터 종류나 문자 코드) 등이 텍스트 형식으로 저장됩니다.

HTTP 요청 메시지 - 메시지 본문

- 메시지 본문에는 보내는 데이터가 저장됩니다. 데이터 형식은 요청 헤더에 지정된 타입을 따릅니다.

HTTP 응답 메시지

- HTTP 응답 메시지는 응답 행, 응답 헤더, 메시지 본문 이라는 세 부분으로 구성됩니다.



HTTP 응답 메시지 – 응답 행

- 다음은 응답 행의 구체적인 예입니다.

- HTTP/1.1 200 OK

- HTTP/1.1: HTTP의 버전
- 200: 상태 코드
- OK: 보충 메시지

HTTP의 주요 상태 코드와 상태 설명

- 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

HTTP 응답 메시지 – 응답 헤더

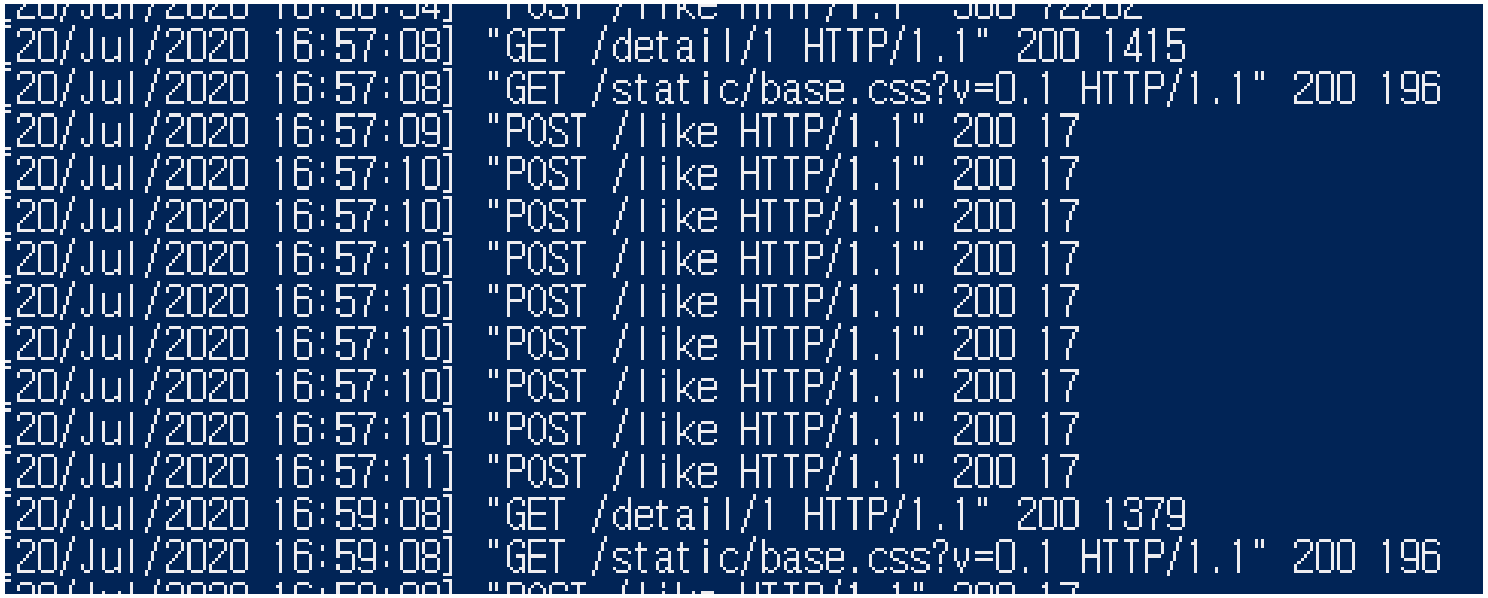
- 응답 헤더에는 메시지의 제어 정보, 메시지 본문에 저장한 데이터 정보(데이터 종류나 문자 코드) 등이 텍스트 형식으로 저장됩니다.

HTTP 응답 메시지 - 메시지 본문

- 메시지 본문에는 받을 데이터가 담기며, 그 데이터의 형식은 요청 헤더에 지정된 타입을 따릅니다. GET 메서드 요청에 대한 응답 메시지는 HTML 파일, CSS 파일, 자바스크립트 파일, 이미지 파일처럼 웹 페이지를 정의하는 데이터입니다.

