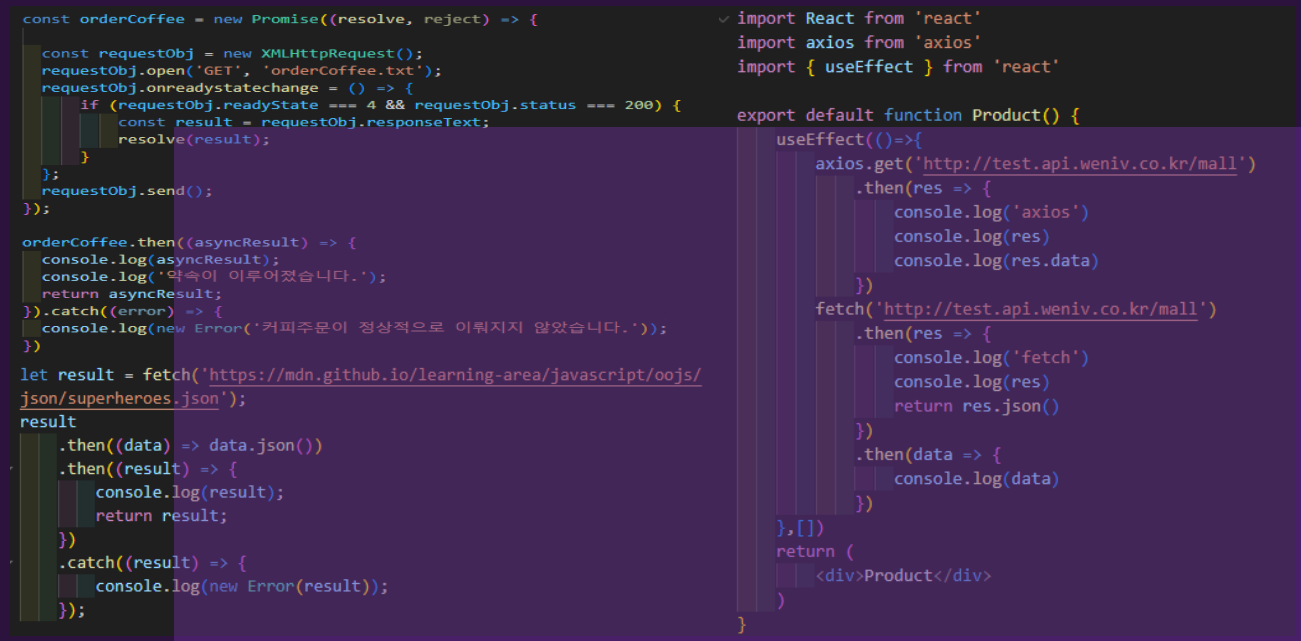


```
const orderCoffee = new Promise((resolve, reject) => {

  const requestObj = new XMLHttpRequest();
  requestObj.open('GET', 'orderCoffee.txt');
  requestObj.onreadystatechange = () => {
      if (requestObj.readyState === 4 && requestObj.status === 200) {
          const result = requestObj.responseText;
          resolve(result);
      }
  };
  requestObj.send();
});

orderCoffee.then((asyncResult) => {
  console.log(asyncResult);
  console.log('약속이 이루어졌습니다.');
  return asyncResult;
}).catch((error) => {
  console.log(new Error('커피주문이 정상적으로 이뤄지지 않았습니다.'));
})

let result = fetch('https://mdn.github.io/learning-area/javascript/oojs/
json/superheroes.json');
result
    .then((data) => data.json())
    .then((result) => {
        console.log(result);
        return result;
    })
    .catch((result) => {
        console.log(new Error(result));
    });
```



