

# HICC 프론트엔드 세미나. 7주차

## JavaScript(3)

통신

By. 최서연

# Today's study

HTTP  
REST API  
ajax & form  
jQuery & AJAX  
Fetch API  
Token  
실습

# 통신



# HTTP란?

W3 상에서 정보를 주고받을 수 있는 프로토콜  
웹 브라우저가 웹 서버로부터 HTML 문서, 이미지, 비디오 등 다양한  
리소스를 요청하고 이를 받아오는 과정을 정의

\* HTTPS: HTTP 프로토콜의 보안 버전

# HTTP 요청과 응답 구조

## HTTP Request Message

**GET /test.html HTTP/1.1**

Host: google.com

Accept: text/html

Accept-Encoding: gzip, deflate

Connection: keep-alive

hl=ko&ogbl=0&page=99

**start line**

**headers**

**blank line**

**body**

# 주요 HTTP 메서드

- GET: 정보를 요청하기 위해 사용. 요청 본문이 없으며 데이터는 URL에 포함.

ex. /user/1은 user 중 user id=1인 데이터를 요청한다는 뜻

- POST: 정보를 밀어넣기 위해 사용. 요청 본문에 데이터를 포함.
- PUT: 정보를 업데이트하기 위해 사용. 요청 본문에 데이터를 포함
- DELETE: 정보를 삭제하기 위해 사용. 데이터는 URL에 포함시킨다.

# HTTP 요청과 응답 구조

## HTTP Response Message

**HTTP/1.1 200 OK**

Date: Sun, 26 June ...

