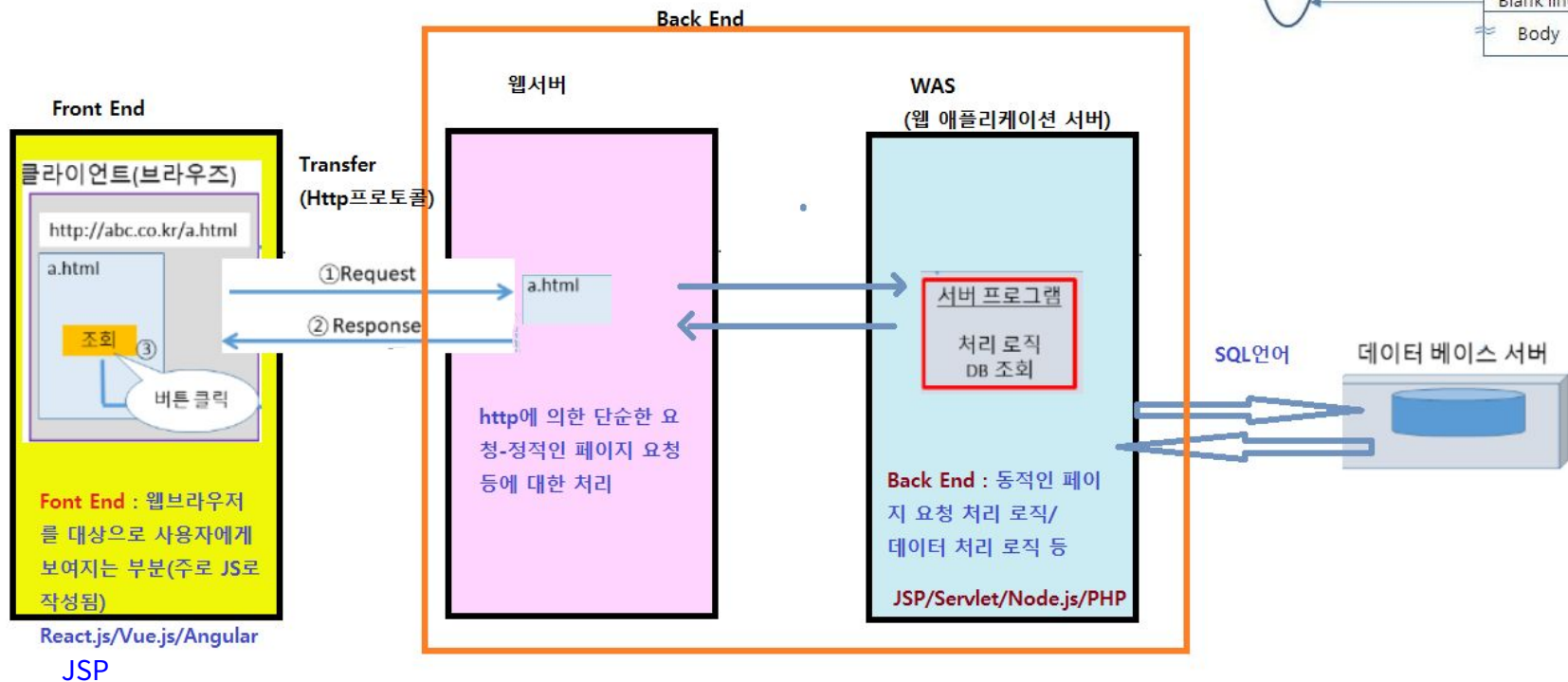


웹 아키텍처에 대한 이해

본 자료는 개인 학습용으로만 사용하고 웹이나 타인에게 배포를 금합니다

2024-08 백성애

웹 서비스 구조 (full stack)



데이터 통신에서 알아야 할 개념-HTTP와 메서드

- 데이터 통신은 컴퓨터와 디바이스, 서버와 클라이언트 간에 데이터를 주고받는 과정을 나타낸다.
- 데이터 통신을 하려면 특정 규칙과 규약이 필요한데 이를 통신 프로토콜이라고 한다.
- 통신 프로토콜을 통해 서버와 클라이언트가 서로 이해할 수 있는 언어로 통신한다
- 주요 프로토콜로는 HTTP, HTTPS, TCP,UDP, FTP 등이 있다.
- 웹 브라우저는 HTTP를 기본으로 사용한다.
- HTTP(Hyper Text Transfer Protocol)는 클라이언트와 서버간 통신 프로토콜로 데이터를 주고 받기 위한 규칙과 표준을 정의한다

클라이언트와 서버의 HTTP 통신의 예

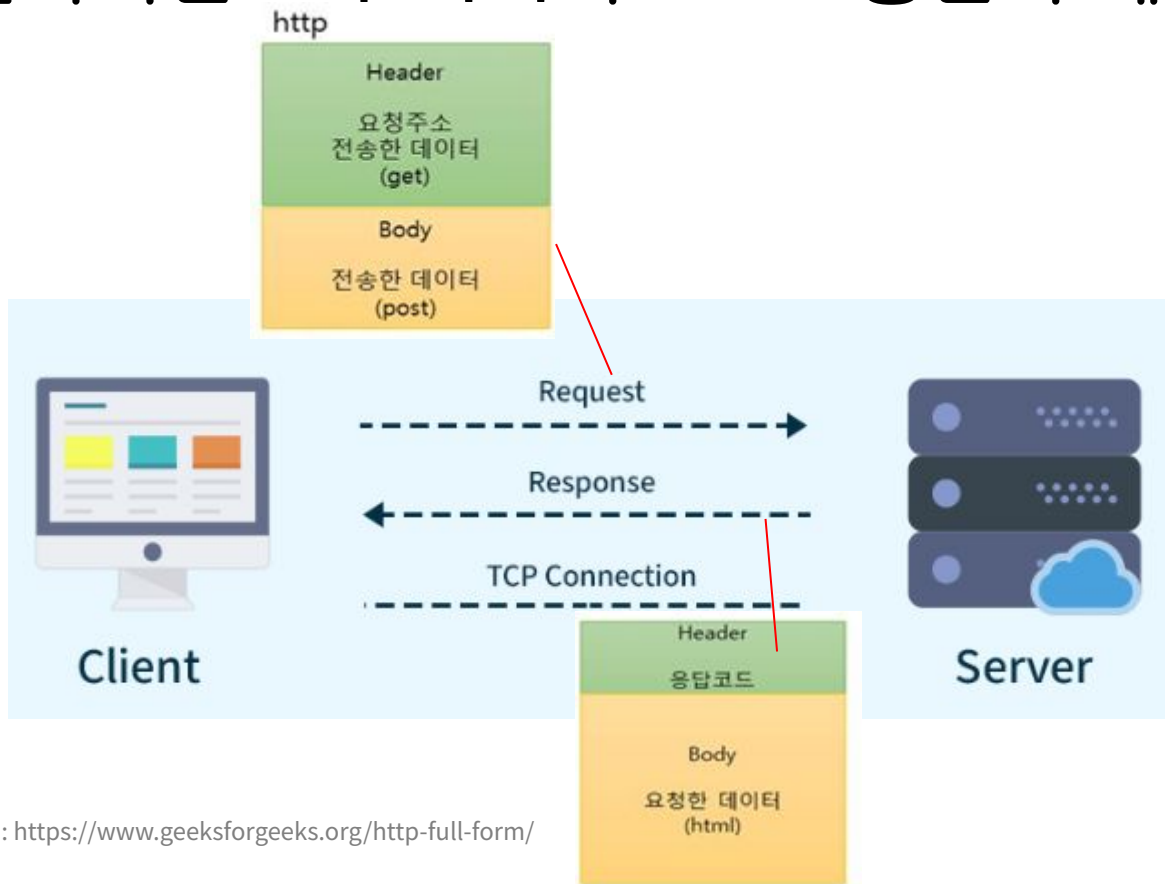


그림: <https://www.geeksforgeeks.org/http-full-form/>

HTTP 메시지 - 요청(request)과 응답(response)

HTTP 메시지는 서버와 클라이언트 간에 데이터가 교환되는 방식입니다. 이때 메시지 타입은 두 가지가 있습니다. Request는 클라이언트가 서버로 전달하는 메시지이고, Response는 Request에 대한 서버의 답변입니다. HTTP 메시지는 ASCII로 인코딩된 텍스트 정보이며, 이러한 메시지는 설정파일 (프록시, 서버), API (브라우저 경우) 혹은 다른 기타의 인터페이스에 의해 가공되어 제공됩니다.

메시지는 세 부분(공백 제외) 으로 구성되어 있습니다.



3. HTTP Request 메시지

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

A blank line separates header & body

Request Message Body

Request Message Header

4. HTTP Response 메시지

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

A blank line separates header & body

Response Message Body

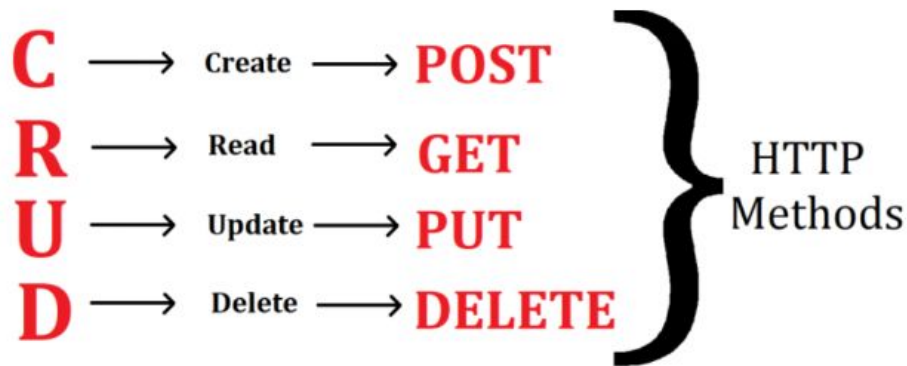
Response Message Header

HTTP 메서드

HTTP 요청에서는 HTTP 메서드를 사용한다.