참고) Django의 서버 로그

- 특정 URL로, 어떠한 HTTP method 요청을 보내서 이렇게 처리되었다. 라는 뜻입니다. 해당 내용이 로그에 남게 됩니다.



Ajax

- Ajax(Asynchronous JavaScript + XML)란 XMLHttpRequest 라는 자바스크립트의 객체를 이용하여 웹 서버와 비동기로 통신하고 DOM을 이용하여 웹 페이지를 동적으로 갱신하는 프로그래밍 기법을 말합니다.

- 참고로, Asynchronous란 '비동기'란 뜻입니다.

- 편의상, XMLHttpRequest 객체를 직접 만들지 않고 fetch API와 axios 라이브러리를 사용해 데이터를 송수신합니다.
- jQuery ajax는 다루지 않습니다.

참고) XML

- Extensible Markup Language의 약자로 W3C에서 정의한 마크업 언어입니다.
- 이는 원래 Ajax에서 사용하는 데이터 형식을 뜻하는 단어였습니다.
- 하지만 현재 XML을 사용하는 경우는 매우 드물고, 주로 JSON과 텍스트 데이터를 사용합니다.

참고) XML vs JSON

- Markup language – 태그 등을 이용하여 문서나 데이터의 구조를 명기하는 언어

- JSON = JavaScript Object Notation

```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'
  }
}
```

Ajax의 특징

- 동기 통신과 비동기 통신의 차이점을 통해 알아봅시다.

동기 통신

1. 좋아요를 누른다.

클라이언트

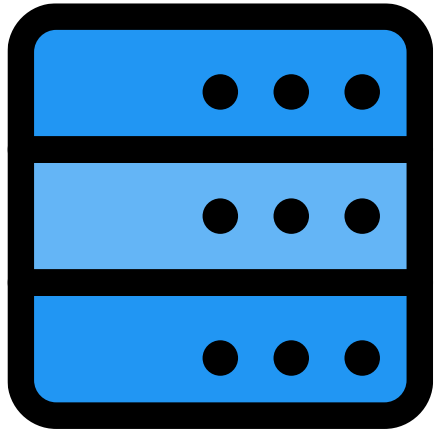
2. 새로운 페이지를 요청한다.

동기 통신

4. HTML 형식으로 응답한다

3. 새로운 페이지를 생성한다.

웹 서버



5. 새로고침되며, 페이지 전체를 다시 그린다

비동기 통신

1. 좋아요를 누른다.

클라이언트

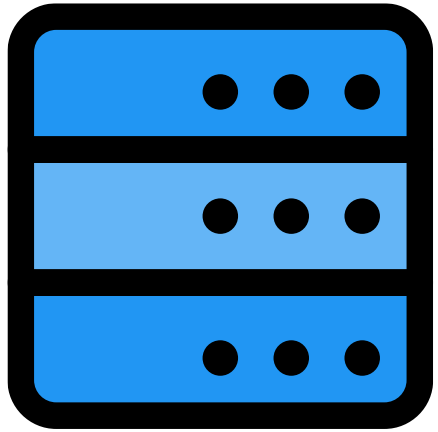
2. 부족한 데이터를 요청한다.

동기 통신

4. JSON이나 XML 형식으로 응답한다

3. 부족한 데이터만 가져온다

웹 서버



5. DOM으로 필요한 부분만 갱신한다

Ajax의 장점

- 최소한의 데이터 통신만 하므로 처리 속도가 빠르고 서버 부하와 통신 트래픽 부하가 적다.

- 비동기로 통신하므로 클라이언트 측에서 다른 작업을 할 수 있다.

- 웹 페이지 갱신을 클라이언트 측이 담당한다. 페이지를 전환하는 대신에 페이지 일부분만 변경하므로 고속 렌더링이 가능하다.

좋아요 구현 - 동기 통신

- 기존에 배운 form-data CRUD를 이용해 좋아요를 구현해봅시다.