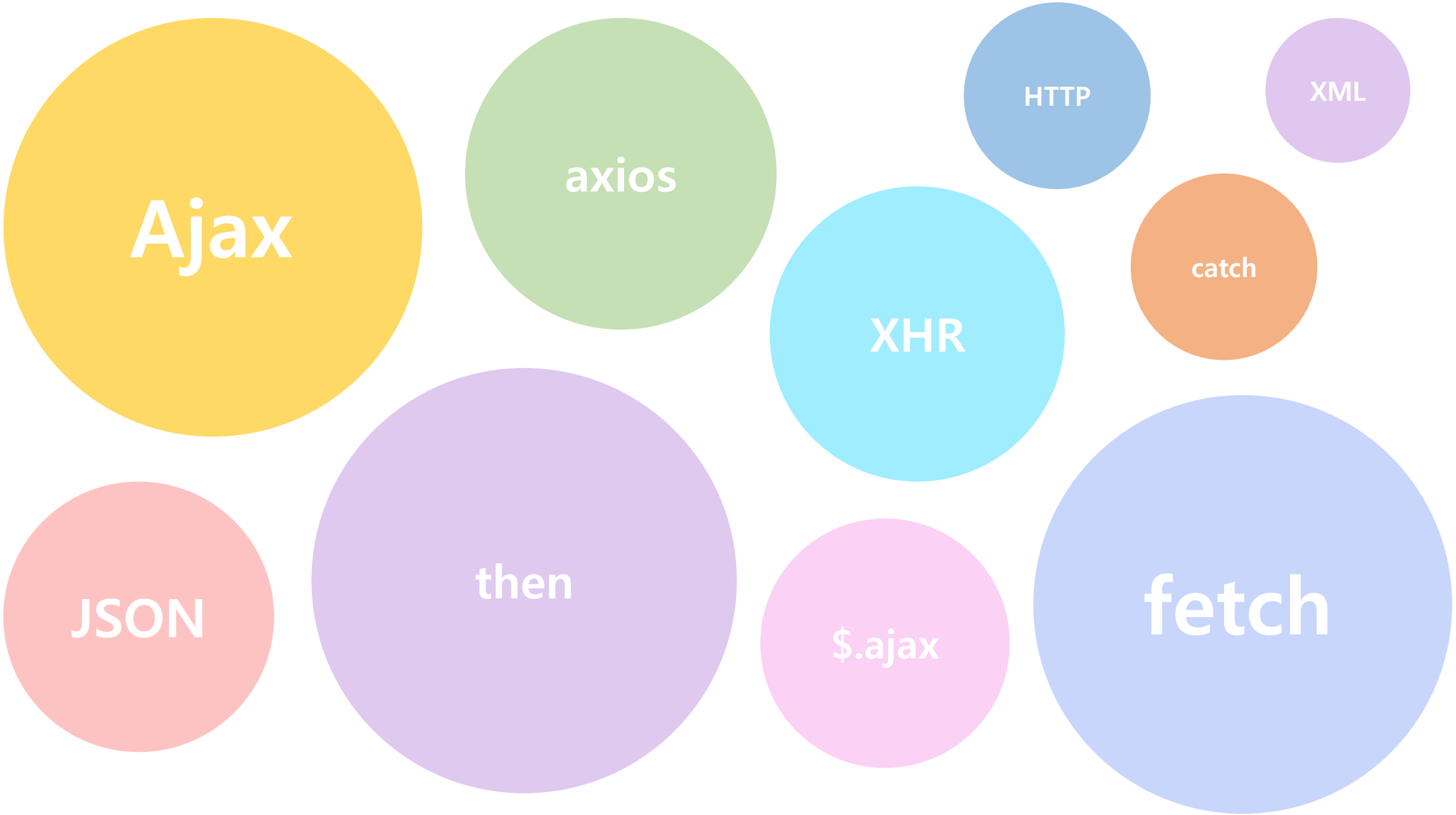
```
import React from 'react'
import axios from 'axios'
import { useEffect } from 'react'

export default function Product() {
    useEffect(()=>{
        axios.get('http://test.api.weniv.co.kr/mall')
            .then(res => {
                console.log('axios')
                console.log(res)
                console.log(res.data)
            })
        fetch('http://test.api.weniv.co.kr/mall')
            .then(res => {
                console.log('fetch')
                console.log(res)
                return res.json()
            })
            .then(data => {
                console.log(data)
            })
    },[])
    return (
        <div>Product</div>
    )
}
```

Ajax

정혜민

Ajax

에이젝스 아작아작 씹어먹어 주겠쓰
아작쓰!