Server: Apache

Content-Length: 35

Content-Type: text/html

**<h1>Hello World</h1>**

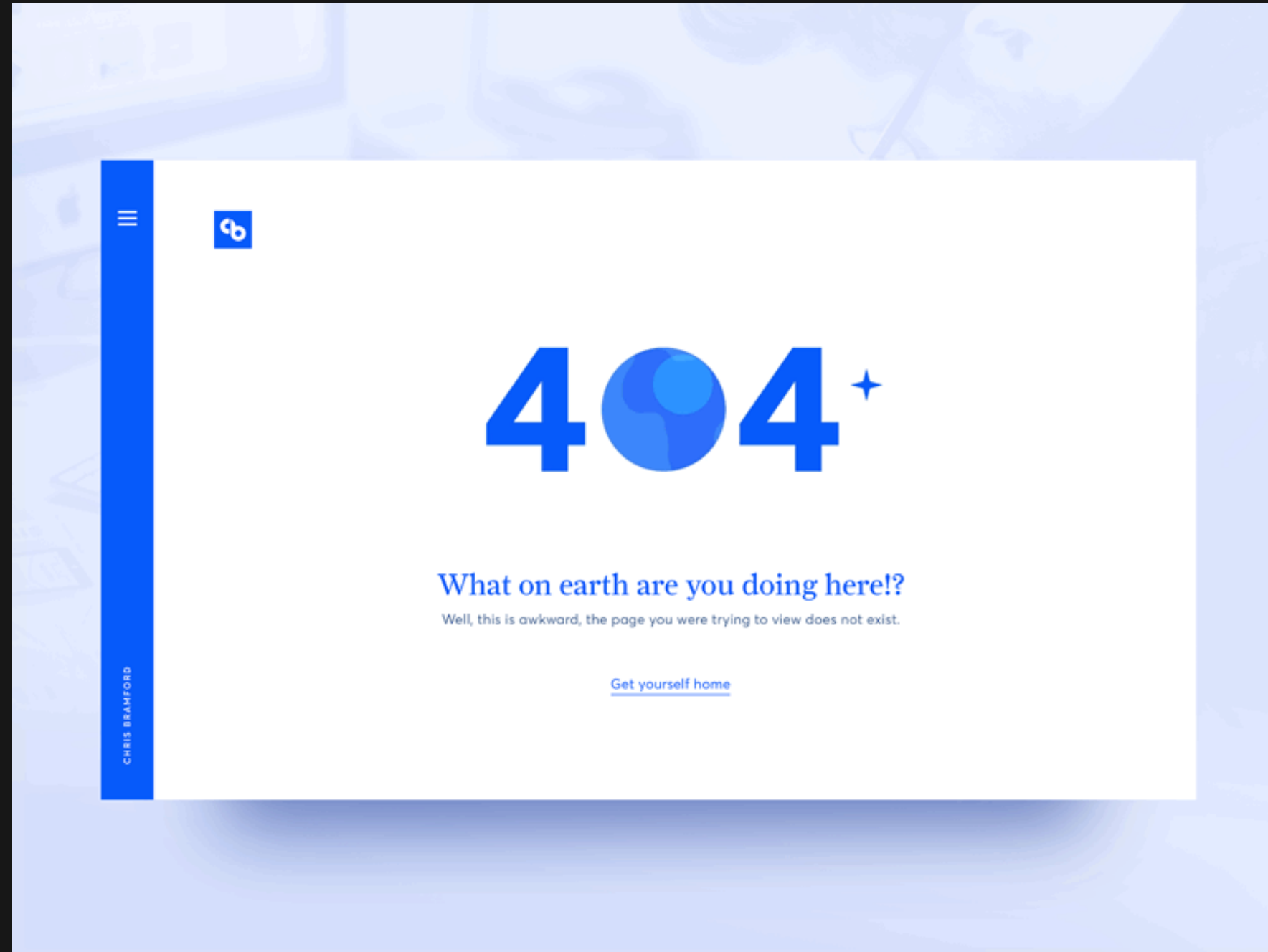
**status line**

**headers**

**blank line**

**body**

# HTTP 상태 코드





# HTTP 상태 코드

특정 HTTP 요청이 성공적으로 완료되었는지 알려줌

200 OK: 요청 성공

301 Moved Permanently: 리소스가 영구적으로 이동됨

400 Bad Request: 잘못된 요청

404 Not Found: 요청한 리소스가 없음

500 Internal Server Error: 서버 오류

# 프론트엔드와 백엔드 통신 방법

**RESTful API:** 가장 일반적인 방법으로, 백엔드에서 RESTful 원칙을 따르는 API를 제공하고, 프론트엔드에서는 이 API를 호출

**GraphQL:** 프론트엔드에서 필요한 데이터 형식을 지정하여 요청할 수 있음

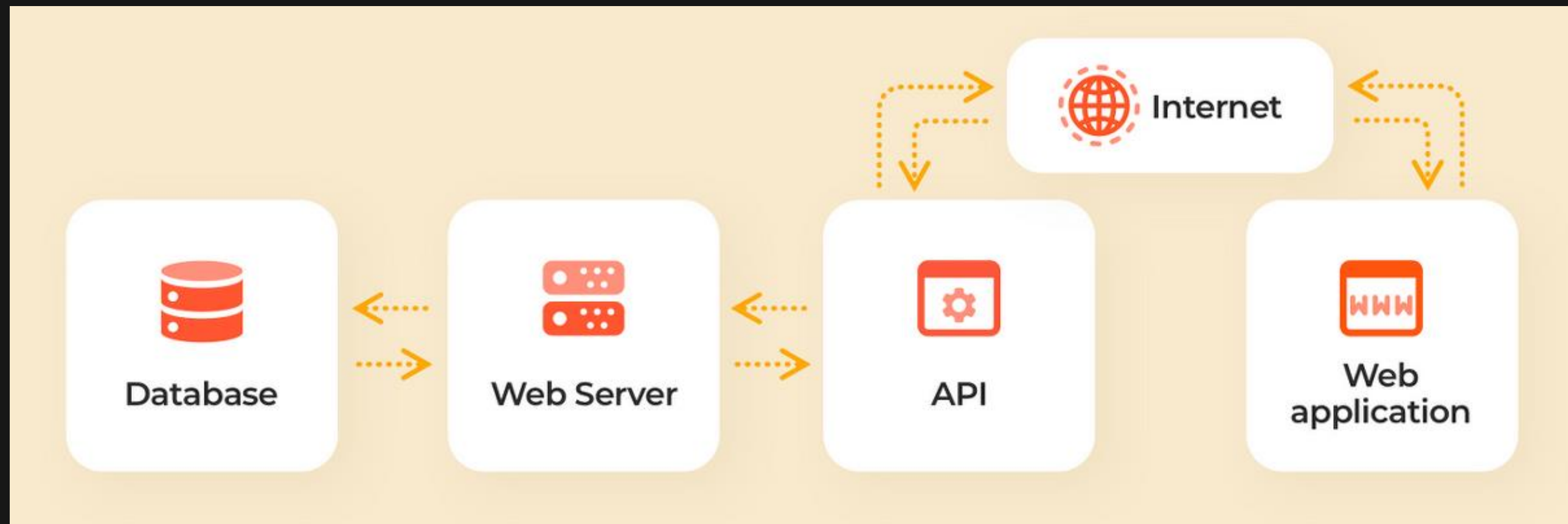
**WebSocket:** 실시간 통신이 필요한 경우 사용되며, HTTP보다 더 빠른 양방향 통신을 지원

