Synchronous & Asynchronous

- 동기(Synchronous)
  - 모든 일을 순서대로 하나씩 처리하는 것
  - 순서대로 처리한다 == 이전 작업이 끝나면 다음 작업을 시작한다.
  - 요청과 응답을 동기식으로 처리한다면?
    - 요청을 보내고 응답이 올때까지 기다렸다가 다음 로직을 처리
- 비동기(Asynchronous)
  - 작업을 시작한 후 결과를 기다리지 않고 다음 작업을 처리하는 것 (병렬적 수행)
  - 시간이 필요한 작업들은 요청을 보낸 뒤 응답이 빨리 오는 작업부터 처리
  - 비동기를 사용하는 이유
    - 사용자 경험
      - 비동기로 처리한다면 먼저 처리되는 부분부터 보여줄 수 있으므로,
        사용자 경험에 긍정적인 효과를 볼 수 있다.
      - 동기식 처리는 특정 로직이 실행되는 동안 다른 로직 실행을 차단하기 때문에
        마치 프로그램이 응답하지 않는 듯한 사용자 경험을 만들게 됨

JavaScript의 비동기 처리

- Single Thread 언어, JavaScript
  - JavaScript는 한 번에 하나의 일만 수행할 수 있는 Single Thread 언어로 동시에 여러 작업을 처리할 수 없음

- JavaScript Runtime
  - JS 자체는 Single Thread이므로 비동기 처리를 할 수 있도록 도와주는 환경이 필요함
  - 특정 언어가 동작할 수 있는 환경을 런타임이라 함
  - JS에서 비동기와 관련한 작업은 브라우저 또는 Node 환경에서 처리
  - 이중에서 브라우저 환경에서의 비동기 동작은 크게 아래의 요소들로 구성된다
    - JS Engine의 Call Stack
    - Web API
    - Task Queue
    - Event Loop

- 비동기 처리 동작 방식
  - 브라우저 환경에서의 JS의 비동기는 아래와 같이 처리된다.
    1. 모든 작업은 Call Stack(LIFO)으로 들어간 후 처리
    2. 오래 걸리는 작업이 Call Stack으로 들어오면 Web API로 보내서 처리하도록 한다.
    3. Web API에서 처리가 끝난 작업들은 Task Queue(FIFO)에 순서대로 들어간다
    4. Event Loop가 Call Stack이 비어 있는 것을 체크하고,
       Task Queue에서 가장 오래된 작업을 Call Stack으로 보낸다.

- 정리

- JS는 한 번에 하나의 작업을 수행하는 Single Thread 언어로 동기적 처리를 하지만,
  브라우저 환경에서는 Web API에서 처리된 작업이 지속적으로 Task Queue를 거쳐
  Event Loop에 의해 Call Stack에 들어와 순차적으로 실행됨으로써 비동기 작업이 가능한 환경이 된다.

Axios

- Axios 라이브러리
  - JS의 HTTP 웹 통신을 위한 라이브러리
  - node 환경은 npm을 이용해서 설치 후 사용할 수 있고,
    browser 환경은 CDN을 이용해서 사용할 수 있음

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
    axios.get({'요청할 url '})
        .then(성공하면 수행할 콜백 함수)
        .catch(실패하면 수행할 콜백 함수)
</script>
```

  - Python 으로 요청해보기 (동기)

```python
import requests

dob_url = "https://api.thedogapi.com/v1/images/search"
response = requests.get(dog_url)

if response.status_code == 200:
    print(response.json())
else:
    print('실패')

# import requests 를 하기 위해선
# $ pip install requests 를 터미널에서 설치
```

  - Axios로 요청해보기 (비동기)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <p>
        <img src="" alt="dog">
    </p>
        <button>클릭</button>

    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script>
        const imgTag = document.querySelector('body > p > img')
        const btnTag = document.querySelector('body > button')
        const url = "https://api.thedogapi.com/v1/images/search"
        btnTag.addEventListener('click', (e)=> {
            axios.get(url)
```

```
                .then((response) => {
                    imgTag.setAttribute('src', response.data[0].url)
                })
                .catch((error) => {
                    btnTag.innerHTML = '실패'
                })
        })
    </script>

</body>
</html>
```

- 추가로 계속 생성하기

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <article></article>
    <button>클릭</button>