Ajax
axios
HTTP
XML
XHR
catch
JSON
then
$.ajax
fetch

CONTENTS

01

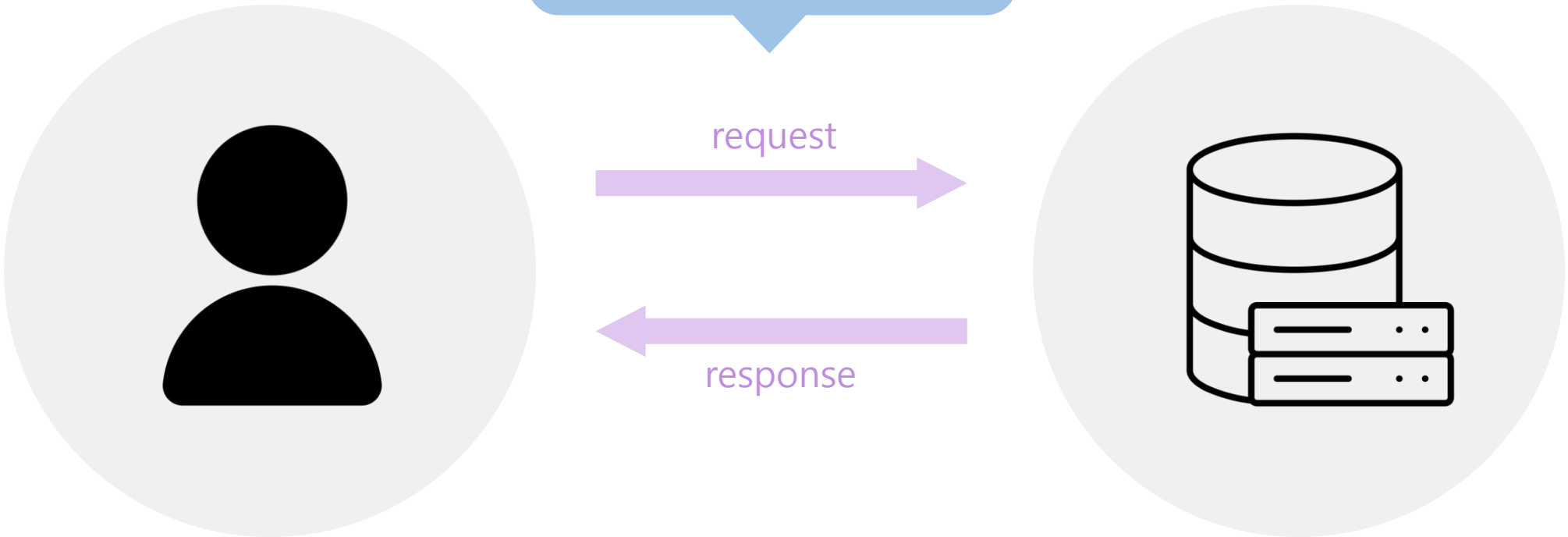