Prerequisites

- git clone https://github.com/Gjunghoon/django-like-async.git
- cd django-like-async
- pipenv shell
- pipenv install
- cd likeproject
- code .
- python manage.py makemigrations
- python manage.py migrate
- python manage.py createsuperuser
- python manage.py runserver

models.py

```
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.py

```python
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)
```

detail.html

```
  {% 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.py

```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
```

```python
def like(request):
    if request.method == "POST":
        post_pk = request.POST['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
            )
        return redirect('detail', post_pk)
```


django

POST /like

POST /like

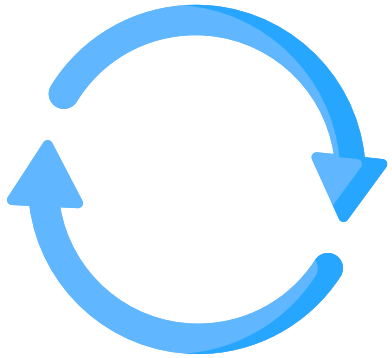
좋아요 DB 업데이트

django

POST /like

좋아요 DB 업데이트

업데이트된 정보가 담긴
정적 파일 전달



POST /like

좋아요 DB 업데이트

업데이트된 정보가 담긴
정적 파일 전달

동기 통신의 단점

- 서버가 데이터를 보내기 전까지 클라이언트는 아무 것도 할 수 없는 상태가 된다.

- 웹 페이지 전체를 주고받아야 하기 때문에 그만큼 시간이 걸린다.

좋아요 구현 - 비동기 통신

- fetch API, axios 라이브러리를 사용해 필요한 데이터만 JSON 형식으로 가져와서, 필요한 부분만 DOM control으로 수정해봅시다.

비동기 통신을 하기 전에 알아야할 개념

- 1. JSON 다루기
- 1) JSON.stringify
- 2) JSON.parse
- 3) json.loads()
- 4) json.dumps()

- 2. DOM control

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

POST /like

POST /like

좋아요 DB 업데이트

django

POST /like

좋아요 개수

django

좋아요 개수
text만 수정!

POST /like

좋아요 개수

detail.html

```
{% 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>
```

onclick 이벤트 추가 방법1

- onclick attribute 추가 후 value로 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" onclick="alert('좋아요 버튼이 눌렸습니다')">좋아요</button>
  {% endif %}
  <div>좋아요 {{ post.likes.count }}개</div>
</div>


{% endblock content %}
```

onclick 이벤트 추가 방법2

- 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 방법3

- 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

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

- fetch(url, {
- method: ‘HTTP method’. // GET, POST, PATCH, DELETE, ⋯
- headers: {
- ‘key’: ‘value of HTTP request headers’
- // Accept, Content-type, Access-Control-Allow-Origin, ⋯
- },
- body: ‘HTTP request message’ // ex. JSON.stringify(data)
- }

```
  {% 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>
  function like() {
    fetch('/like', {
      method: "POST",
      headers: { "X-CSRFToken": "{{ csrf_token }}" },
      body: JSON.stringify({ post_pk: "{{ post.pk }}" }),
    }).then(response => response.json().then(jsonRes => {
      document.getElementById('like-count').innerHTML = '좋아요 ' + jsonRes.like_count + '개'
    })).catch(err => console.error(err))
  }
</script>
{% endblock content %}
```

```
    {% 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>
  {% endblock content %}
```

views.py

```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

```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

```
# 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))
```

```
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.
```

POST /like

POST /like

좋아요 DB 업데이트

django

POST /like

좋아요 개수

django

좋아요 개수
text만 수정!

POST /like

좋아요 개수


```
{
  'like_count' : 업데이트 된 좋아요 개수
}
```

axios

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

- 장점
- JSON data를 자동으로 변환해준다.
- node.js에서도 사용이 가능하다.
- 문법이 간결하다.

- `<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 사용하세요.