HTTP 메서드는 클라이언트가 서버에 요청하는 작업이 무엇인지 나타내며,

HTTP 메서드를 통해 특정 작업을 수행할 수 있다.



HTTP Method

GET : 특정 리소스를 조회할 때 보내는 요청 메서드입니다.
POST : 리소스에 정보를 추가할 때 보내는 요청 메서드입니다.
PUT : 리소스의 모든 것을 수정하기 위한 요청 메서드입니다.
PATCH : 리소스의 일부만을 수정하기 위한 요청 메서드입니다.
DELETE : 리소스를 삭제할 때 보내는 요청 메서드입니다.

HTTP 응답 상태 코드

상태	코드	메시지	설 명
정상	200	OK	클라이언트의 요청은 정상적으로 처리됨
리 다이렉트	304	Not Modified	요청된 문서는 변경되지 않음
클라이언트 에러	400	Bad Request	비정상적인 요청임
	404	Not Found	요청된 URI가 서버상에 존재하지 않음
서버에러	500	Internal Sever Error	서버 내부에 문제가 발생함
	503	Service Unavailable	서비스를 이용할 수 없음

출 처: <https://velog.io/@still3028/TIL-20210430HTTP-Request-Http-Response-CORS>

RESTful 이란?

RESTful은 **REST(Representational State Transfer-표현 상태 전이)** 아키텍처 스타일을 따르는 웹 서비스 디자인 방법론이다.

REST는 **HTTP** 프로토콜을 기반으로 하는 웹 서비스의 설계 원칙을 정의하며,

RESTful 웹 서비스는 이러한 원칙을 준수하여 구현된다. **REST**는 웹의 기존 기능을 최대한 활용하는 데 중점을 둔다.

REST는 웹의 **HTTP** 프로토콜을 기반으로 하며, **CRUD(Create, Read, Update, Delete)** 작업을 **HTTP 메서드(GET, POST, PUT, DELETE 등)**로 매핑하여 사용한다. 이러한 방식은 **RESTful 웹 서비스**를 설계하는 데 널리 사용되며, **API**의 설계와 구현을 단순화하고 일관성 있게 만든다

Rest의 주요 원칙

REST의 주요 원칙

1. 자원 (Resource):

- 모든 것은 자원으로 간주됩니다.
- 자원은 URI(Uniform Resource Identifier)로 고유하게 식별됩니다.
- 예: <https://api.example.com/users/123>는 ID가 123인 사용자 자원입니다.

2. 표현 (Representation):

- 자원은 JSON, XML, HTML 등 다양한 형태로 표현될 수 있습니다.
- 클라이언트와 서버는 자원의 표현을 주고받습니다.

3. 상태 전이 (Stateless):

- 각 요청은 독립적이고 상태가 없습니다.
- 서버는 각 요청을 완전히 이해하기 위해 필요한 모든 정보를 요청에 포함시켜야 합니다.
- 클라이언트의 상태는 클라이언트 측에서 관리됩니다.

Rest의 주요 원칙

4. HTTP 메서드(HTTP Methods):

- 자원에 대한 **CRUD** 작업은 **HTTP** 메서드로 정의됩니다 .
 - **GET** : 자원의 표현을 가져옵니다 .
 - **POST** : 새로운 자원을 생성합니다 .
 - **PUT** : 기존 자원을 업데이트합니다 .
 - **DELETE**: 자원을 삭제합니다 .
 -
- 예: **GET /users/123**는 ID가 **123**인 사용자의 정보를 가져옵니다 .

5. 계층화된 시스템(Layered System):

- 클라이언트는 중간 서버(프록시, 게이트웨이 등)와 상호작용할 수 있으며, 이러한 중간 서버는 요청을 필터링하거나 로드 밸런싱 등을 수행할 수 있습니다.

RESTful 웹 서비스의 설계 예

자원 기반 URI 설계

GET /users -> 모든 사용자 정보 조회

POST /users -> 새 사용자 생성

GET /users/123 -> ID가 123인 사용자 정보 조회

PUT /users/123 -> ID가 123인 사용자 정보 업데이트

DELETE /users/123 -> ID가 123인 사용자 삭제

RESTful API의 장점

단순하고 일관된 인터페이스

- HTTP 프로토콜을 활용하여 일관된 인터페이스를 제공한다. URL만 보고도 무슨 행동을 하는 api인지 명확히 알 수 있다.

확장성

- Stateless한 설계로 인해 서버와 클라이언트의 역할이 명확하게 분리되어 확장이 용이하다.

유연성

- 다양한 표현 형식을 지원하며, 클라이언트와 서버 간의 느슨한 결합을 유지한다.

가독성

- URI를 통해 자원을 명확히 식별할 수 있어 이해하기 쉽다.

RESTful API의 단점

표준 부족

- REST는 아키텍처 스타일로서 특정 구현 표준이 없기 때문에 일관된 구현이 어려울 수 있다.

복잡한 연산의 비효율성

- 대량의 데이터나 복잡한 연산을 수행하는 경우, HTTP 요청/응답의 오버헤드가 성능 저하를 초래할 수 있다.

상태 유지의 어려움

- **Stateless** 특성으로 인해 클라이언트의 상태를 유지하는 데 어려움이 있을 수 있다.

Rest와 Restful의 차이점

- **REST**: 아키텍처 스타일, 디자인 원칙을 의미합니다.
- **RESTful**: **REST** 원칙을 실제로 구현한 웹 서비스를 의미합니다.

RESTful 웹 서비스는 현대 웹 애플리케이션에서 매우 널리 사용되며, 클라이언트와 서버 간의 통신을 효율적으로 관리하는 데 중요한 역할을 합니다.

RESTful 설계의 Stateless 특성

RESTful 서비스에서의 stateless는 서버가 클라이언트의 상태를 전혀 유지하지 않는다는 의미입니다. 각 요청은 독립적이며, 필요한 모든 상태 정보는 요청에 포함되어 있어야 합니다. 클라이언트의 상태는 클라이언트 측에서 관리됩니다.

예를 들어, 클라이언트가 서버에 요청을 보낼 때 세션 정보를 포함하여 서버가 클라이언트의 이전 상태를 기억할 필요 없이 각 요청을 처리할 수 있어야 합니다.

Stateless 설계의 확장성 장점

1. 수평 확장 용이:

- **부하 분산:** Stateless 특성 덕분에, 모든 요청은 독립적이므로 부하 분산 장치(load balancer)를 사용하여 여러 서버에 균등하게 요청을 분산시킬 수 있습니다. 서버는 요청에 대한 상태 정보를 유지하지 않으므로, 클라이언트는 특정 서버에 종속되지 않고 아무 서버에나 요청을 보낼 수 있습니다.
- **서버 추가/제거의 용이성:** 서버를 추가하거나 제거하는 것이 간단합니다. 각 서버는 클라이언트의 상태를 유지할 필요가 없기 때문에, 특정 서버의 상태와 무관하게 다른 서버가 요청을 처리할 수 있습니다.

2. 장애 허용성 증가:

- **서버 장애:** 특정 서버가 장애를 일으키더라도, 클라이언트는 다른 서버에 요청을 보낼 수 있습니다. 상태 정보가 서버에 저장되지 않기 때문에, 서버 간의 장애가 전체 서비스에 미치는 영향이 적습니다.
- **유지보수:** 서버를 재시작하거나 교체하는 과정에서도 클라이언트의 상태 정보를 고려할 필요가 없어 유지보수가 용이합니다.

3. 단순화된 서버 설계:

- **상태 관리 불필요:** 서버는 클라이언트의 상태를 관리할 필요가 없기 때문에, 서버의 설계와 구현이 단순해집니다. 이는 개발과 운영의 복잡성을 줄여줍니다.

예시 1: 클라이언트 요청 예시

plaintext

```
GET /orders/123
```

```
Host: api.example.com
```

```
Authorization: Bearer token123
```

- 이 요청에는 필요한 모든 정보가 포함되어 있어, 어떤 서버라도 이 요청을 처리할 수 있습니다. 서버는 이전 요청의 상태를 기억할 필요가 없습니다.

정리

요약

- **Stateless의 의미**: RESTful 서비스에서 **stateless**는 서버가 클라이언트의 상태를 기억하지 않는다는 것을 의미합니다.
- **확장성의 장점**: 이러한 설계로 인해 서버를 쉽게 추가하거나 제거할 수 있으며, 부하 분산과 장애 허용성이 증가하고, 서버 설계가 단순해집니다.

따라서, RESTful 서비스의 **stateless** 특성은 시스템의 확장성과 안정성을 크게 향상시킵니다.

REST API를 사용하는 방법

- [규칙1]- URL에는 동사를 쓰지 말고, **자원을 표시해야 한다**. 여기서 자원은 가져오는 데이터를 말한다.

예1] `/members/1` [o]

예2] `/getMember?id=1` [x] ⇒ 자원이 아닌 표현을 사용함으로써 추후 개발시 혼란 야기할 수 있다. 어떤 개발자는 `getXXX`, 다른 개발자는 `showXXX`식으로 URL구조에 일관성을 잃을 수 있다.

예3] `/articles/1` [o]

예4] `/articles/show/1` [x]

예5] `/show/articles/1` [x]

REST API를 사용하는 방법

- [규칙2] - 동사는 HTTP 메서드로 표현한다
-

설명	적합한 HTTP 메서드와 URL
id가 1인 POST글을 조회하는 API	GET /api/posts/1
새로운 POST글을 등록하는 API	POST /api/posts
1번 POST글을 수정하는 API	PUT /api/posts/1
1번 POST글을 삭제하는 API	DELETE /api/posts/1

GET,POST,PUT,DELETE는 URL에 입력하는 값이 아니라 내부적으로 처리하는 방식을 미리 정하는 것.