들어가기

HTTP / HTTPS

HTTP

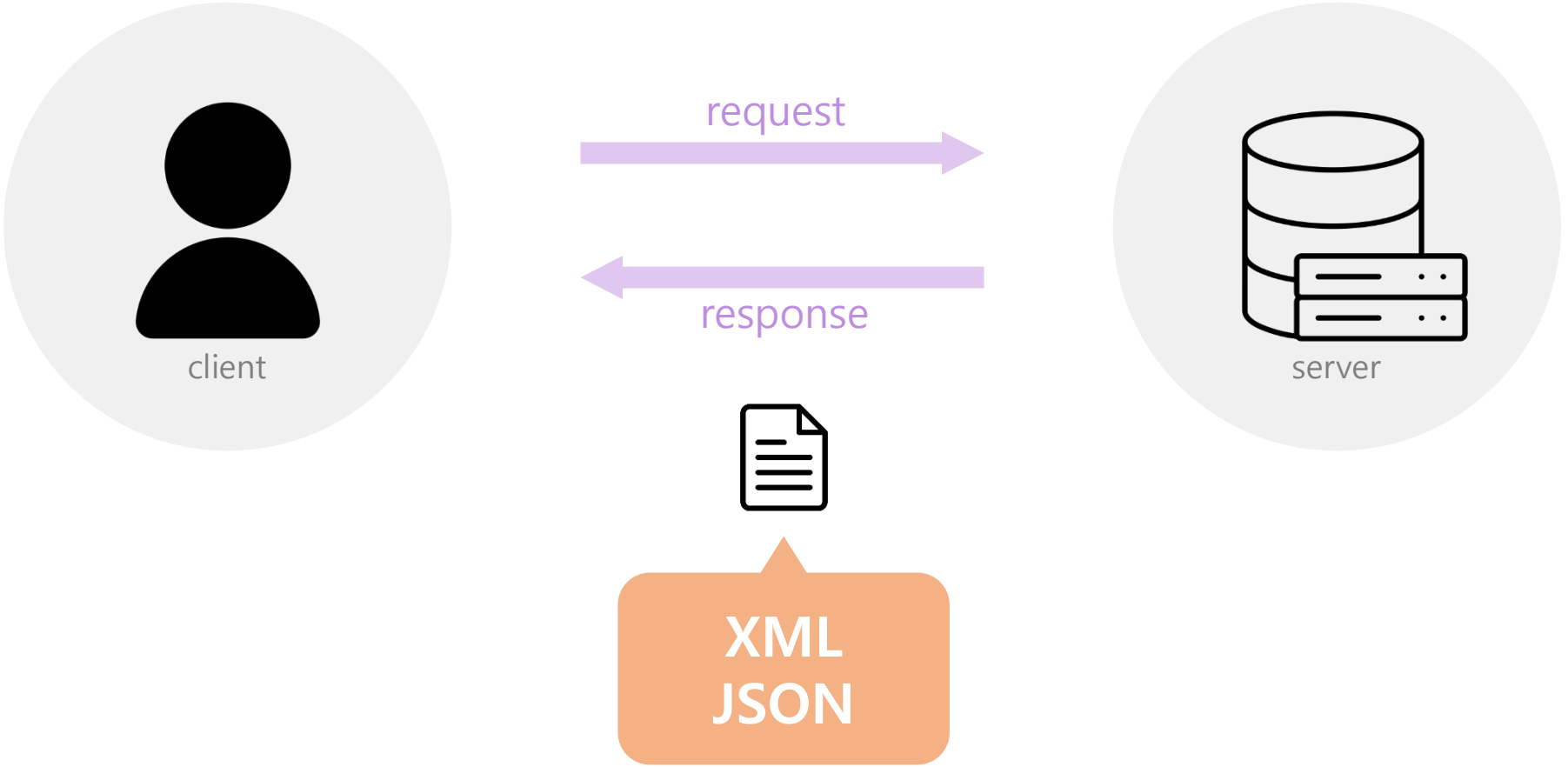
(Hypertext Transfer Protocol)

