# Date

| | |
|---|---|
| 강의자 | 한조 배한조 |
| 일자 | @2023년 10월 28일 |



# GMT(🍗 존맛탱?) / UTC / Timezone은 무엇인가?

## GMT (Greenwich Mean Time) :

- 경도 0도에 위치한 영국 그리니치 천문대를 기준으로 하는 태양 시간을 의미
- GMT는 자연적인 변동으로 인해 정확한 시간 표준을 제공하지 못하고, 이를 보완하기 위해 UTC가 도입
- 과거에는 GMT가 국제적인 시간 표준으로 사용

### UTC (Coordinated Universal Time):

- 원자 시계를 기반으로 하여 초당 정확한 시간을 제공하여 GMT보다 정확한 시간 표준을 위해 사용
- 현재 국제적으로 가장 널리 사용되는 시간 표준이다.

### 타임존 (Timezone):

- 타임존은 지역의 경도와 지리적 위치, 정치적 결정 등에 따라 결정된다.
- 타임존은 지역마다 UTC와의 시간 차이를 나타내는 오프셋(offset)을 사용하여 표현된다.

예를 들어 한국 표준시(KST)는 UTC+09:00 이다.

즉 UTC로 낮 12시라면 한국시간으로 오후 9시를 의미

## 요약

- GMT는 과거에 사용된 시간 표준
- UTC는 현재 세계적으로 사용되는 정확한 시간 표준
- 타임존은 지역별로 서로 다른 시간대를 나타내는 개념

# 다양한 날짜 형식과 ISO 형식

### 미국식 날짜 형식 (MM/DD/YYYY):

예: "07/27/2023" (7월 27일 2023년)

### 유럽식 날짜 형식 (DD/MM/YYYY):

예: "27/07/2023" (27일 7월 2023년)

### 한국식 날짜 형식(YYYY/MM/DD):

예: 2023/07/27 (2023년 07월 27일)

### Timestamp(Epoch Time | Unix Time)

1970년 1월 1일 00:00:00 UTC 부터 지난 시간을 `밀리초`로 표현

예: 1690463627899 (대략 2023년 07월 27일)

# ISO 형식

ISO 형식은 국제 표준 기구인 ISO(International Organization for Standardization)에서 정의한 날짜와 시간을 표현하는 표준 형식이다.

다양한 프로그래밍 언어와 시스템에서 날짜와 시간을 표현하고 전송하는데 사용되며, 인간과 기계 모두 이해하기 쉬운 형태로 구성되어 있다.

ISO 형식은 다음과 같은 형태로 구성된다. `YYYY-MM-DDTHH:mm:ss.sssZ`

- YYYY: 연도 (네 자리로 표시)
- MM: 월 (01부터 12까지의 두 자리 숫자로 표시)
- DD: 일 (01부터 31까지의 두 자리 숫자로 표시)
- T: 날짜와 시간을 구분하는 문자 "T" (Time의 약자)
- HH: 시간 (00부터 23까지의 두 자리 숫자로 표시)
- mm: 분 (00부터 59까지의 두 자리 숫자로 표시)
- ss: 초 (00부터 59까지의 두 자리 숫자로 표시)
- sss: 밀리초 (000부터 999까지의 세 자리 숫자로 표시)
- `Z`: 시간대 정보 (UTC를 의미)

예를 들어, "2023-07-27T12:30:00.000Z"는 2023년 7월 27일 오후 12시 30분 0초(UTC)를 나타낸다.

ISO 형식으로 표현된 날짜와 시간에 `"Z"가 없으면` 해당 시간은 `로컬 타임존`을 기준으로 표시된다.

# 시간을 네트워크 전송하거나 DB에 저장할 때는 어떠한 형태로 전송, 저장하는 것이 좋을까?

## ISO 형식

### 장점:

1. 시간대 변환 문제를 피할 수 있다.
2. 사람이 이해할 수 있는 표준 형식이다.
3. 국제적으로 표준화되어 데이터 교환에 용이하다.
4. 데이터베이스와 시스템의 일관성을 유지한다.

### 적합한 상황:

1. 데이터 교환과 분석에 많이 사용되는 경우
2. 시간대 변환 문제를 고려해야 하는 경우

## Timestamp 방식

### 장점:

1. 숫자 값으로 간결하게 표현되기 때문에 데이터 크기가 작고, 데이터 처리가 빠르다.
2. 빠른 데이터 접근과 인덱싱에 용이하다.
3. 정수형 숫자로 표현되기 때문에 일부 계산에 용이하다.

### 적합한 상황:

1. 데이터 크기를 최소화하고 빠른 데이터 접근이 필요한 경우
2. 데이터 처리 성능이 중요한 경우

### 누군가의 실수를 예로들어보자

모 회사에선 ISO 형식을 사용한다. 글로벌 서비스이기 때문이다.

```
'2017-03-16T17:40:00+09:00'
```

배모씨는 iso형식이 뭔지 몰랐고 뒤에 +09:00이 왜 붙는지 몰랐다. 서버 받아온 데이터를 가공할 때나 서버에 데이터를 전송할 때 오프셋을 계산하지 않고 UTC로 전송했다.

그 결과 약 30개 국가의 날짜가 모두 UTC기준으로 통일되는 대참사가 일어났다. DB도 다 꼬였다.

다행히 테스트서버에서 발생한 것이라 테스트서버 DB를 날리는 것으로 끝났지만 배모씨는 컴공이 날짜표기 방식도 모르냐며 매우 혼났다. 슬펐다.

# Date 객체 사용 예시

## 시간 생성

```
new Date('2023-07-27T12:30:00.000Z')
// Thu Jul 27 2023 21:30:00 GMT+0900 (한국 표준시)
new Date('2023-07-27T21:30:00.000')
// Thu Jul 27 2023 21:30:00 GMT+0900 (한국 표준시)
// 로컬 시간을 의미

new Date(1690461000000)
// Thu Jul 27 2023 21:30:00 GMT+0900 (한국 표준시)

new Date(2023, 06, 27, 21, 30)
// new Date(year, monthIndex, day, hours, minutes, seconds, milliseconds)
// Thu Jul 27 2023 21:30:00 GMT+0900 (한국 표준시)
```

## Timezone 시간 가져오기

```
const date = new Date('2023-07-27T21:30:00.000')
// 년
// 2023
```

```
date.getFullYear()

// 월
// 06 - 0부터 시작하므로 7월을 의미
date.getMonth()

// 일
// 27
date.getDate()

// 시간
// 21
date.getHours()

// 분
// 30
date.getMinutes()
```

# UTC 시간 가져오기

```
const date = new Date('2023-07-27T21:30:00.000')
// 년
// 2023
date.getUTCFullYear()

// 월
// 06 - 0부터 시작하므로 7월을 의미
date.getUTCMonth()

// 일
// 27
date.getUTCDate()

// 시간
// 12
date.getUTCHours()

// 분
// 30
date.getUTCMinutes()
```

# Timestamp 값 가져오기

```
const date = new Date('2023-07-27T21:30:00.000')
date.getTime()
// 1690461000000
```

## ISO 형식으로 값 가져오기

```
const date = new Date('2023-07-27T21:30:00.000')
date.toISOString()
// '2023-07-27T12:30:00.000Z'
```

## 날짜 관련 라이브러리

대표적인 날짜 관련 라이브러리로는 date-fns와 moment.js가 있다.

여기서는 심화있게 다루진 않으나 JavaScript의 date 네이티브 객체는 다루면 다룰 수록 화밖에 나지 않으니 소규모 프로젝트에선 라이브러리를 활용하는 것도 좋은 방법이다.