7주차

# 소프트웨어 시스템 설계 및 개발

2024.1학기

# CONTENTS

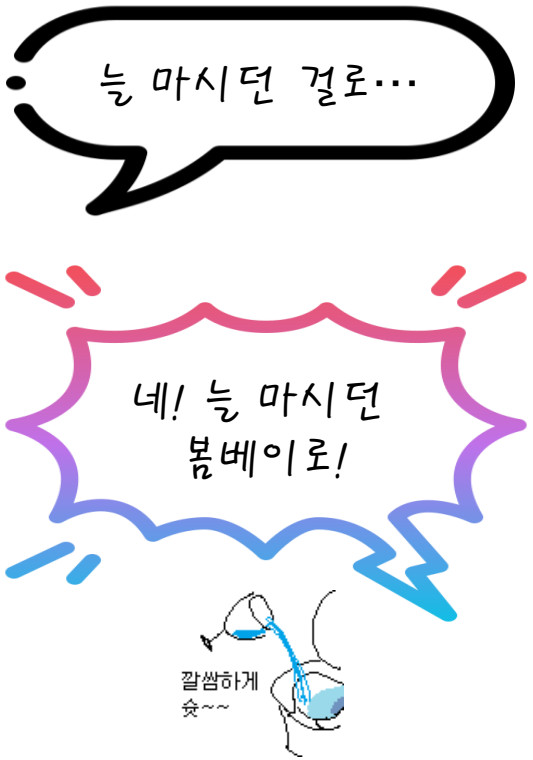
# REST

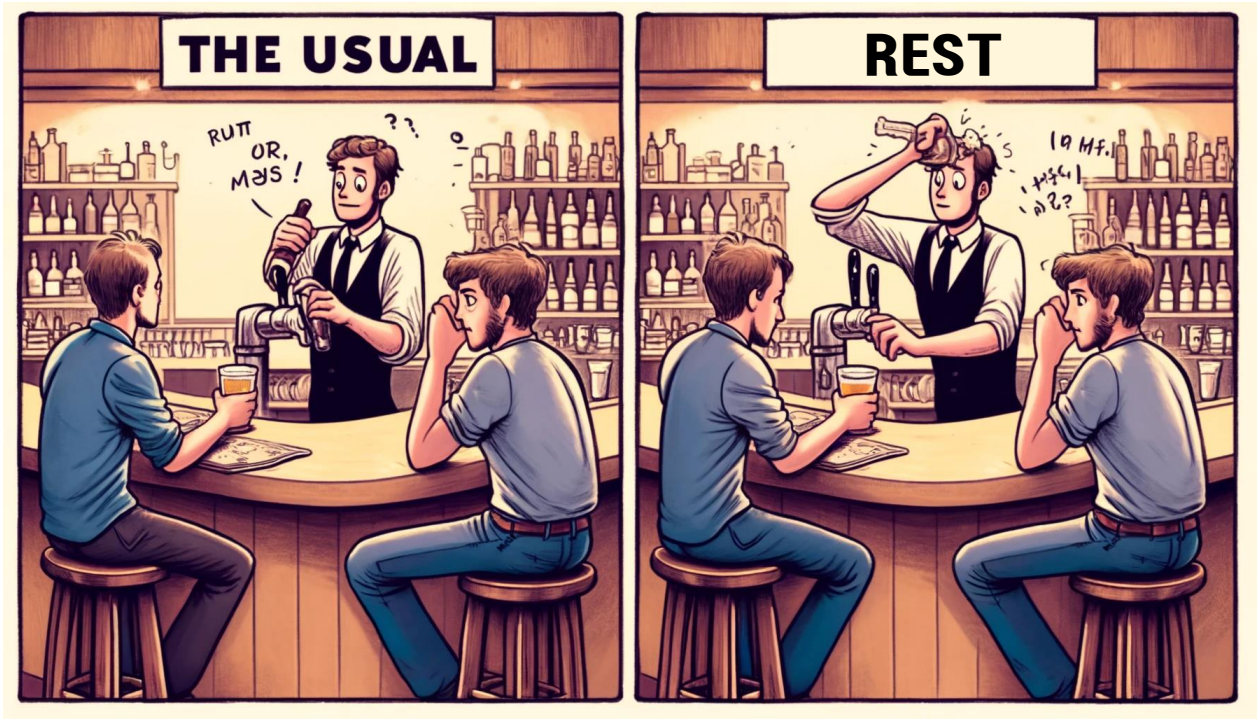
- **REST(Representational State Transfer)**
  - **REST는 웹을 위한 네트워크 아키텍처 스타일로 아래와 같은 특성을 가짐**
  - **RESTful 하다는 것은 REST 아키텍처 원칙을 준수한다는 것**