HTTP
HTTPS



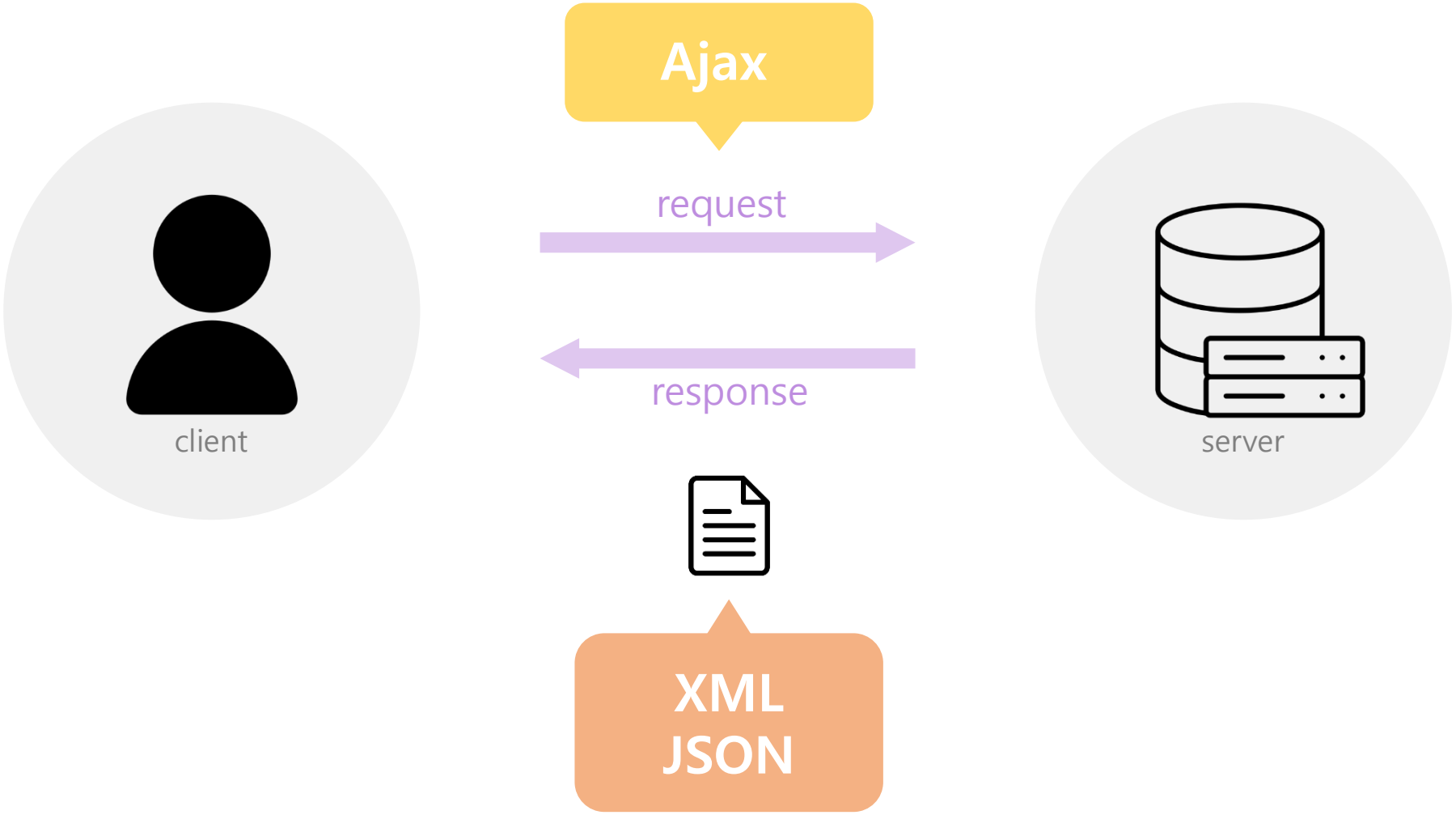
XML · JSON

데이터 주고 받는 형식