**SSE:** 서버에서 클라이언트로 실시간으로 데이터를 전송할 때 사용

# API란?

**API:** Application Programming Interface

프로그램을 작성하기 위한 일련의 부프로그램, 프로토콜 등을 정의하여 상호 작용을 하기 위한 인터페이스



# API란?



저는  
1번메뉴



1. 파스타
2. 피자
3. 식전빵

## ▲ 식당의 API

: 식당과 손님이 음식을 주고받기 위한 방법

# REST란?

**REST**: Representational State Transfer

자원을 이름으로 구분하여 해당 자원의 상태를 주고 받는 모든 것

**“REST”**

GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update an existing movie
DELETE	/movies	Delete an existing movie

# REST란?

**REST**: HTTP URI를 통해 자원을 명시하고,  
HTTP Method(POST, GET, PUT, DELETE 등)를 통해  
해당 자원에 대한 CRUD Operation을 사용하는 것

\*CRUD Operation: 대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리 기능인 Create, Read, Update, Delete를 묶어서 일컫는 말

## REST에서 CRUD Operation 동작 예시

Create: 데이터 생성(POST)

Read: 데이터 조회(GET)

Update: 데이터 수정(PUT, PATCH)

Delete: 데이터 삭제(DELETE)

# REST 특징

**Server-Client:** 자원이 있는 쪽이 Server, 요청하는 쪽이 Client

**Stateless(무상태):** 서버는 각각의 요청을 완전히 별개의 것으로 인식하고 처리

**Cacheable(캐시 처리 기능):** 대량의 요청을 효율적으로 처리하기 위해 캐시가 요구됨

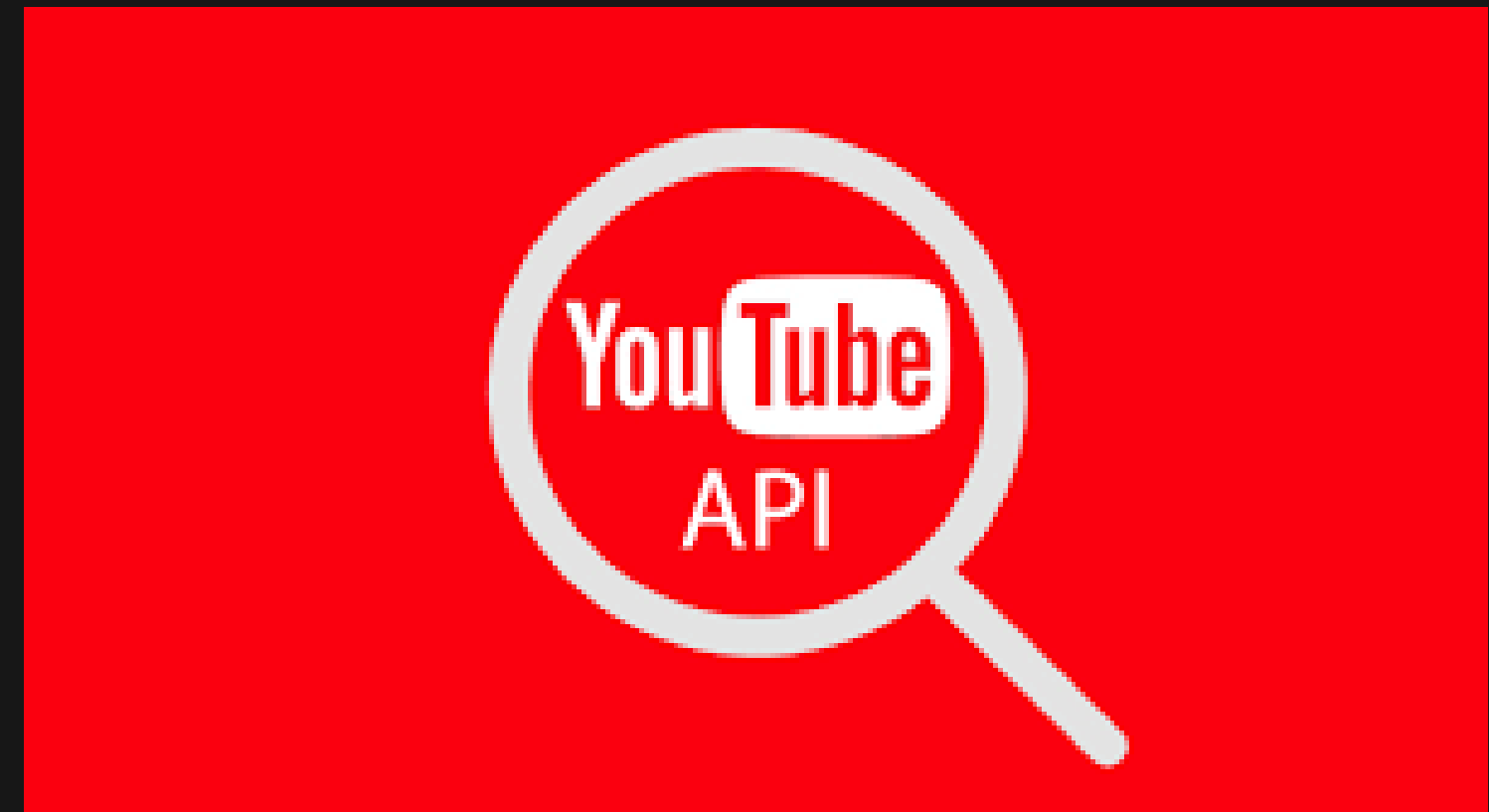
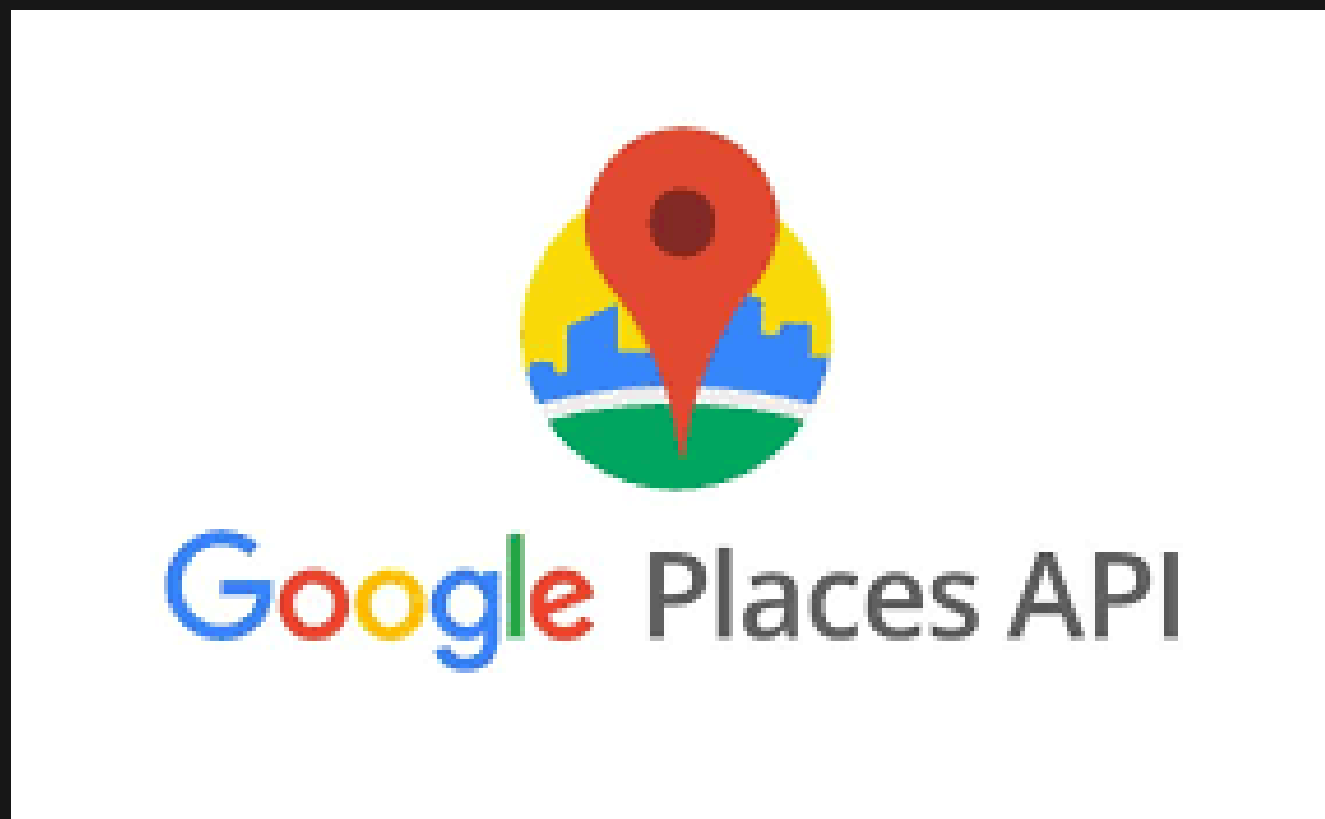
**Layered System(계층화):** REST Server는 다중 계층으로 구성될 수 있음