Ajax란

- **Asynchronous JavaScript and Xml**
- 비동기 방식의 자바스크립트와 XML을 의미
- 자체가 하나의 특정 기술을 말하는 것이 아니며, 대화식 웹 애플리케이션의 제작을 위해 함께 사용하는 기술(XML, HTML, CSS, JAVASCRIPT, JSON)의 묶음을 지칭하는 용어이다

AJAX 특징

- **기존의 웹 어플리케이션 동작방식**

- 하나의 요청으로 웹 서버에 전달
- 웹 서버는 요청에 해당하는 응답으로 새로운 웹 페이지를 돌려줌.
- 웹 브라우저는 응답에 해당하는 웹 페이지를 다시 그려줌.
(대여폭 낭비, 사용자와 대화식의 서비스가 어려움)

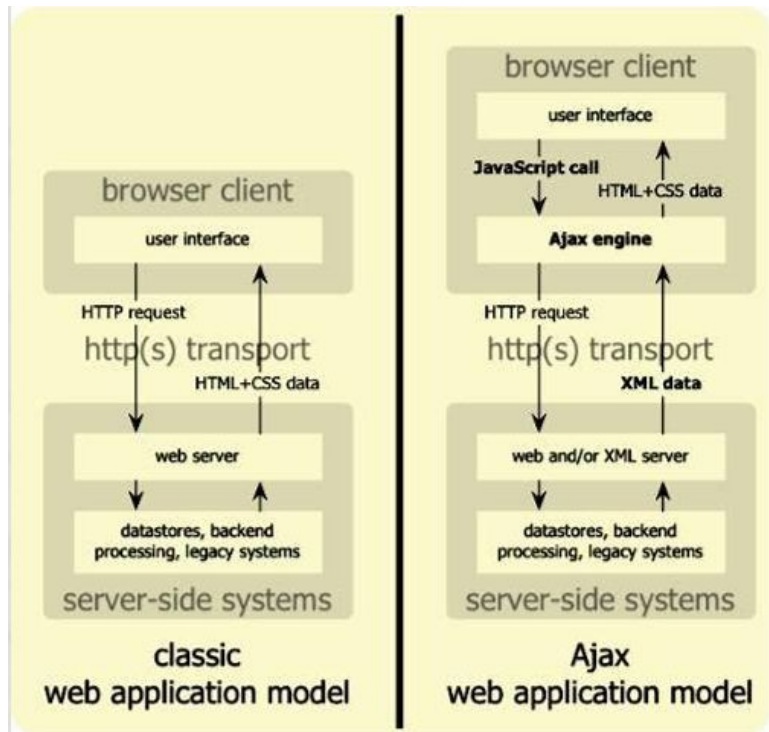
- **AJAX 웹 어플리케이션의 동작방식**

- 여러 개의 비동기 요청으로 웹 서버에 전달
- 웹 서버는 요청에 해당하는 응답으로 필요한 데이터만을 돌려줌.
- 웹 브라우저의 자바스크립트를 이용하여 화면의 필요한 부분을 갱신함.
(대여폭이 줄어듦, 사용자와 대화식 서비스가 가능해짐, 웹 서버는 필요한 데이터만 생산하므로 일 양이 줄어듦.)

AJAX 특징

장점

- 페이지 이동 없이 고속으로 화면을 전환할 수 있다
- 서버 처리를 기다리지 않고 비동기 요청이 가능하다
- 수신하는 데이터 양을 줄일 수 있고, 클라이언트에게 처리를 위임할 수도 있다
- 향상된 사용자 경험을 제공한다



AJAX 특징

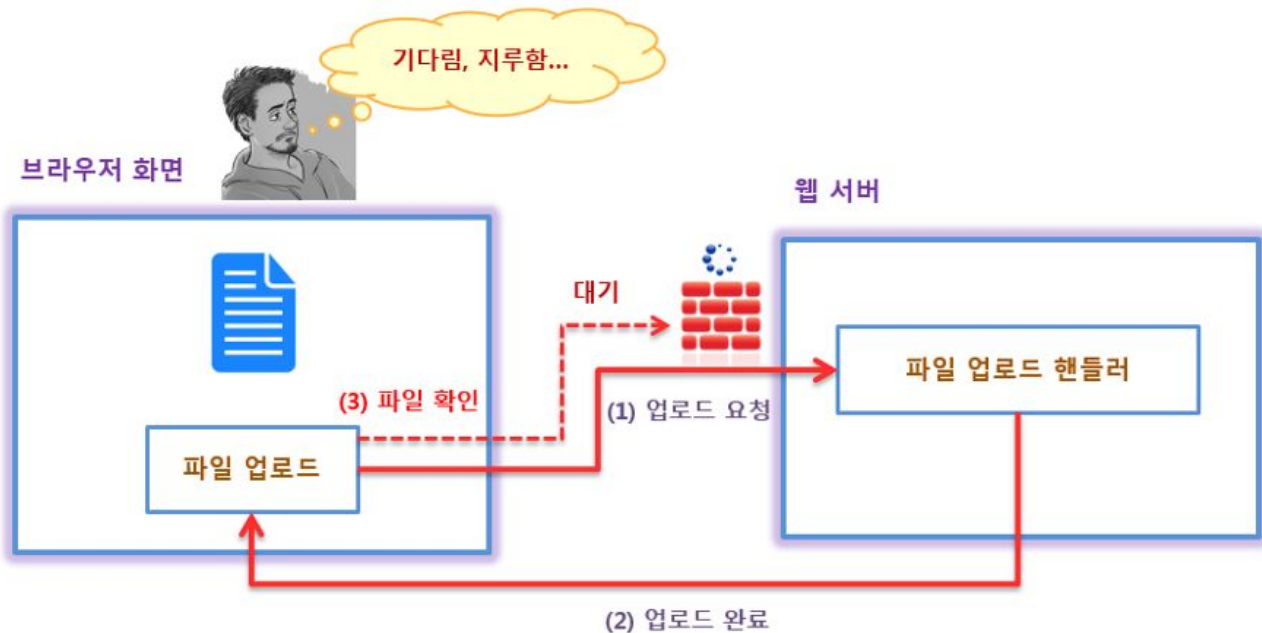
단점

- 브라우저 호환성 문제- 모든 브라우저가 동일한 방식으로 Ajax를 지원하지 않을 수 있다.
- SEO 문제- Ajax를 통해 동적으로 로드된 콘텐츠는 검색엔진에 의해 인덱싱되지 않을 수 있다.(검색엔진 크롤러가 초기 데이터만 수집하거나 JS를 완전하게 실행하지 못하는 경우가 생김)
- 지원하는 charact set이 한정되어 있다
- 페이지 이동 없는 통신으로 인해 보안상 문제가 생길 수 있다.
- 요청을 남발하면 역으로 서버 부하가 늘 수 있다

동기방식과 비동기방식의 차이

동기방식(파일업로드 등 파일 입출력 작업시)

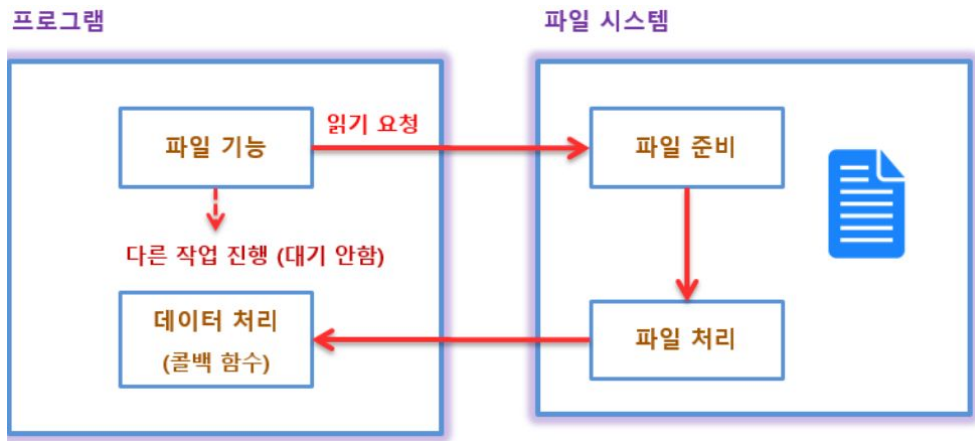
- 파일 처리를 다 하고 난 후에 다른 일을 처리할 수 있다.



동기방식과 비동기방식의 차이

비동기 방식

- 하나의 요청 처리가 끝날 때까지 기다리지 않고 다른 요청을 동시에 처리한다



비동기 방식(Asynchronous)이란?

비동기 방식의 의미

1. 백그라운드 작업:

비동기 방식에서는 특정 작업이 메인 스레드와 독립적으로 백그라운드에서 수행됩니다. 예를 들어, 웹 애플리케이션에서 서버로 데이터를 요청하는 동안, 사용자는 페이지와 계속 상호작용할 수 있습니다.

2. 순서 예측 불가:

여러 비동기 작업이 동시에 시작되면, 각 작업이 완료되는 순서를 예측하기 어렵습니다. 이는 각 작업의 처리 시간에 따라 달라지기 때문입니다. 비동기 방식은 이러한 순서 예측 불가성 때문에 결과를 비동기적으로 처리하는 콜백 함수나 프로미스(Promises), async/await 같은 구문을 사용하여 처리합니다.

3. 병렬 수행:

비동기 방식은 여러 작업을 병렬로 수행할 수 있게 합니다. 이는 하나의 작업이 완료될 때까지 다른 작업이 차단되지 않도록 하여 전체적인 성능과 반응성을 높입니다.

비동기 통신

순서 상관 없이
한 페이지에서
필요할 때마다
서버와 통신

비동기 방식의 특징

1. 페이지 리로드 없이 데이터 업데이트:

비동기 방식으로 서버에 요청을 보내고 응답을 받을 때, 전체 페이지를 새로고침하지 않고도 필요한 부분만 업데이트할 수 있습니다. 이는 사용자 경험을 개선하고 서버의 부하를 줄이는 데 도움이 됩니다.

2. 사용자 인터페이스의 향상:

서버에서 데이터를 가져오는 동안 사용자는 웹 페이지의 다른 부분과 상호 작용할 수 있습니다. 이는 웹 애플리케이션을 더 빠르고 반응성이 뛰어나게 만들어 줍니다.

3. 병렬 요청 처리:

여러 개의 비동기 요청을 동시에 보낼 수 있습니다. 각 요청은 독립적으로 처리되며, 하나의 요청이 완료될 때까지 다른 요청들이 차단되지 않습니다.