    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script>
        const btnTag = document.querySelector('body > button')
        const bodyTag = document.querySelector('body')
        const articleTag = document.querySelector('article')
        const url = "https://api.thedogapi.com/v1/images/search"
        btnTag.addEventListener('click', (e)=> {
            e.preventDefault()
            axios.get(url)
                .then((response) => {
                    const img = document.createElement('img')
                    articleTag.appendChild(img)
                    img.setAttribute('src', response.data[0].url)
                })
                .catch((error) => {
                    btnTag.innerHTML = '실패'
                })
        })
    </script>
</body>
</html>
```

  - 여기서 주의 할 점은
    - axios.get({url 주소})에 대한 성공시 응답으로

      .then(response) 에서 response의 data (response.data)를 확인 후 원하는 데이터 가져오기

- 정리
  - axios는 비동기로 데이터 통신을 가능하게 하는 라이브러리
  - 같은 방식으로 우리가 배운 Django REST API로 요청을 보내서 데이터를 받아온 후 처리할 수 있다.

Callback과 Promise

- 비동기 처리의 단점

    - 비동기 처리의 핵심은 Web API로 들어오는 순서가 아니라
      작업이 완료되는 순서에 따라 처리한다는 것.
    - 단점으로는 실행 결과를 예상하면서 코드를 작성할 수 없게 한다.
    - 해결 방법: 콜백 함수를 사용하자.

Callback Function (콜백 함수)

- 콜백 함수란?
    - 다른 함수의 인자로 전달되는 함수를 의미한다.
    - 동기식, 비동기식 모두 사용 가능
    - 시간이 걸리는 비동기 작업이 완료된 후 실행할 작업을 명시하는 데 사용되는
      콜백 함수를 비동기 콜백이라 부른다.

```
// Ex.
const btn = document.querySelector('button')
btn.addEventListener('click',() ={
    alert('GG')
})<body>
    <button></button>
    <script>
        const btn = document.querySelector('body>button')
        btn.addEventListener('click',() => {
            alert('GG')
        })
    </script>
</body>
```

    - 콜백 함수를 사용하는 이유
        - 비동기 처리를 순차적으로 동작할 수 있게 한다.
        - 비동기 처리를 위해서는 콜백 함수의 형태가 반드시 필요하다.
        - ex. 요청이 들어오면 or 이벤트가 발생하면 or 데이터를 받아오면 등의 조건을 추가

- 콜백 지옥(callback hell)
    - 코드의 가독성을 해치고
    - 유지 보수가 어려워짐

##

Promise (프로미스)

- Callback Hell 문제를 해결하기 위해 등장한 비동기 처리를 위한 객체
- 의미: "작업이 끝나면 실행 시켜줄게" 라는 약속
- 비동기 작업의 완료 또는 실패를 나타내는 객체
- Promise 기반의 클라이언트가 바로 이전에 사용한 Axios 라이브러리
    - 성공에 대한 약속 .then(callback function)
    - 실패에 대한 약속 .catch(callback function)

- then 과 catch 모두 항상 promise 객체를 반환하기에 계속해서 chaining을 할 수 있음
- axios로 처리한 비동기 로직이 항상 promise 객체를 반환하기에 then을 게속 이어 나가며 작성 가능

```
// 기존의 콜백 함수 작성 방식

work1(function () {
    work2(result1, function (result2) {
        work3(result2, function (result3) {
            console.log('최종 결과 : ' + result3)
        })
    })
})


// promise 방식

work1()
    .then((result1) =>{
          return result2
          })
    .then((result2) =>{
        return result3
    })
    .catch((error) =>{
    console.log(error)
    })
```

- Promise가 보장하는 것 (vs 비동기 콜백)
  - 비동기 콜백 작성 스타일과 달리 Promise가 보장하는 특징
    1. callback 함수는 JS의 Event Loop가 현재 실행 중인 Call Staack을 완료하기 이전에는 절대 호출되지 않음
    2. 비동기 작업이 성공하거나 실패한 뒤에 .then() 메서드를 이용하여 추가한 경우에도 1번과 똑같이 동작
    3. .then()을 여러 번 사용하여 여러 개의 callback 함수를 추가할 수 있다(Chaning)
       - 각각의 callback은 주어진 순서대로 하나하나 실행하게 됨
       - Chaning은 Promise의 가장 뛰어난 장점

AJAX

- AJAX란?
  - 비동기 통신을 이용하면 화면 전체를 새로고침 하지 않아도 서버로 요청을 보내고, 데이터를 받아 화면의 일부분만 업데이트 가능
  - AJAX의 특징
    1. 페이지 새로고침 없이 서버에 요청
    2. 서버로부터 응답(데이터)을 받아 작업을 수행
  - 이러한 비동기 웹 통신을 위한 라이브러리 중 하나가 Axios

Async(비동기) 적용하기

1. html 페이지 생성
2. axios CDN 넣어주기
3. script 태그 생성
4. form 요소 선택을 위해 id 속성 지정 및 선택
   - 불필요해진 action과 method 속성은 삭제 (요청은 axios로 대체되기 때문)

5. form 요소에 이벤트 핸들러 작성 미 submit 이벤트 취소 => event.preventDefault()
6. axios 요청 준비
    - get() 안의 요소로는 url 뿐만 아니라 객체도 들어갈 수 있다.