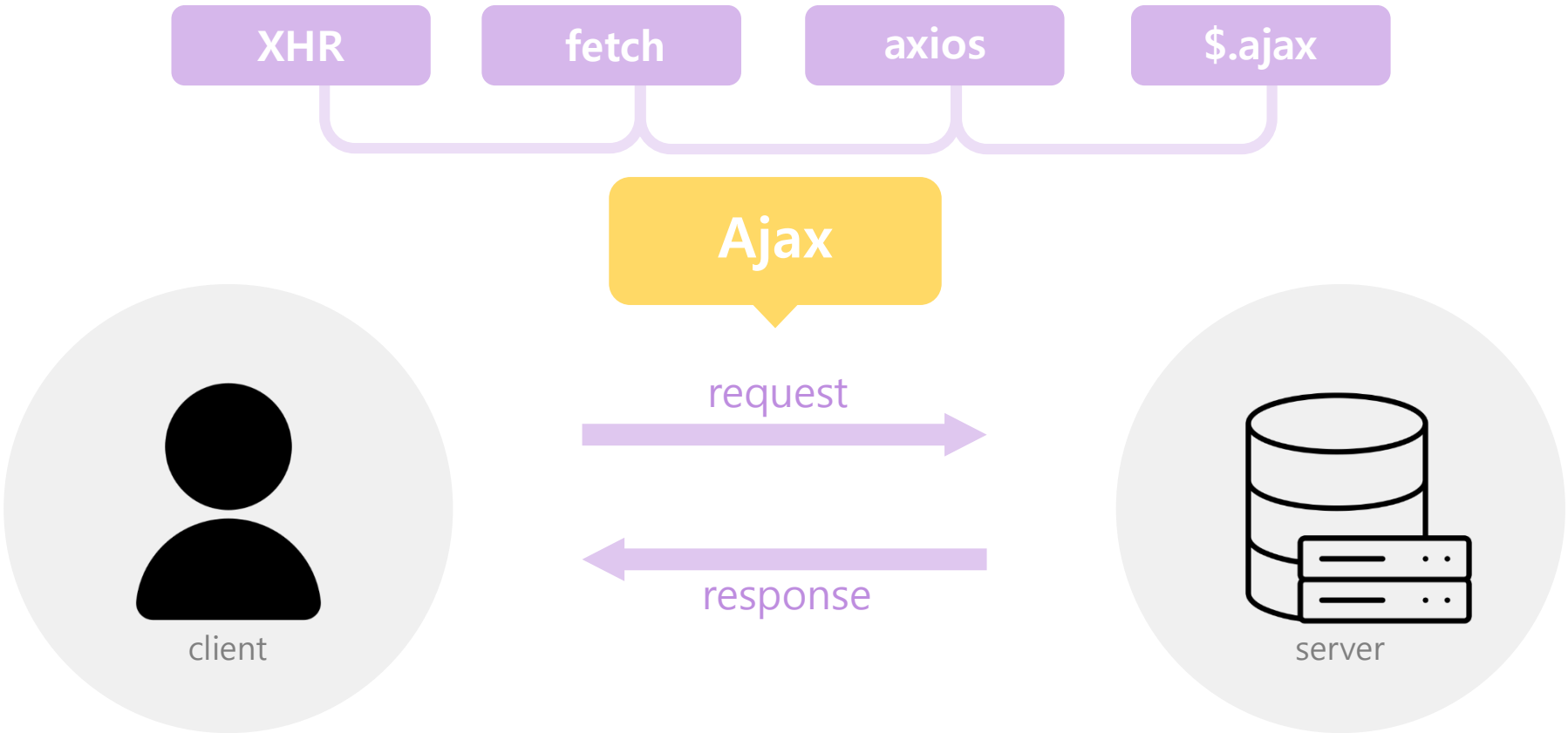


AJAX

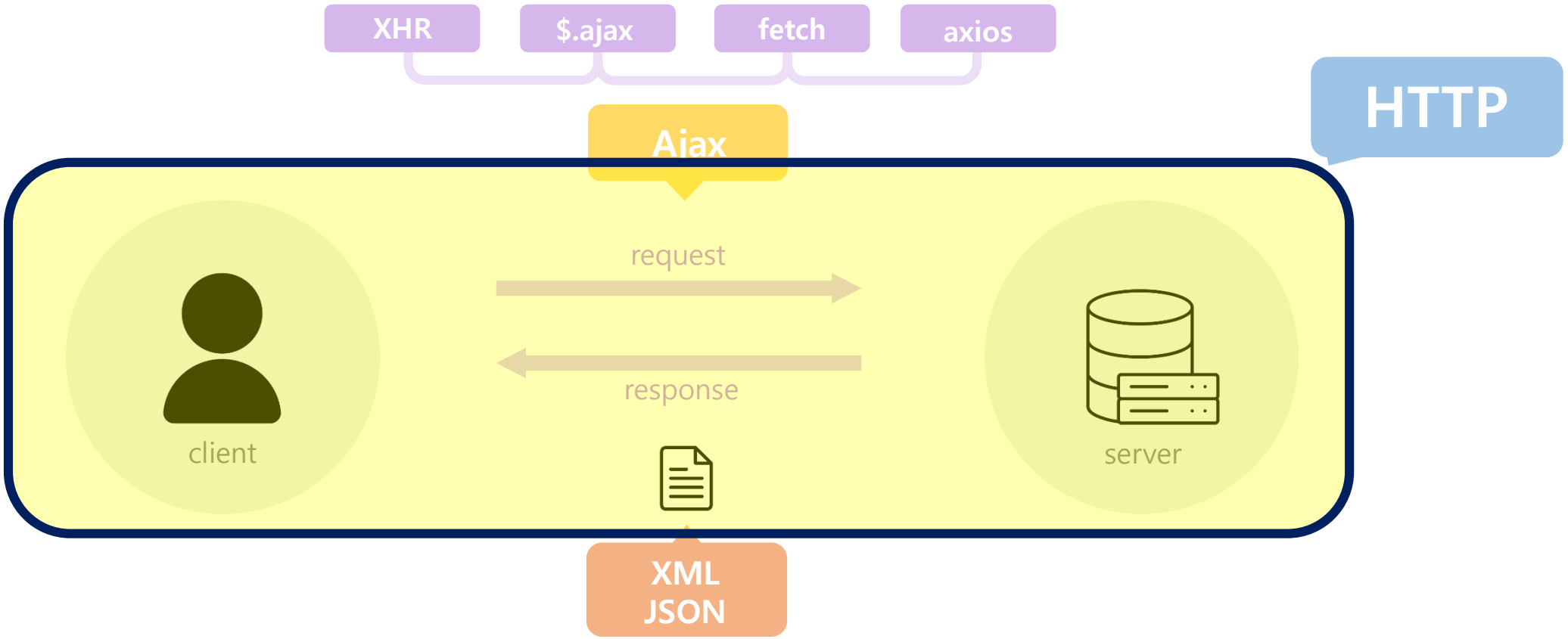
Ajax ?