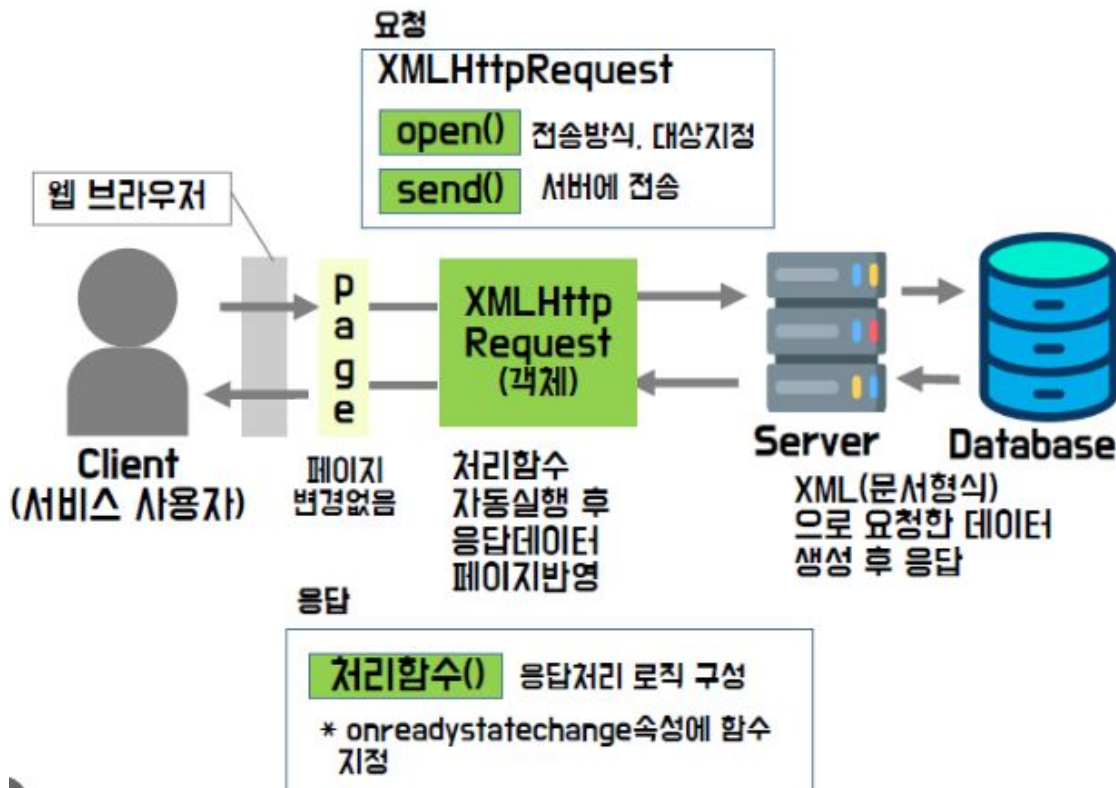
Ajax 동작 방식

javascript에서는

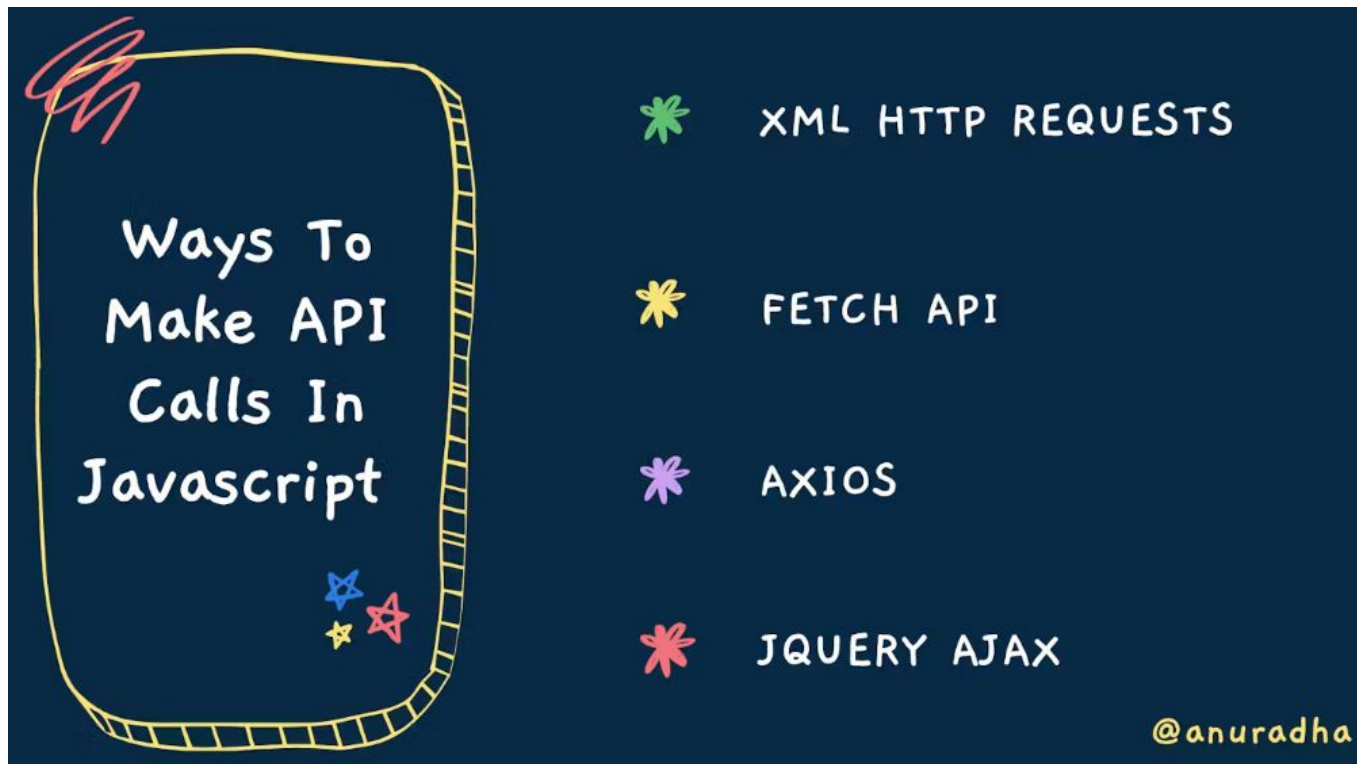
XMLHttpRequest 객체가
ajax를 수행하는 객체다.

fetch/Promise/Axios 등을
이용해서도

Ajax통신을 할 수 있다



Ajax 통신에 사용되는 API



XMLHttpRequest

vs Fetch

```
var request = new XMLHttpRequest();
request.open('GET',
'https://api.github.com/users/anuradha9712');
request.send();
request.onload = ()=>{
  console.log(JSON.parse(request.response));
}
```

- 모든 최신 브라우저에서 지원함
- 오래된 브라우저와 새 브라우저에서도 작동함
- ES6에서는 더이상 사용되지 않음

```
fetch('https://api.github.com/users/anuradha9712')
.then(response =>{
  return response.json();
}).then(data =>{
  console.log(data);
})
```

- fetch() 함수를 통해 좀 더 쉽게 비동기 통신을 할 수 있다
- fetch() 함수는 Promise를 반환하며, 이 객체는 response와 error를 포함하는 객체다.

Axios

vs

jQuery

```
<script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

axios.get('https://api.github.com/users/anuradha9712')
.then(response =>{
  console.log(response.data)
})
```

- 브라우저와 Node.js 모두에서 동작한다
- 외부 CDN을 사용하여 axios를 사용할 수 있다.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
$(document).ready(function(){
  $.ajax({
    url:
    "https://api.github.com/users/anuradha9712",
    type: "GET",
    success: function(result){ console.log(result); }
  })
})
```

- 역시 CDN방식으로 라이브러리 사용 가능
- \$.ajax() 메서드를 이용하여 요청을 보내고 응답을 받는다.

Promise란?

- JavaScript에서 비동기 작업을 처리하는 데 사용되는 객체로, 비동기 작업의 완료 또는 실패 상태를 나타내는 값을 가지고 있다.
- Promise는 세 가지 상태를 갖는다

Pending (대기 중): 초기 상태, 아직 결과를 얻지 못한 상태. Promise가 생성되었지만 아직 완료되지 않은 상태.

Fulfilled (이행됨): 작업이 성공적으로 완료된 상태. 'resolve'가 호출되면 이 상태가 된다

Rejected (거부됨): 작업이 실패한 상태. 'reject'가 호출되면 이 상태가 된다.

Promise란?

Promise의 메서드

then(): Promise가 이행(**fulfilled**) 상태가 되면 호출된다. 이 메서드는 성공적인 결과를 처리하는 콜백 함수를 받는다.

catch(): Promise가 **rejected** 상태가 되면 호출된다. 이 메서드는 오류를 처리하는 콜백 함수를 받는다.

finally(): Promise가 **fulfilled**든 **rejected**든 관계없이 항상 호출된다. 정리 작업에 유용하다

Promise 객체의 기본 구조

```
let promise = new Promise((resolve, reject) => {  
  // 비동기 작업 수행  
  if(성공) {  
    resolve(결과); // 작업이 성공했을 때 호출  
  } else {  
    reject(오류); // 작업이 실패했을 때 호출  
  }  
});
```

new Promise:

- `Promise`는 새로운 `Promise` 객체로, 두 개의 콜백 함수를 인자로 받습니다: `resolve`(성공)와 `reject`(실패).

Promise 객체의 기본 구조

then() 메서드:

- Promise가 **resolve** 상태가 되면 호출됩니다.

catch() 메서드:

- Promise가 **reject** 상태가 되면 호출됩니다.

finally() 메서드:

- Promise가 완료된 후(**fulfilled** 또는 **rejected**) 항상 실행됩니다.

- Promise를 사용하면 콜백 지옥을 줄여주고 **코드의 가독성을 높여준다**
- 또한 **catch()**를 **이용**하여 비동기 작업에서 발생하는 오류를 쉽게 처리할 수 있다
- 여러 작업을 순차적으로 수행할 때 **'then()'** 메서드를 **체인으로 연결**하여 구현할 수 있다
- 하지만 복잡한 흐름 제어가 필요한 경우에는 코드가 복잡해질 수 있다

Promise 사용 예제