```
<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))
  }
```

detail.html

```
<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 %}
```

더 간결하게

- 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) 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>
```

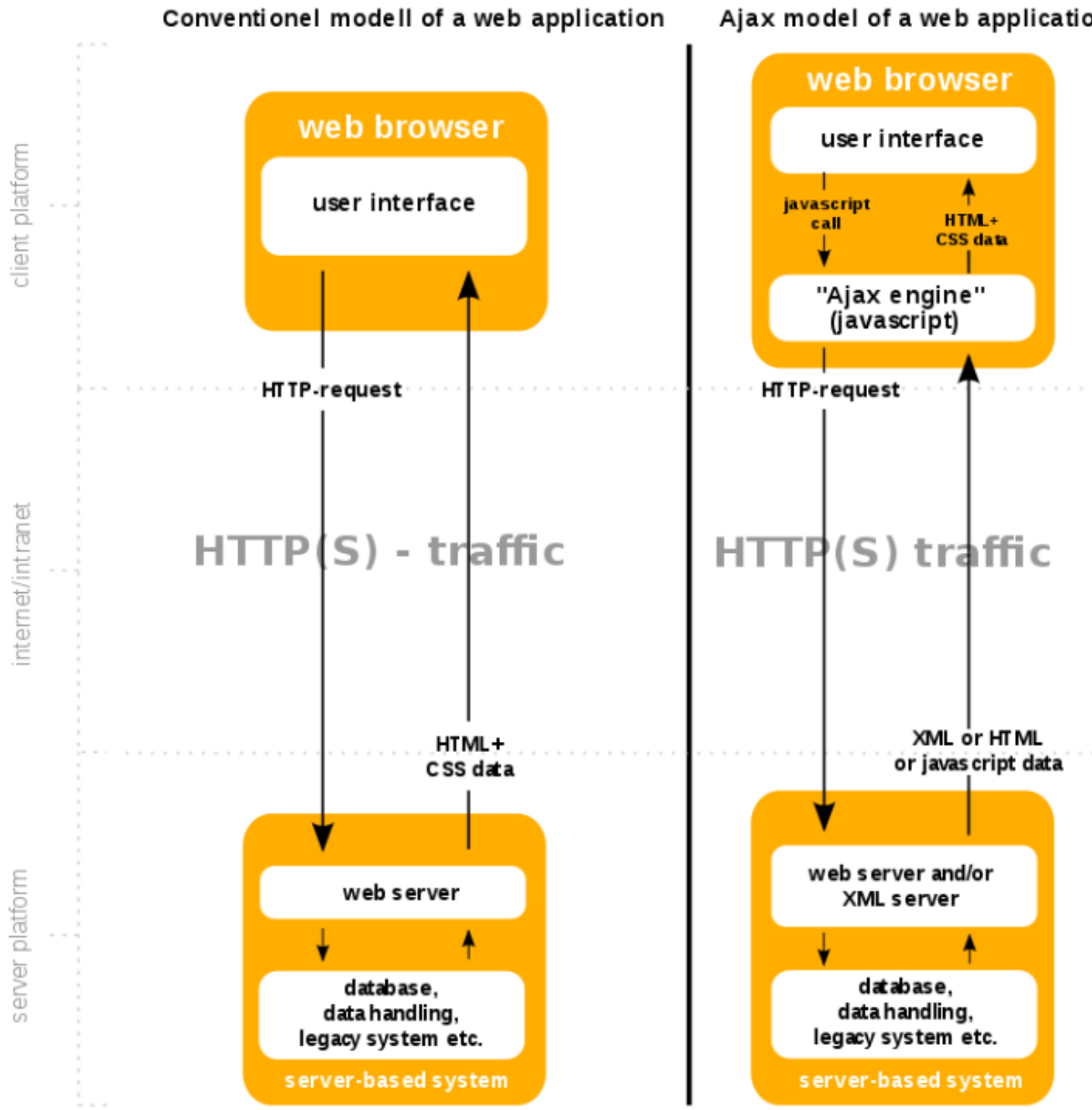
심화) ES6 비구조화 할당 2

```
<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 %}
```

총정리

- 동기 통신
- 1) HTML 정적 파일 수신
- 2) 새로고침

- 비동기통신
- 1) JSON 송수신
- 2) 필요한 부분만 DOM으로 수정



추가적으로..

- 비동기의 개념은 이보다 심오합니다.
- 콜백 지옥
- Promise의 개념
- 에 대해 검색해보세요.

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 만들어주세요.
- 4. 마이페이지를 만들어서, 찜하기를 누른 게시글 제목만 모아볼 수 있게 만들어주세요.

- Hint) style 이용하면 됩니다.
- response 처리 연습입니다.