**Uniform Interface(인터페이스 일관성):** URI로 지정한 리소스에 대한 조작이 통일되고  
한정적인 인터페이스로 수행함.

**Code on Demand(Optional):** 서버로부터 스크립트를 받아서 클라이언트에서 실행

# REST API

REST 기반으로 서비스 API를 구현한 것  
이해가 쉽고 관리가 쉬운 API를 작성하는 방법론





# REST API 설계 기본 규칙

## 1. URI는 정보의 자원을 표현해야 한다.

- resource는 동사보다는 명사를, 대문자보다는 소문자를 사용한다.
- resource의 문서 이름으로는 단수 명사를 사용해야 한다.
- resource의 컬렉션 이름으로는 복수 명사를 사용해야 한다.
- resource의 스토어 이름으로는 복수 명사를 사용해야 한다.

ex) GET /Member/1 -> GET /members/1

# REST API 설계 기본 규칙

2. 자원에 대한 행위는 HTTP Method로 표현한다.

- URI에 HTTP Method가 들어가면 안된다.

  - ex) GET /members/delete/1 -> DELETE /members/1

- URI에 행위에 대한 동사 표현이 들어가면 안된다.

  - ex) GET /members/show/1 -> GET /members/1

  - ex) GET /members/insert/2 -> POST /members/2

- 경로 부분 중 변하는 부분은 유일한 값으로 대체한다.