```
<script>
function addNum(a, b) {
  return new Promise((resolve, reject) => {
    //1초후에 실행되는 타이머 설정
    setTimeout(() => {
      if (typeof a === 'number' && typeof b === 'number') {
        resolve(a + b);
      } else {
        reject('Error: 두 인수는 숫자여야 해요');
      }
    }, 1000);
  });
}
```

```
addNum(2, 3)
  .then((result) => alert(result))
  .catch((error) => alert(error));
```

이 경우는 then()을 수행

```
addNum(2, '3')
  .then((result) => alert(result))
  .catch((error) => alert(error));
```

이 경우는 catch()를 수행한다

```
</script>
```

Promise와 XMLHttpRequest를 사용한 통신 예제

```
8 <meta name="Description" content="">
9 <title>Document</title>
10 </head>
11 <body>
12 <script>
13   function sendData() {
14     const data = {
15       name: '홍길동',
16       age: 30
17     };
18     // Promise를 사용하여 XMLHttpRequest 요청 처리
19     const promise = new Promise((resolve, reject) => {
20       const xhr = new XMLHttpRequest();
21       xhr.open('POST', 'http://localhost:3000/data');
22       xhr.setRequestHeader('Content-Type', 'application/json');
23
24       xhr.onload = () => {
25         if (xhr.status >= 200 && xhr.status < 300) {
26           resolve(JSON.parse(xhr.responseText)); // 성공적으로 응답을 받은 경우
27         } else {
28           reject(new Error(`요청 실패: ${xhr.status}`)); // 상태 코드가 2xx가 아닌 경우
29         }
30       };
31
32       xhr.onerror = () => {
33         reject(new Error('요청 중 오류 발생')); // 네트워크 오류 등의 문제가 발생한 경우
34       };
35
36       xhr.send(JSON.stringify(data)); // JSON 형식으로 데이터 전송
37     });
38   }
```

```
38
39   promise
40     .then(data => {
41       console.log('성공:', data); // 성공적으로 응답을 받은 경우 처리
42     })
43     .catch(error => {
44       console.error('오류 발생:', error); // 오류가 발생한 경우 처리
45     });
46 }
47
48 // 함수 호출
49 sendData();
50
51 </script>
52
53 </body>
54 </html>
```

Fetch API 사용한 통신

fetch api를 사용하면 ajax통신이 가능하며, 다른 라이브러리를 설치할 필요가 없다.

- fetch 특징
- Promise를 지원한다
- 요청과 응답을 Request와 Response객체로 캡슐화하여 제공한다
- CORS 지원: Cross-Origin Resource Sharing을 지원하여 도메인간 요청을 쉽게 처리할 수 있다.
- 스트리밍: 대용량 데이터를 효율적으로 처리할 수 있다

Fetch API 기본 문법

- Fetch API는 `fetch()` 함수를 사용해서 HTTP 통신을 할 수 있다.

```
fetch(url [, options])  
  .then(response => {  
    // 응답을 처리하는 로직  
    return response.json(); // 또는 response.text(), response.blob(), response.formData(),  
  })  
  .then(data => {  
    // 응답 데이터를 활용하는 로직  
    console.log(data);  
  })  
  .catch(error => {  
    // 에러 처리 로직  
    console.error('Error', error);  
  });
```

옵션으로 `method`, `body`, `headers` 등의 속성을 지정할 수 있다

`fetch()` 함수 요청에 대한 응답은 `Promise`로 반환된다. 따라서 비동기 데이터 처리를 위해 `then()`을 사용하고 에러 처리를 위해서는 `catch()` 함수를 사용한다

fetch - GET방식 요청

fetch(url) 메서드는 응답을 포함하는 프로미스를 반환한다. 응답의 json본문을 읽기 위해서는 json() 메서드를 사용하면 된다.

```
fetch('https://jsonplaceholder.typicode.com/posts')  
  .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 has been a problem with your fetch operation:', error));
```

1. Response 객체: `fetch`가 반환하는 Promise는 `Response` 객체를 반환합니다. 이 객체를 사용하여 상태 코드, 헤더, 본문 등의 정보를 얻을 수 있습니다.
2. `.json()`, `.text()`, `.blob()` 등: `Response` 객체에는 다양한 형태로 응답 본문을 변환할 수 있는 메서드가 있습니다. 예를 들어 `.json()`은 JSON 형태로 변환합니다.
3. 에러 처리: 네트워크 에러는 `catch` 블록에서 처리하고, 응답 자체의 문제(예: 404)는 `response.ok`를 사용하여 처리할 수 있습니다.

fetch - POST방식 요청

post는 데이터를 추가하는데 사용한다. `fetch()`함수로 요청할 때 추가할 데이터를 포함해서 전달하는데 이 때 `body` 속성을 사용하며, json형태 데이터를 포함하는 경우 `body`속성에 json형태의 데이터로 지정한다

POST방식의 요청은

첫번째 인수는 URL주소,

두번째 인수는 요청에 대한 설정을 정의한 객체가 들어간다.

- method**: 요청방식
- headers**: 헤더 설정(보내는 데이터 컨텐타입설정 등)
- body**: 요청 데이터를 json문자열 형태로 전달한다

①, url