Ajax ?



Ajax ?


XHR
$.ajax
fetch
axios
Ajax
HTTP
request
response
client
server
XML
JSON

02

Ajax

Ajax ?

XHR
(XMLHttpRequest)

$.ajax

fetch

axios

XHR

XHR
(XMLHttpRequest)

XMLHttpRequest

XMLHttpRequest 객체는 **웹 브라우저가 서버와 데이터를 교환**할 때 사용

서버와의 비동기 통신을 가능하게 하는 여러 기능들을 가진 자바스크립트 객체

브라우저 API에서 제공하는 object

간단하게 서버에게 데이터를 요청하고 받아올 수가 있음

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

XMLHttpRequest 객체 생성

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

onreadystatechange 등록

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

readyState 프로퍼티

status 프로퍼티

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

응답 데이터 프로퍼티

XMLHttpRequest

XHR
(XMLHttpRequest)

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if(this.readyState == 4 && this.status == 200){
    console.log(xhr.responseText)
  }
};
xhr.open("GET",'url',true);
xhr.send();
```

서버로 보낼 Ajax 요청의 형식을 설정

작성된 Ajax 요청을 서버로 전달

XMLHttpRequest

```
searchUser(userName, password) {
        return new Promise((resolve, reject) => {
          const xhr = new XMLHttpRequest();
          xhr.open('GET', 'users.json');
          xhr.onreadystatechange = () => {
            if (xhr.readyState === 4 && xhr.status === 200) {
              const result = JSON.parse(xhr.responseText).user.find((item) => {
                return item.userName === userName && item.password === password
              });
              if (result) {
                resolve(userName);
                // onSuccess(userName);
              } else {
                reject('user not found');
                // onError(new Error('user not found'));
              }
            }
          }
          xhr.send();
        })
      }
```

XMLHttpRequest

```
searchUser(userName, password) {
        return new Promise((resolve, reject) => {
          const xhr = new XMLHttpRequest();
          xhr.open('GET', 'users.json');
          xhr.onreadystatechange = () => {
            if (xhr.readyState === 4 && xhr.status === 200) {
              const result = JSON.parse(xhr.responseText).user.find((item) => {
                return item.userName === userName && item.password === password
              });
              if (result) {
                resolve(userName);
                // onSuccess(userName);
              } else {
                reject('user not found');
                // onError(new Error('user not found'));
              }
            }
          }
          xhr.send();
        })
      }
```

???

XMLHttpRequest

```
searchUser(userName, password) {
    return new Promise((resolve, reject) => {
      const xhr = new XMLHttpRequest();
      xhr.open('GET', 'users.json');
      xhr.onreadystatechange = () => {
        if (xhr.readyState === 4 && xhr.status === 200) {
          const result = JSON.parse(xhr.responseText).user.find((item) => {
            return item.userName === userName && item.password === password
          });
          if (result) {
            resolve(userName);
            // onSuccess(userName);
          } else {
            reject('user not found');
            // onError(new Error('user not found'));
          }
        }
      }
      xhr.send();
    })
  }
```

XMLHttpRequest

```
searchUser(userName, password) {
    return new Promise((resolve, reject) => {
      const xhr = new XMLHttpRequest();
      xhr.open('GET', 'users.json');
      xhr.onreadystatechange = () => {
        if (xhr.readyState === 4 && xhr.status === 200) {
          const result = JSON.parse(xhr.responseText).user.find((item) => {
            return item.userName === userName && item.password === password
          });
          if (result) {
            resolve(userName);
            // onSuccess(userName);
          } else {
            reject('user not found');
            // onError(new Error('user not found'));
          }
        }
      }
      xhr.send();
    })
  }
```

$.ajax

$.ajax

삐빅- 죄송합니다.
$.ajax는 다루지 않겠습니다. 빵긋 ☺

fetch

fetch

fetch

XHR은 성능에는 문제는 없지만 코드가 복잡하고 가독성이 좋지 않다는 단점

fetch API는 Promise 기반으로 구성되어 있어 **비동기 처리 프로그래밍 방식**에 잘 맞는 형태

then이나 catch 와 같은 체이닝으로 작성할 수 있다는 장점

Promise반환. Promise는 비동기 통신을 매우 쉽게 만들어 줌

fetch