```
<script>
    const form = document.querySelector('#follow-form')
    form.addEventListener('submit', (event)=> {
        event.preventDefault()
        axios({
            method: 'post',
            url: `/accounts/${???}/follow`,
        })
    })
</script>
```

- 현재 axios로 POST 요청을 보내기 위해 필요한 것
    1. url에 작성할 user pk는 어떻게 작성해야 할지.
        - url에 작성할 user pk 가져오기 (HTML -> JavaScript)
            - form태그 안에 data-user-id 속성의 이름으로 context로 받아온 person의 pk값인 {{person.pk}}을 넣어주고
        - script 태그 안에서 userId를 가져오려고 할때는 event.target.dataset.userId 를 통해 담아준다.
            - 주의
                - html에선 케밥 케이스
                - JS에선 카멜 케이스

```
<form id="follow-form" data-user-id="{{person.pk}}">

</form>

<script>
    const form = document.querySelector('#follow-form')
    form.addEventListener('submit', (event) => {
        event.preventDefault()
        const userId = event.target.dataset.userId
        axios({
            method: 'post',
            url: `/accounts/${userId}/follow`,
        })
    })

</script>
```

2. method가 post 이기에 csrftoken은 어떻게 보내야 할지.
    - 먼저 csft 값은 input type이 hidden 으로 되어있다.

        그래서 hidden 타입으로 숨겨져있는 csrf 값을 가진 input 태그를 선택해야 한다.

```
<script>
    const form = document.querySelector('#follow-form')
    const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

    // 이후 axios({header: {'X-CSRFToken': csrftoken}]}) 으로 넣어주면 된다.
</script>
```

3. views.py와 연동

- 팔로우 버튼을 토글하기 위해서는 현재 팔로우가 된 상태인지 여부 확인이 필요

  axios 요청을 통해 받는 response 객체를 활용해 view 함수를 통해서 팔로우 여부를 파악 할 수 있는 변수를 담아 JSON 타입으로 응답하기

```
from django.http import JsonResponse

@require_POST
def follow(reequest, user_pk):
    if request.user.is_authenticated:
        User = get_user_model()
        me = request.user
        you = User.objects.get(pk=user_pk)
        if me != you:
            if you.followers.filter(pk=me.pk).exist():
                you.followers.remove(me)
                is_followed = False
            else:
                you.followers.add(me)
                is_followed = True
            context = {
                'is_followed' : is_followed
            }
            return JsonResponse(context)
```

- view 함수에서 응답한 is_followed를 사용해 버튼 토글하기

```
<body>
    <form id="follow-form" data-user-id="{{user.pk}}"></form>

    <script>
        const form = document.querySelector('#follow-form')
        form.addEventListener('submit', (event)=>{
            event.preventDefault()

            const userID = event.target.dataset.userId
            const csrftoken = document.querySelector('[name:csrfmiddlewaretoken]').value
            axios({
                method: 'post',
                url : `/accounts/${userId}/follow`,
                header: {'X-CSRFToken': csrftoken}
                })
                .then((response) => {
                const isFollowed = response.data.is_followed
                const followBtn = document.querySelector('#follow-form > input[type=submit]')
                if (isFollowed==='true') {
                    followBtn.value = '언팔로우'
                } else {
                    followBtn.value = '팔로우'
                }
            })
            })
    </script>
</body>
```

XHR

- 'XMLHttpRequest'
- AJAX 요청을 생성하는 JavaScript API
- XHR의 메서드로 브라우저와 서버 간 네트워크 요청을 전송할 수 있다.
- Axios는 손쉽게 XHR을 보내고 응답 결과를 Promise 객체로 반환해주는 라이브러리

팔로워 & 팔로잉 수 비동기 적용

- 해당 요소를 선택할 수 있도록 span 태그와 id 속성 작성

```
<block>
    <h1> {{person.username}}님의 프로필</h1>
    <div>
        팔로워 : <span id="followers-count">{{ person.followers.all | length }}</span>
        팔로잉 : <span id="followings-count">{{ person.followings.all | length }}</span>
    </div>

</block>

<script>
        const form = document.querySelector('#follow-form')
        form.addEventListener('submit', (event)=>{
            event.preventDefault()

            const userID = event.target.dataset.userId
            const csrftoken = document.querySelector('[name:csrfmiddlewaretoken]').value
            axios({
                method: 'post',
                url : `/accounts/${userId}/follow`,
                header: {'X-CSRFToken': csrftoken}
                })
                .then((response) => {
                    const isFollowed = response.data.is_followed
                    const followBtn = document.querySelector('#follow-form > input[type=submit]')
                    if (isFollowed==='true') {
                        followBtn.value = '언팔로우'
                    } else {
                        followBtn.value = '팔로우'
                    }
                    const followersCountTag = document.querySelector('#followers-count')
                    const followingsCountTag = document.querySelector('#followings-count')
                    followersCountTag.innerText= followersCount
                    followingsCountTag.innerText = followingsCount
                 })
            })
    </script>
```

- 팔로워, 팔로잉 인원 수 연산은 view 함수에서 진행하여 결과를 응답으로 전달

```
from django.http import JsonResponse

@require_POST
def follow(request, user_pk):
    context= {
        'is_followed': is_followed,
        'followers_count': you.followers.count(),
        'followings_count': you.followings.count()
    }
    return JsonResponse(context)
```