```
fetch('https://jsonplaceholder.typicode.com/posts', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify({  
    title: 'foo',  
    body: 'bar',  
    userId: 1  
  })  
})  
  .then(response => response.json())  
  .then(data => console.log(data))  
  .catch(error =>  
    console.error('There has been a problem with your fetch operation:', error));
```

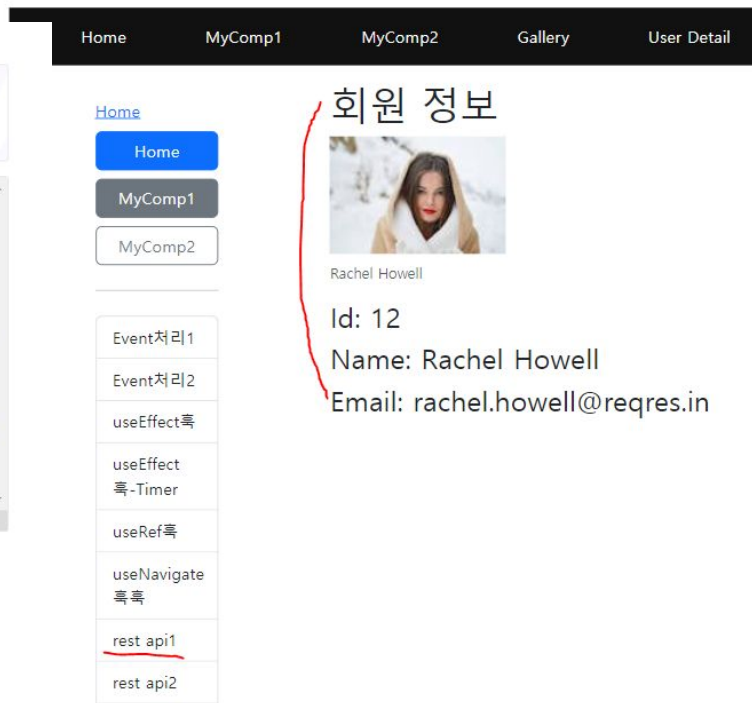
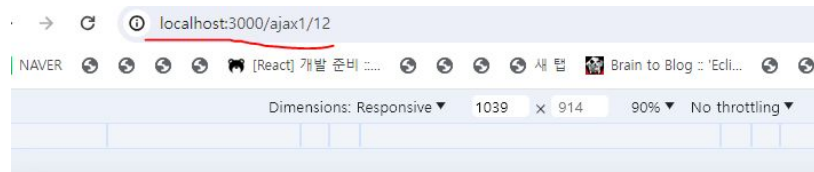
②

fetch API 사용 실습1

reqres.in 사이트를 이용해 데이터 받아오기

GET	LIST USERS
GET	<u>SINGLE USER</u>
GET	SINGLE USER NOT FOUND
GET	LIST <RESOURCE>
GET	SINGLE <RESOURCE>
GET	SINGLE <RESOURCE> NOT FOUND

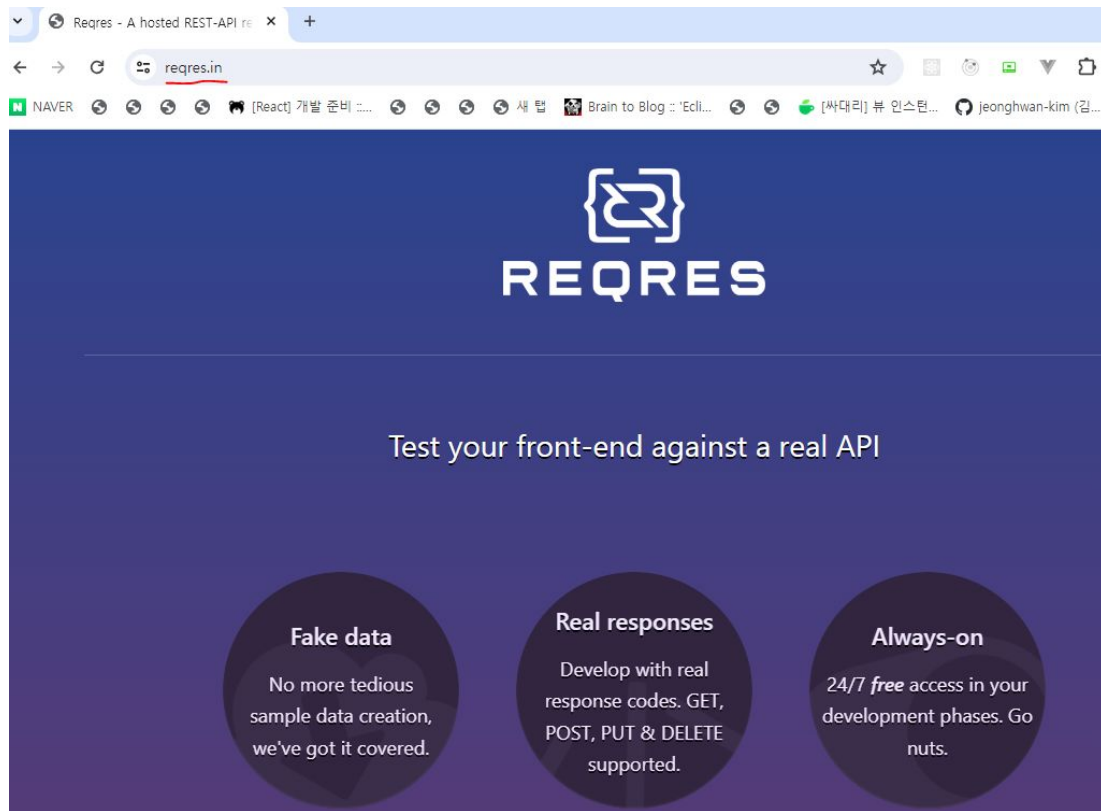
Request	Response
<u>/api/users/2</u>	200
	<pre>{ "data": { "id": 2, "email": "janet.weaver@reqres.in", "first_name": "Janet", "last_name": "Weaver", "avatar": "https://reqres.in/img/fac" }, "support": { "url": "https://reqres.in/#support-h", "text": "To keep ReqRes free, contri" } }</pre>



Rest API 테스트 사이트

<https://reqres.in/>

백엔드를 구현해서 연동해보면
Best이지만,
당장 백엔드 구현이 어려우므로
우선 reqres.in 사이트에 접속
하여 네트워크 연동을 해보자.



requires.in 사이트 제공 서비스

Reqres.in은 테스트와 프로토타이핑을 위해 설계된 다양한 모의 API 서비스를 제공한다

1. 사용자 관리:

- 사용자 생성: ``POST /api/users`` 엔드포인트를 사용하여 새로운 사용자를 생성하는 것을 시뮬레이션할 수 있습니다.
- 사용자 조회: ``GET /api/users`` 및 ``GET /api/users/:id``를 사용하여 단일 또는 여러 사용자 정보를 가져올 수 있습니다.
- 사용자 업데이트: ``PUT /api/users/:id`` 또는 ``PATCH /api/users/:id``를 사용하여 사용자 정보를 업데이트하는 것을 시뮬레이션할 수 있습니다.
- 사용자 삭제: ``DELETE /api/users/:id``를 사용하여 사용자를 삭제하는 것을 시뮬레이션할 수 있습니다.

requires.in 사이트 제공 서비스

2. 인증:

- 로그인: ``POST /api/login`` 엔드포인트를 사용하여 로그인을 시뮬레이션할 수 있습니다.
- 등록: ``POST /api/register``를 사용하여 사용자 등록을 시뮬레이션할 수 있습니다.

3. 리소스 관리:

- 리소스 목록: ``GET /api/unknown``을 사용하여 리소스 목록을 가져올 수 있습니다.
- 단일 리소스: ``GET /api/unknown/:id``를 사용하여 단일 리소스의 세부 정보를 조회할 수 있습니다.

4. 지연된 응답:

- ``GET /api/users?delay=3``을 사용하여 응답 지연을 시뮬레이션하고, 애플리케이션이 지연된 응답을 어떻게 처리하는지 테스트할 수 있습니다.

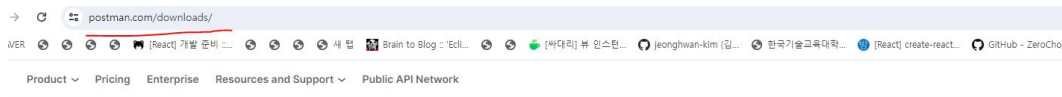
이러한 모의 서비스는 실 서버 없이도 프론트엔드나 백엔드 애플리케이션을 테스트하려는 개발자에게 유용합니다. 모의 데이터와 엔드포인트는 실제 시나리오를 반영하도록 설정되어 있어 다양한 HTTP 요청과 응답을 애플리케이션이 어떻게 처리하는지 테스트하기 쉽게 합니다.

더 자세한 정보와 이러한 엔드포인트를 사용해보려면 [Postman 워크스페이스를 방문](#)할 수 있습니다 (Postman) (Postman).

postman desktop설치

<https://www.postman.com/>

<https://www.postman.com/downloads/>



Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

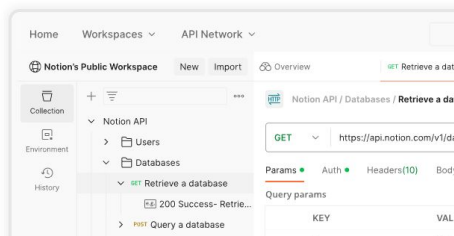
The Postman app

Download the app to get started with the Postman API Platform.

Windows 64-bit

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

[Release Notes](#) ->



Mac에서 postman desktop설치 참조 사이트

<https://pink1016.tistory.com/187>

fetch API 사용 실습1 - Single User정보 가져오기

```
let btn2 = document.querySelector('#btn2');
btn2.onclick = () => {
  const findId = prompt('검색할 회원의 id번호를 입력하세요');
  if (!findId) return;
  const url = `https://reqres.in/api/users/${findId}`;
  fetch(url)
    .then((response) => {
      if (!response.ok) throw new Error('데이터가 없거나 Network 응답에 문제가 있어요');
      return response.json();
    })
    .then((data) => {
      let tmpUser = data.data;
      // alert(JSON.stringify(tmpUser));
      showUser(tmpUser);
    })
    .catch((err) => {
      document.getElementById(
        'result'
      ).innerHTML = `<h3 class='err'>${findId}님 데이터를 찾을 수 없거나 서버 네트워크 연결에 문제가 있습니다</h3>`;
    });
};

function showUser(data) {
  const { id, email, first_name, last_name, avatar } = data;
  // alert(first_name);
  document.getElementById('result').innerHTML = `
    <h2>회원 정보</h2>
    <img src='${avatar}'>
    <h3>ID: ${id} </h3>
    <h3>Name: ${first_name} ${last_name}</h3>
    <h3>Email: ${email}</h3>
  `;
}
```

Windows 정품 인증

Fetch API 활용

User정보 가져오기1

User정보 가져오기2- reqres.in에서 받아오기

회원 정보



ID: 5

Name: Charles Morris

Email: charles.morris@reqres.in

실습2-모든 Users 가져오기

1. 모든 Users 정보 12명을

출력해보기

2. 페이징 처리하여

한 페이지에 3명씩

보여주기

HomeMyComp1MyComp2GalleryUser DetailGo to

Home

MyComp1

MyComp2

Event처리1

Event처리2

useEffect훅

useEffect훅-Timer

useRef훅


useNavigate훅훅

rest api1


rest api2

Vestibulum at eros


All Users



George Bluth [george.bluth@reqres.in]



Janet Weaver [janet.weaver@reqres.in]



Emma Wong [emma.wong@reqres.in]

1234

axios 사용한 통신

- axios 라이브러리를 사요하면 `fetch()`에서 설정정보로 넣어줘야할 사항들을 간단히 설정할 수 있어 편리하자. 다만 **axios라이브러리를 설치**해야 한다.
- Node.js에서 사용할 때는 `npm install axios` 로 설치
- 브라우저에서 사용할 때는 `cdn`방식으로 참조해보자.

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

axios 사용법

- 기본 형식

- `axios(config)`
- `.then(res=>console.log(res.data))`
- `.catch(err=>console.log(err))`

- 옵션(config)

`config`는 `axios()`를 호출할 때 사용하는 여러 옵션을 설정하는 객체.

- `method`, `data`, `headers` 등 다양한 속성을 사용할 수 있다

```
1 axios({  
2   method: 'get',  
3   url: '/api/users'  
4 })  
5 .then(response=>{})  
6 .catch(error=>{})
```

axios POST/PUT/DELETE

```
1 axios({
2   method: 'post',
3   url: '/api/users',
4   data: {name: 'Tom', job: 'Manager'}, //json형태로 자동 변환된다
5 })
6 .then(response=>{})
7 .catch(error=>{})
```

```
1 axios({
2   method: 'put',
3   url: '/api/users/1',
4   data: {name: 'Tom', job: 'Manager'},
5 })
6 .then(response=>{})
7 .catch(error=>{})
```

```
1 axios({
2   method: 'delete',
3   url: '/api/users/1',
4 })
5 .then(response=>{})
6 .catch(error=>{})
```

axios 단축 메서드(get()/post()...) 사용법

- GET방식 요청일 경우

```
- axios.get(url[,config])  
- .then(res=>console.log(res.data))  
- .catch(err=>console.log(err))
```

- POST방식 요청일 경우

```
- axios.post(url[, body[,config]])  
  .then(res=>console.log(res.data))  
  .catch(err=>console.log(err))
```

get(),post()외에도 put(),patch(),delete(), head() 등의 단축 메서드가 있다

fetch와 axios 차이점

- 우선 GET, POST와 같은 메소드를 사용할 때, Axios에서는 `axios.get()`, `axios.post()` 같은 함수를 사용한다.
- JSON 형식의 응답 데이터를 가져올 때 `fetch`에서는 `await` 키워드를 사용해야 하지만,
- `axios`에서는 그럴 필요가 없다.

[fetch]

```
async function getBoard() {  
  const res = await fetch('/board/');  
  const board = await res.json();  
  return board;  
}
```

[axios]

```
async function getBoard() {  
  const res = await axios.get('/board/');  
  const board = res.data;  
  return board;  
}
```

fetch와 axios 차이점

- POST방식

```
[fetch]
async function insertBoard(body) {
  const res = await fetch(
    '/board',
    {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(body),
    }
  );

  const data = await res.json();
  return data;
}
```

```
[axios]

async function insertBoard(body) {
  const res = await axios.post(
    '/board',
    body
  );

  const data = res.data;
  return data;
}
```

fetch와 axios 차이점

- Fetch에서는 쿼리 문자열을 직접 만들어서 넣는다
- Axios에서는 **params**라는 속성의 객체로 전달한다

[fetch]