```
let result = fetch('url');
result
    .then((data) => data.json())
    .then((result) => {
        console.log(result);
        return result;
    })
    .catch((result) => {
        console.log(new Error(result));
    });
```

- fetch는 함수처럼 바로 실행 가능
- 완료되면 바로 fulfilled 상태의 프로미스가 반환
- then을 통해 데이터를 가져옴
- .json() 메소드를 통해 자바스크립트 객체로 변환이 가능

fetch

```
fetch("url")
  .then((response) => {
    if (!response.ok) {
      throw new Error("400 아니면 500 에러남");
    }
    return response.json();
  })
  .then((결과) => {
    console.log(결과);
  })
  .catch(() => {
    console.log("에러남");
  });
```

정확히 에러를 catch하고 싶을 때

- response.ok – HTTP 상태 코드가 200과 299 사이일 경우 true
- response.json() – 응답을 JSON 형태로 파싱한다,

fetch

```
async function 데이터가져오는함수() {
  const response = await fetch("url");
  if (!response.ok) {
    throw new Error("400 아니면 500 에러남");
  }
  const result = await response.json();
  console.log(result);
}
데이터가져오는함수().catch(() => {
  console.log("에러남");
})
```

async / awai (then x)

axios

Axios

운영 환경에 따라 브라우저의 XMLHttpRequest 객체 또는 Node.js의 http api 사용

Promise(ES6) API 사용

HTTP 요청과 응답을 JSON 형태로 자동 변경

axios

```
axios.get('url')
  .then((result)=>{
    console.log(result.data)
  }).catch(()=>{
    console.log('에러남')
  })
```

HTTP 요청과 응답을 JSON 형태로 자동 변경

axios

```
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

async / await

axios vs fetch

```
import React from 'react'
import axios from 'axios'
import { useEffect } from 'react'

export default function Product() {
    useEffect(()=>{
        axios.get('http://test.api.weniv.co.kr/mall')
            .then(res => {
                console.log('axios')
                console.log(res)
                console.log(res.data)
            })
        fetch('http://test.api.weniv.co.kr/mall')
            .then(res => {
                console.log('fetch')
                console.log(res)
                return res.json()
            })
            .then(data => {
                console.log(data)
            })
    },[])
    return (
        <div>Product</div>
    )
}
```

axios / fetch

03

정리하기

XHR
(XMLHttpRequest)

XMLHttpRequest

XMLHttpRequest 객체는 **웹 브라우저가 서버와 데이터를 교환**할 때 사용

서버와의 비동기 통신을 가능하게 하는 여러 기능들을 가진 자바스크립트 객체

브라우저 API에서 제공하는 object

간단하게 서버에게 데이터를 요청하고 받아올 수가 있음

fetch

fetch

XHR은 성능에는 문제는 없지만 코드가 복잡하고 가독성이 좋지 않다는 단점

fetch API는 Promise 기반으로 구성되어 있어 **비동기 처리 프로그래밍 방식**에 잘 맞는 형태

then이나 catch 와 같은 체이닝으로 작성할 수 있다는 장점

Promise반환. Promise는 비동기 통신을 매우 쉽게 만들어 줌

Axios

Axios는 브라우저, Node.js를 위한 Promise API를 활용하는 HTTP 비동기 통신 라이브러리

Axios 설치

Promise(ES6) API 사용

HTTP 요청과 응답을 JSON 형태로 자동 변경

Q & A

Reference

서적

이웅모,『모던 자바스크립트 Deep Dive』, 위키북스

인터넷자료

"Inpa Dev, "XMLHttpRequest 으로 AJAX 요청하기"·"Fetch API 으로 AJAX 요청하기"
· "AXIOS 설치 & 특징 & 문법 💯 정리", Inpa Dev,
2021, https://inpa.tistory.com/

기타

멋쟁이사자처럼 5기 교안,
https://likelion.notion.site/14-AJAX-438d88e05e7b4bd1b444899c34ffc4e7

감사합니다