  - ex) student를 생성하는 route: POST /students

  - ex) id=12인 student를 삭제하는 route: DELETE /students/12

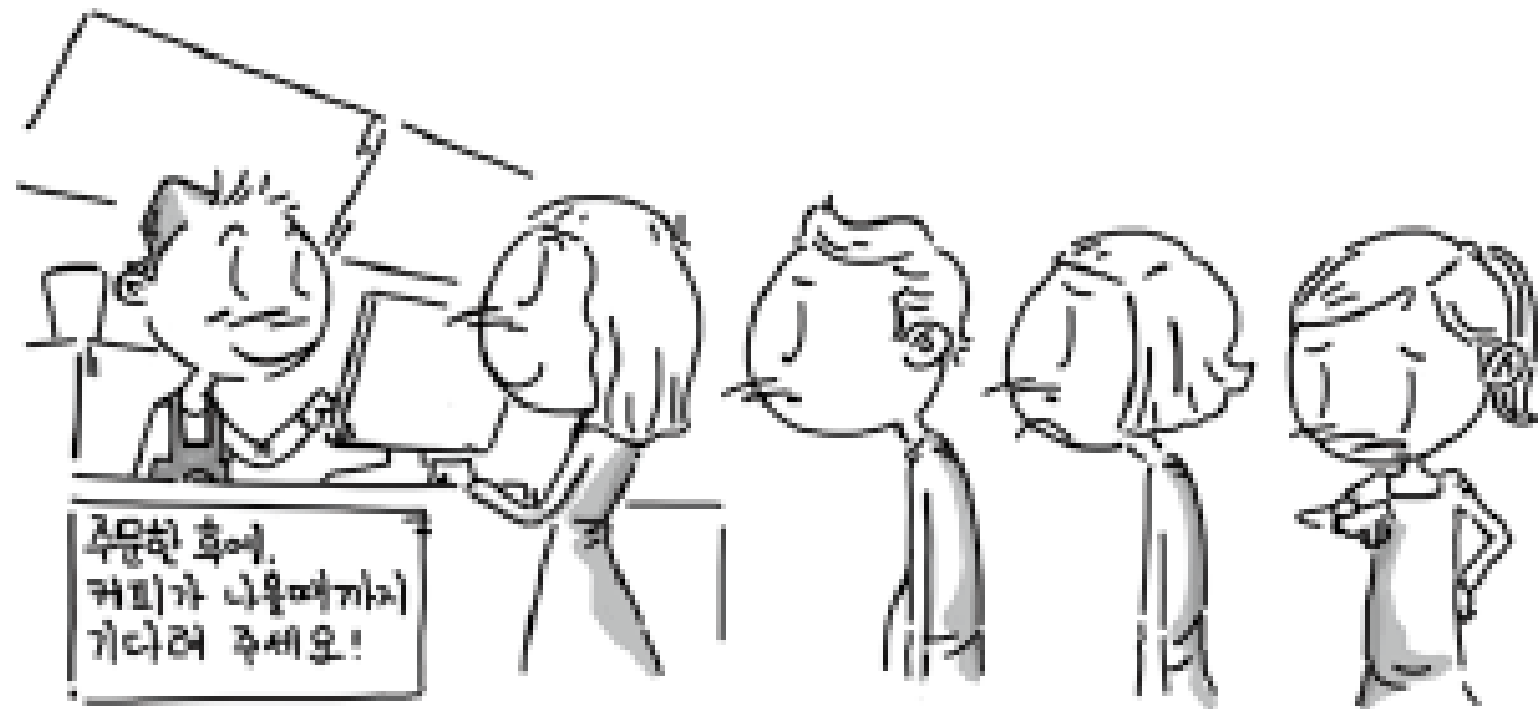
# RESTful

**RESTful**: REST의 원리를 따르는 시스템

REST를 사용해도 RESTful하지 않을 수 있음.

→ REST API의 설계 규칙을 올바르게 지키지 못한 시스템은 REST API를 사용하였지만 RESTful하지 못한 시스템

## 동기(Synchronous)



## 비동기(Asynchronous)



## 동기(Synchronous)

- 리로드가 되는 동안 다른 데이터를 처리할 수 없음.
- 전체 페이지를 리로드하기 때문에 서버의 부하가 더 커지고, 시간도 오래 걸림.

## 비동기(Asynchronous)

- 서버에서 return Data가 날라오던 말던 다른 작업을 진행.
- 대기시간이 줄어들어 웹페이지를 더욱 효과적으로 사용할 수 있음.
- 일부분만 리로드하여 시간 단축.

**form**

서버와 통신하는 동기적인 방법

서버와 통신하는 비동기적인 방법

**ajax**

# form

## 장점

- 설계가 매우 간단하고 직관적임

## 단점

- 동기 방식이라 전송 요청을 보내고 응답이 올 때까지 기다려야 함
- 전체 리소스를 다시 불러와야 하기 때문에 리소스 낭비 발생

사용 예시: 로그인, 개발자가 정말정말 코드 짜기 귀찮을 때

# AJAX(Asynchronous Javascript And XML)

- 자바스크립트를 이용해 서버와 브라우저가 비동기 방식으로 데이터를 교환할 수 있는 통신 기능.
- 클라이언트와 서버 간에 XML 데이터를 주고 받는 기술.

\* XML은 마크업 언어



jQuery

"코드가 브라우저의 영향을 받아 작동하지 못하는 문제를 해결하기 위해 개발된  
오픈소스 기반의 자바스크립트 라이브러리"

# jQuery

## 기능

- HTML 요소를 간단하고 편리하게 사용하는 기능 내재
- 자바스크립트 코드를 단순하게 변경
- DOM(Document Object Model) 구조 탐색이 뛰어남
- 크로스 브라우징: 다양한 브라우저에서 같은 코드로 동일한 동작

## 단점

- 느린 구현 속도
- 코드 관리의 어려움

# jQuery 적용

jQuery는 자바스크립트 라이브러리로, jQuery 파일은 자바스크립트 파일 형태로 존재하기 때문에 jQuery를 웹페이지에 로드해야 함

CDN(Content Delivery Network)을 이용하여 로드

```
<head>  
  <script src="https://code.jquery.com/jquery-3.2.1.min.js">  
  </script>  
</head>
```