```
async function getBoard({ offset = 0, limit = 10 }) {  
  const query = new URLSearchParams({ offset, limit });  
  const url = `/board?${query.toString()}`;  
  // query를 URL에 추가  
  
  const res = await fetch(url);  
  const board = await res.json();  
  return board;  
}
```

[axios]














```
async function getBoard({ start= 0, size= 5}) {  
  const res = await axios.get('/board/',  
    {  
      params: { start, size },  
    });  
  const board = res.data;  
  return board;  
}
```


axios 실습 1 - 한국닷컴 사이트 RSS

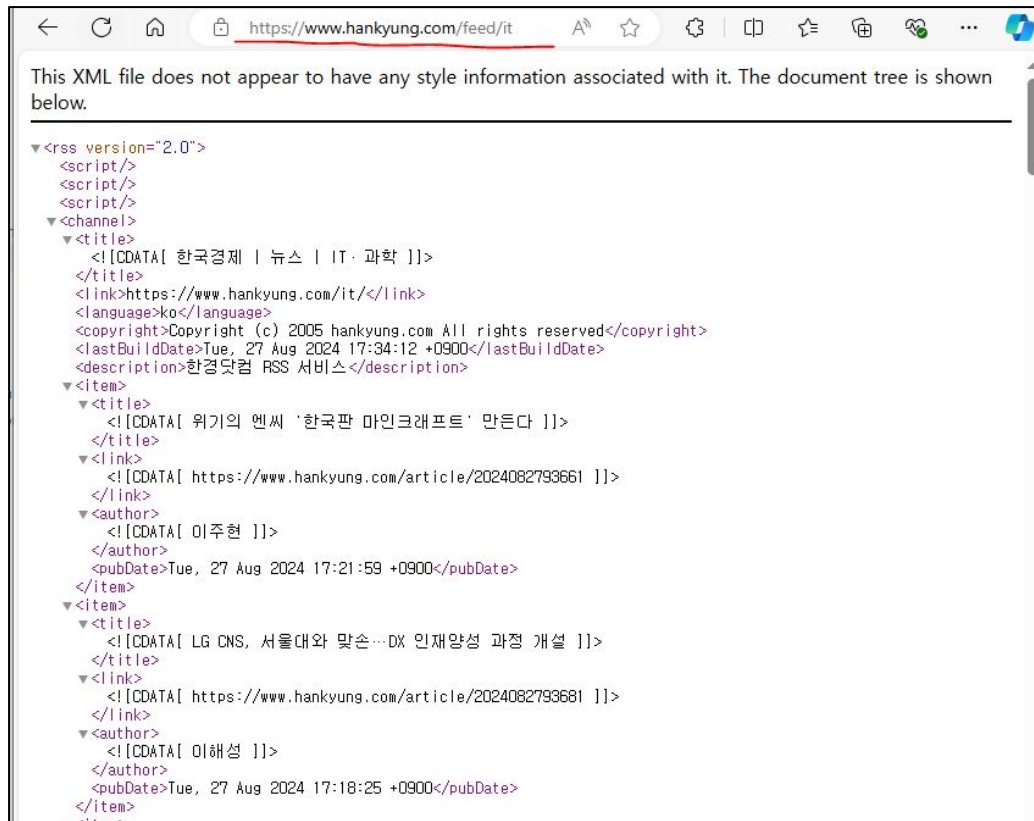
[RSS | 한경닷컴 \(hankyung.com\)](#)

 <https://www.hankyung.com/feed>  

한국경제 RSS

전체뉴스 https://www.hankyung.com/feed/all-news		증권 https://www.hankyung.com/feed/finance	
경제 https://www.hankyung.com/feed/economy		부동산 https://www.hankyung.com/feed/realestate	
IT https://www.hankyung.com/feed/it		정치 https://www.hankyung.com/feed/politics	
국제 https://www.hankyung.com/feed/international		사회 https://www.hankyung.com/feed/society	
생활 https://www.hankyung.com/feed/life		오피니언 https://www.hankyung.com/feed/opinion	
스포츠 https://www.hankyung.com/feed/sports		연예 https://www.hankyung.com/feed/entertainment	
VIDEO https://www.hankyung.com/feed/video			

axios 사용하여 환경 RSS 연결



Axios라이브러리를 이용한 통신

한경닷컴 RSS뉴스 데이터 받기

XML 데이터 받아오기1-cors문제 발생

XML 데이터 받아오기2-proxy서버 통해 통신

JSON 데이터 받아오기

RSS란?

- **RSS**(Really Simple Syndication 또는 Rich Site Summary)는 **웹사이트의 업데이트 내용을 쉽고 빠르게 사용자에게 제공하기 위한 표준 형식의 XML 기반 문서**다.
- RSS는 주로 뉴스, 블로그, 팟캐스트와 같은 웹사이트에서 새로운 콘텐츠를 자동으로 사용자에게 전달하기 위해 사용된다.

RSS 피드 구조

<code><rss></code>	: 루트 요소이며, RSS 피드가 시작되는 곳을 나타낸다.
<code><channel></code>	: 하나의 피드를 나타내며, 피드의 메타데이터와 여러 개의 아이템을 포함한다.
<code><title></code>	: 피드 또는 항목의 제목.
<code><link></code>	: 피드나 특정 항목에 대한 URL .
<code><description></code>	: 피드나 항목에 대한 간단한 설명.
<code><item></code>	: 피드에서 제공하는 각 개별 콘텐츠 항목을 나타내며, 제목, 링크, 설명 등의 요소를 가진다.

CORS 문제 발생

axios1.html > html > body > script

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>Document</title>
7   <!-- ==axios cdn===== -->
8   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
9   <!-- ===== -->
10 <style>
11   body {
12     padding: 1em;
13     text-align: center;
14   }
15 </style>
16 </head>
17 <body>
18   <h1>Axios라이브러리를 이용한 통신</h1>
19   <h2>한경닷컴 RSS뉴스 데이터 받기</h2>
20   <button id="btn1">데이터 받아오기1</button>
21   <button id="btn2">데이터 받아오기2</button>
22   <hr color="red" />
23 <script>
24   const bt1 = document.getElementById('btn1');
25   const bt2 = document.getElementById('btn2');
26   const url = 'https://www.hankyung.com/feed/it';
27   bt1.onclick = () => {
28     axios
29       .get(url)
30       .then((response) => alert(response.data))
31       .catch((err) => alert(err));
32   };
33
```

127.0.0.1:5500 내용:

AxiosError: Network Error

확인

데이터 받아오기1-cors문제 발생

데이터 받아오기2-proxy서버 통해 통신

CORS란?

- Cross Origin Resource Sharing
- 웹 애플리케이션에서 다른 도메인에 있는 리소스에 접근하고자 할 때 발생하는 보안기능을 의미한다
- 기본적으로 웹 브라우저는 보안상의 이유로 다른 출처의 리소스에 대한 요청을 제한한다.
- 이를 "동일 출처 정책(Same-Origin Policy)"이라고 하며, 이는 악의적인 사이트가 사용자의 데이터를 훔치거나 조작하지 못하도록 보호하기 위한 것
- CORS는 이러한 제한을 완화하여, 웹 페이지가 자신과 다른 출처의 리소스에 안전하게 접근할 수 있도록 하는 메커니즘이다.
- 서버가 특정 출처에서의 요청을 허용할 수 있도록 응답 헤더에 관련 정보를 포함하여 브라우저에

서버가 CORS를 허용하는 HTTP 응답의 예

이 응답에서는 <https://example.com>에서 온 요청만 허용하며, GET과 POST 메서드를 사용한 요청이 허용됩니다. 또한, Content-Type과 Authorization 헤더가 요청에 포함될 수 있으며, 자격 증명(쿠키 등)도 포함될 수 있습니다.

HTTP/1.1 200 OK

Access-Control-Allow-Origin: https://example.com

Access-Control-Allow-Methods: GET, POST

Access-Control-Allow-Headers: Content-Type,
Authorization

Access-Control-Allow-Credentials: true

서버측에서 cors관련 헤더 설정

CORS 관련 주요 헤더

- **Access-Control-Allow-Origin:** 서버가 허용하는 출처를 지정합니다. `*`을 사용하면 모든 출처를 허용합니다.
- **Access-Control-Allow-Methods:** 서버가 허용하는 HTTP 메서드를 지정합니다(예: `GET, POST, PUT`).
- **Access-Control-Allow-Headers:** 클라이언트가 요청에서 사용할 수 있는 헤더를 지정합니다.
- **Access-Control-Allow-Credentials:** 클라이언트의 자격 증명(예: 쿠키)을 포함할 수 있는지 여부를 지정합니다.
- **Access-Control-Max-Age:** 사전 요청의 결과를 캐시할 수 있는 시간(초)을 지정합니다.

CORS에러 해결하기

- [1] 백엔드와 [2] 프론트엔드에서 각각 해결하는 방법이 있다
- [1] 서버에서 해결하기

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors()); // 모든 출처에 대해 CORS를 허용

app.get('/api/data', (req, res) => {
  res.json({ message: 'Hello, CORS!' });
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

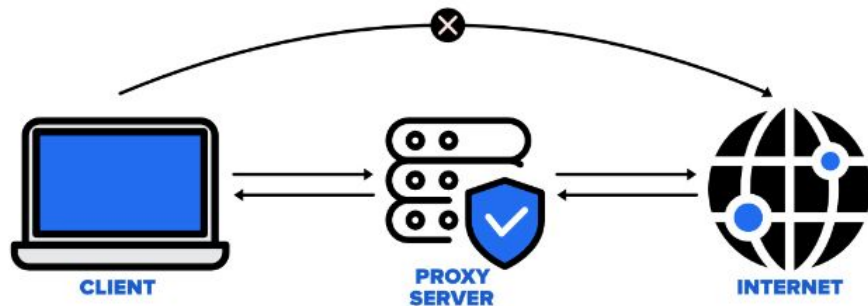
가장 일반적이고 권장되는 방법은 서버 측에서 올바른 CORS 헤더를 설정하는 것.