\* CDN은 웹사이트의 접속자가 서버에서 콘텐츠를 다운받아야 할 때, 자동으로 가장 가까운 서버에서 다운받도록 하는 기술

# jQuery 문법

---

`$(선택자).함수();`

`$(선택자).함수1().함수2();` //함수 여러 개 실행

`$`: jQuery를 의미. jQuery를 선택하는 식별자

ex. `$(this).hide();` //해당 HTML 요소를 숨김

ex. `$("img").first();` //img 태그가 다수일 때 첫번째 한 개만 선택

# jQuery 내장함수 호출

---

`$.함수();` //함수 호출

ex. `$.trim(문자열);` //문자열 처음과 끝 공백을 제거해줌

# ajax 문법

```
$.ajax({
  type : 'post',      // 타입(get, post, put 등등)
  url : '/test',      // 요청할 서버 url
  async : true,       // 비동기화 여부(default : true)
  headers : {         // Http header
    "Content-Type" : "application/json",
    "X-HTTP-Method-Override" : "POST"
  },
  dataType : 'text',  // 응답 받을 데이터 타입(html, xml, json, text 등)
  data : JSON.stringify({ // 보낼 데이터(Object, String, Array)
    "name" : name,
    "nick" : nick
  }),
  success : function(result) { // 결과 성공 콜백함수
    console.log(result);
  },
  error : function(request, status, error) { // 결과 에러 콜백함수
    console.log(error)
  }
})
```

# Fetch API

---

웹 브라우저에서 제공하는 새로운 네트워크 요청 인터페이스

Ajax의 대체제로 등장

Promise 기반으로 설계되어 있어서 비동기적인 데이터 요청을 처리하기 용이함

간결하고 직관적인 API를 제공하며,

기존의 XMLHttpRequest보다 더 강력하고 유연한 기능 제공

# Promise

처리가 성공했는지 실패했는지 상태를 나타내는 객체.  
callback hell을 해결하기 위해 등장.

pending

Promise

```
1 // Callback Hell
2
3
4 a(function (resultsFromA) {
5     b(resultsFromA, function (resultsFromB) {
6         c(resultsFromB, function (resultsFromC) {
7             d(resultsFromC, function (resultsFromD) {
8                 e(resultsFromD, function (resultsFromE) {
9                     f(resultsFromE, function (resultsFromF) {
10                         console.log(resultsFromF);
11                     })
12                 })
13             })
14         })
15     })
16 });
17
```

.then()  
.catch()

...



# Promise 객체 생성과 호출

Promise 함수는 생성자에 executor(즉시 실행되는 함수)를 파라미터로 받음

```
const promise
    = new Promise(function(resolver, rejecter){ 실제 할 일 });
promise.then(function() {}, function() {});
promise.then(function() {}).catch(function() {});
```

executor는 **resolver**와 **rejecter**라는 2개의 callback 함수를 인자로 받음.  
Promise는 executor 내부에서 상황에 따라 **작업 성공 시 resolver**를,  
**실패 시 rejecter**를 반환하는 것을 약속.

# Fetch API

.then(성공 시 처리 내용)

.catch(실패 시 처리 내용)

```
fetch("URL")  
  .then(response => {  
    return response.json();  
  })  
  .then(data => {  
    console.log(data);  
  })  
  .catch(error => {  
    console.log(error);  
  });
```

# Fetch API

---

## GET

```
fetch('url')  
  .then(response => response.json())  
  .then(data => {  
    console.log(data);  
  })  
  .catch(error => console.log(error));
```

# Fetch API

## POST

```
fetch('url', {
  method: 'post',
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    "name" : name,
    "nickname" : nickname,
  })
})
.then(response => response.json())
.then((data) => {
  console.log(data);
})
.catch((error) => {
  console.log('Request failed', error);
});
```

# JWT Token

---

유저의 신원이나 권한을 결정하는 정보를 담고 있는 데이터 조각.

JWT 토큰을 사용해서 클라이언트와 서버가 안전하게 통신.

# Token 탈취

---

평소 통신 할 때: Access Token

Access Token이 만료되어 갱신될 때: Refresh Token

JWT Token 구조

- Header
- Payload: 만료 기간이 적혀 있음 → 바뀌도 Signature가 바뀌지 않음
- Signature: Header + Payload를 비밀키로 암호화

# Refresh Token 탈취 위험

---

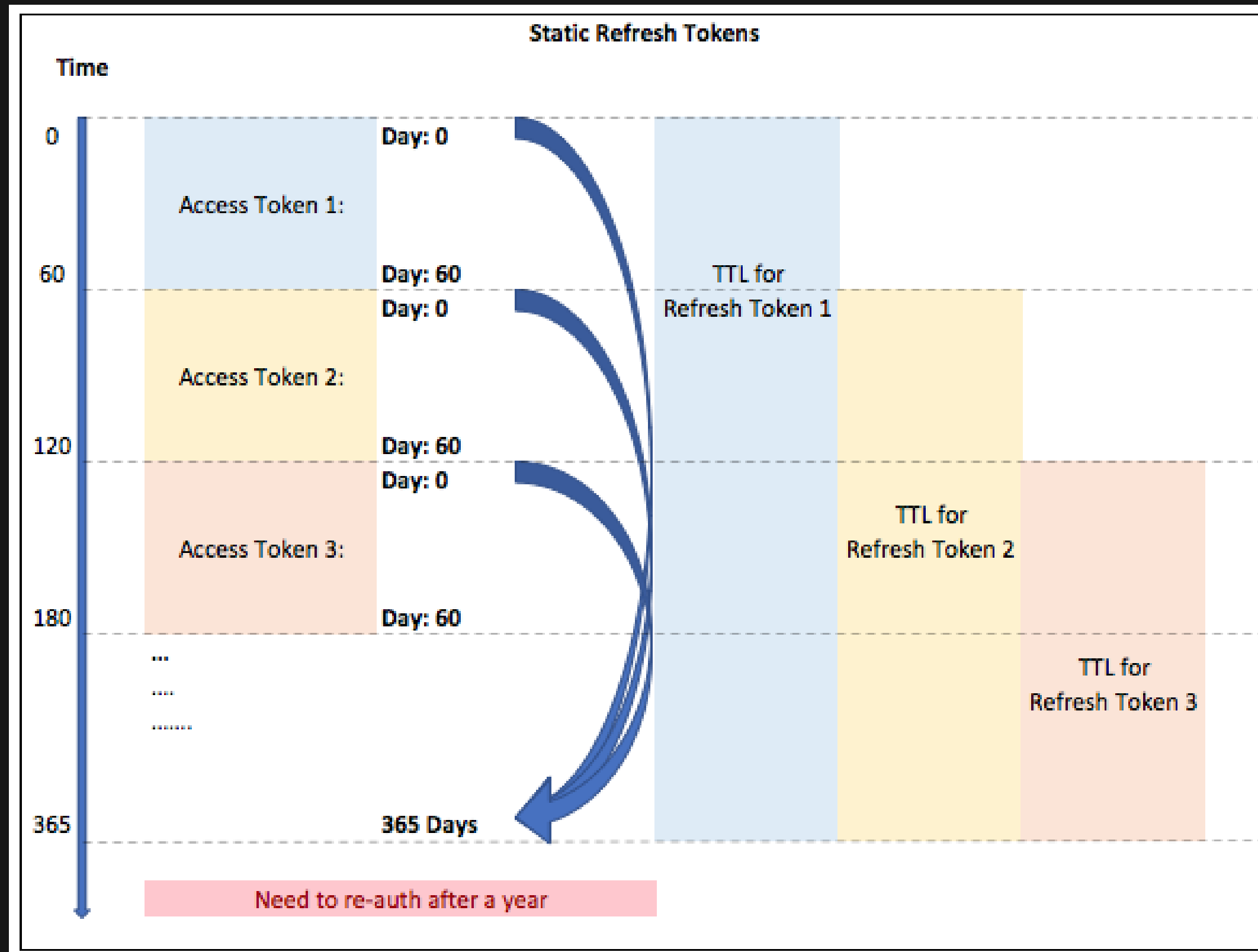
Refresh Token의 통신 빈도가 적지만 탈취 위험에서 완전히 벗어난 것은 아님.

## Refresh Token의 탈취를 예방하는 방법

Refresh Token Rotation = 클라이언트가 Access Token을 재요청할때마다

Refresh Token을 새로 발급

# Refresh Token 탈취 위험





# 실습

## 참고

`<h4>` 태그 안에 유저 이름

`<p>` 태그 안에 이메일


`<p id="list-intro">` 태그 안에 자기소개

템플릿 리터럴에 대해 찾아보기

# 수고하셨습니다!

 **Instagram**  
**@seoyeonzn**

 **EMAIL ADDRESS**  
**seoyun0207@naver.com**

 **PHONE NUMBER**  
**010-4567-4917**