서버는 응답에 **Access-Control-Allow-Origin** 헤더를 추가하여 어떤 출처에서 요청을 허용할지를 명시할 수 있다.

Node.js와 Express를 사용하여 서버를 운영하고 있다면, **cors** 미들웨어를 사용하여 쉽게 CORS 문제를 해결할 수 있다

왼쪽 코드에서 **app.use(cors())**를 사용하면 모든 출처에서의 요청이 허용됩니다. 특정 출처만 허용하려면 **cors** 함수에 옵션을 전달할 수 있습니다:

CORS에러 해결하기



[2] 클라이언트에서 해결하기

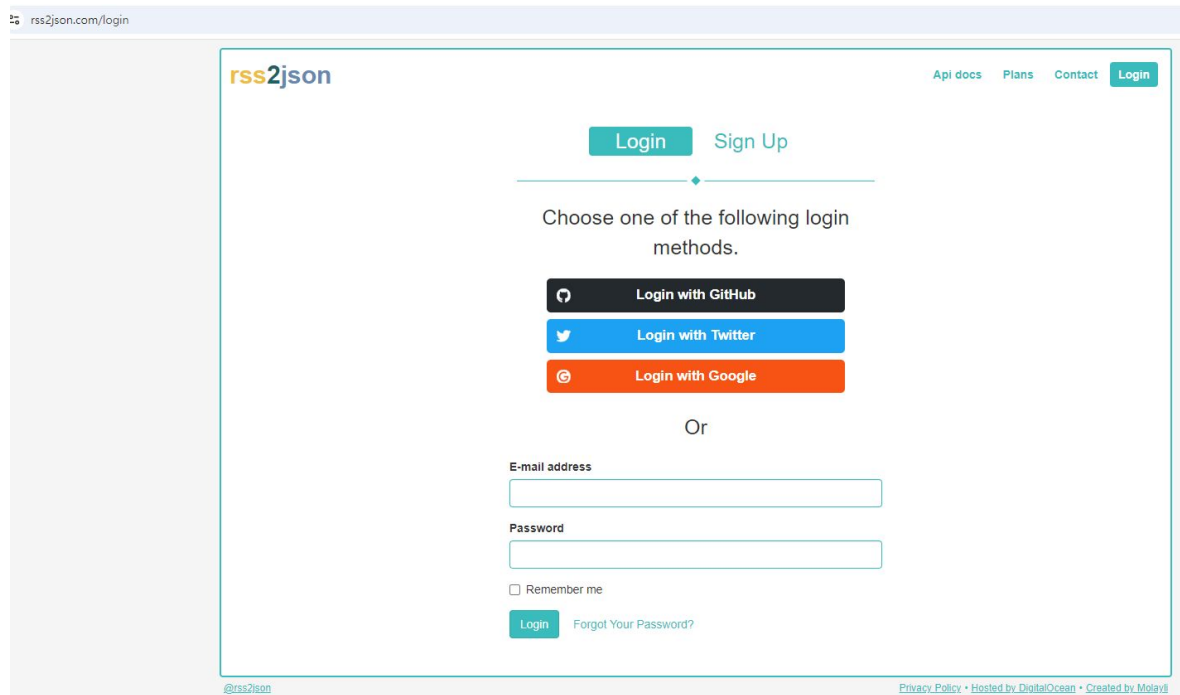
- 프론트엔드와 백엔드가 다른 도메인에서 운영되는 경우, **프록시 (PROXY) 서버를 설정**하여 CORS 문제를 피할 수 있다.
- **프록시**는 클라이언트와 실제 서버 사이에서 **중계 역할**을 하며, **같은 도메인에서 요청이 이루어지도록 보이게 만든다**
- 프록시 서버는 보통 클라이언트와 동일한 출처에서 생성한다.
- 클라이언트는 프록시 서버로 요청을 보낸다.
- 프록시 서버는 API로 서버에 요청하고 응답받은 뒤 이를 다시 클라이언트에 응답한다.

proxy server



rss2json사이트 회원가입

<https://rss2json.com/login>



The screenshot shows the login page of the rss2json website. The browser's address bar displays 'rss2json.com/login'. The page features the 'rss2json' logo in the top left corner. In the top right corner, there are links for 'Api docs', 'Plans', 'Contact', and a 'Login' button. The main content area has two buttons: 'Login' and 'Sign Up'. Below these, a horizontal line with a diamond in the center separates the buttons from the text 'Choose one of the following login methods.' Underneath this text are three social login buttons: 'Login with GitHub' (dark grey), 'Login with Twitter' (blue), and 'Login with Google' (orange). Below these is the word 'Or'. Further down are two input fields labeled 'E-mail address' and 'Password'. Below the password field is a checkbox labeled 'Remember me'. At the bottom of the form area are a 'Login' button and a link for 'Forgot Your Password?'. The footer of the page includes the Twitter handle '@rss2json' on the left and links for 'Privacy Policy', 'Hosted by DigitalOcean', and 'Created by Molay' on the right.

rss2json.com/login

rss2json

Api docs Plans Contact Login

Login Sign Up

Choose one of the following login methods.

Login with GitHub

Login with Twitter

Login with Google

Or

E-mail address

Password

☐ Remember me

Login Forgot Your Password?

@rss2json

Privacy Policy • Hosted by DigitalOcean • Created by Molay

Api key 발급받기

rss2json.com/me/api_key

rss2json

My Account

Api docs

Plans

Contact

Logout

Overview

My Feeds

Stats

API Key

Api key management

Api Key

seblckldzbz

Regenerate

Api key를 복사해두자

Api restrictions

To prevent unauthorized use and quota theft, restrict your api key. Api key restriction lets you specify which web sites or IP addresses can use this api key.

☒ None

☐ HTTP referrers

☐ IP addresses

Save

rss 주소 넣고 convert to json버튼 클릭

이때 api key를

파라미터로 넣어

주자

=>응답이 오면

api call에 출력된

주소를 복사한다

[My Account](#) [Api docs](#) [Plans](#) [Contact](#) [Logout](#)

rss to json online converter

Enter the URL to your RSS feed

[Advanced options](#)

↓ Convert to JSON ↓

API call : `https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Fwww.hankyung.com%2Ffeed%2Fit&api_key=sebidkidzbzqgk...`

```
{
  "status": "ok",
  "feed": {
    "url": "https://www.hankyung.com/feed/it",
    "title": "한국경제 | 뉴스 | IT·과학",
    "link": "https://www.hankyung.com/it/",
    "author": "",
    "description": "한경닷컴 RSS 서비스",
    "image": ""
  },
  "items": [
    {
      "title": "엔비디아 GPU, 2030년 차세대 반도체로 대체",
      "pubDate": "2024-08-27 09:18:29",
      "link": "https://www.hankyung.com/article/2024082795191",
      "guid": "edf28f6cf75d3951b926aa1c210602b9",
      "author": "이해설",
      "thumbnail": "",
      "description": "",
      "content": "",
      "enclosure": {
      },
      "categories": [

```

proxy server 통해 데이터 받아오기

```
<body>
  <h1>Axios라이브러리를 이용한 통신</h1>
  <h2>한경닷컴 RSS뉴스 데이터 받기</h2>
  <button id="btn1">데이터 받아오기1-cors문제 발생</button>
  <button id="btn2">데이터 받아오기2-proxy서버 통해 통신</button>
  <hr color="red" />
  <script>
    const bt1 = document.getElementById('btn1');
    const bt2 = document.getElementById('btn2');
    const url = `https://www.hankyung.com/feed/it`;
    const url_proxy = `https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Fwww.hankyung.com%2Ffeed%2Fit&api_key=seblkdld`;
    bt1.onclick = () => {
      //axios.defaults.headers.common['Access-Control-Allow-Origin'] = '*'; ==> 이거 안된다
      axios
        .get(url)
        .then((response) => alert(response.data))
        .catch((err) => alert(err));
    };
    bt2.onclick = () => {
      axios
        .get(url_proxy)
        .then((response) => alert(response.data))
        .catch((err) => alert(err));
    };
  </script>
```



```
const url_proxy = `https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Fwww.hankyung.com%2Ffeed%2Fit&api_key=seblldt2`;
bt2.onclick = () => {
  axios
    .get(url_proxy, {
      params: { count: 20 }, //파라미터 데이터
    })
    .then((response) => {
      // alert(response.data)
      showNews(response.data);
    })
    .catch((err) => alert(err));
};

const showNews = (data) => {
  console.log(data);
  const list = data.items;
  //alert(list.length);
  // map() 함수를 이용하여 배열을 HTML li 요소로 변환
  const listItems = list.map((item) => {
    return `<li>${item.title}</li>`; // 여기서 item.title은 각 뉴스의 제목을 나타냄
  });

  // HTML 요소에 listItems를 추가하여 화면에 표시
  const newsContainer = document.getElementById('newsContainer'); // <ul> 태그를 가진 요소
  newsContainer.innerHTML = `<ul class='newsList'>`;
  newsContainer.innerHTML += listItems.join(''); // 배열을 문자열로 변환하여 innerHTML에 추가
  newsContainer.innerHTML += `</ul>`;
};
```

</script>

</body>

style 입히기

Axios라이브러리를 이용한 통신

한경닷컴 RSS뉴스 데이터 받기

데이터 받아오기1-cors문제 발생

데이터 받아오기2-proxy서버 통해 통신

"엔비디아 GPU, 2030년 차세대 반도체로 대체"

위기의 엔씨 '한국판 SF 오픈월드 대작' 만든다

LG CNS, 서울대와 맞손...DX 인재양성 과정 개설

카카오, 비수도권 대학생 돕는다...서비스 기획 지원

"전 세계 최초"...구글, 2년 연속 韓서 AI 연구 사례 공유

멜론 음악 채팅 서비스 '뮤직웨이브'...등접자 1만명 돌파

LG헬로비전, 지역 특화 상품으로 농가 수출 일궜다

엔젠바이오, 결핵 진단기기로 '해외수출' 목표 국가사업 선정