| 특성 | 내용 |
|---|---|
| Client-Server Architecture | 클라이언트와 서버는 서로 독립적으로 작동해야 하며, 이를 통해 각각의 부분을 독립적으로 발전시킬 수 있음 |
| Stateless | 각 요청은 독립적이어야 하며, 서버는 클라이언트의 상태 정보를 저장하지 않아야 함. 모든 필요한 정보는 요청과 함께 전송되어야 함 |
| Cacheable | 클라이언트는 응답을 캐시할 수 있어야 하며, 서버 응답은 캐싱 가능 여부를 명시해야 함 |
| Uniform Interface | 인터페이스의 통일성을 통해 시스템의 각 부분을 독립적으로 개선할 수 있음. 이를 위해 리소스 식별, 자기 서술적 메시지, 하이퍼미디어 등의 원칙을 따름 |
| Layered System | 클라이언트는 서버가 직접적으로 연결되어 있는지, 중간 서버를 통해 연결되어 있는지 알 수 없어야 함 |
| Code on Demand (Optional) | 서버가 클라이언트에 실행 가능한 코드를 전송할 수 있어야 함(옵션) |

# REST (Stateless)

- **Stateless**
  - 클라이언트와 서버 간의 각 통신은 완전히 독립적이어야 하며, 서버는 클라이언트의 상태나 세션 정보를 저장하면 안 됨
  - 대신, 각 요청은 필요한 모든 정보를 포함해야 하고, 서버는 그 요청만을 처리하여 응답해야 함
  - 요청이 서로 의존적이지 않기 때문에, 각 요청은 이전의 다른 요청들로부터 독립적



# REST (Stateless)

- State를 저장 안 함
  - 독립성
    - 각 요청이 필요한 모든 정보를 포함하므로, 서버는 이전 요청의 상태를 기억할 필요가 없어서 서버 설계를 단순화하고, 다른 서버로 요청을 자유롭게 전달할 수 있게 함
  - 확장성
    - 서버가 상태 정보를 유지하지 않기 때문에, 어떤 서버에 요청이 배치되더라도 모든 서버가 동일하게 요청을 처리할 수 있음
    - 이는 서버 클러스터의 확장성을 증가시키며, 서비스의 가용성과 장애 허용성을 향상시킴
  - 성능
    - 각 요청이 독립적으로 처리되므로, 서버는 상태를 관리하기 위한 추가적인 리소스를 사용할 필요가 없어 서버의 처리 성능을 최적화함
- State를 저장함
  - 복잡성 증가
    - 서버가 클라이언트의 상태를 추적하려면 추가적인 로직과 저장 공간이 필요하여, 서버의 복잡성을 증가시키고, 오류 발생 가능성을 높임
  - 확장성 제한
    - 서버가 상태 정보를 저장하면, 특정 클라이언트의 요청을 처리할 수 있는 서버가 제한될 수 있어 로드 밸런싱을 어렵게 만들고, 서버 확장을 복잡하게 할 수 있음
  - 리소스 사용 증가
    - 상태 정보를 저장하고 관리하기 위해 추가 서버 리소스가 필요하므로 전체 시스템의 성능에 부담을 줌

# 세션과 쿠키

- 세션(Session)
  - 세션은 서버 측에서 사용자 데이터를 저장
  - 세션 ID는 일반적으로 쿠키를 통해 클라이언트에 저장되며, 이 ID를 사용하여 서버는 세션 데이터에 접근하고 사용자를 식별
  - 세션은 주로 로그인 상태 유지에 사용
- 쿠키(Cookies)
  - 클라이언트 측에서 작은 데이터 조각을 저장
  - 이 데이터는 웹 브라우저에 저장되며, 사용자가 웹사이트를 다시 방문할 때마다 웹 서버로 전송
  - 쿠키는 사용자의 선호, 세션 트래킹, 사용자 식별 정보 등을 저장
- 그럼 세션과 쿠키를 쓰면 RESTful 하지 않은가...?
  - 그렇다
  - 원칙적으로는 세션과 쿠키는 REST 아키텍처를 위반하는 것이지만, 사용자 편의를 위해 실제 서비스에서는 세션과 쿠키를 사용하고 있음
  - 따라서 세션과 쿠키를 쓰더라도 RESTful 하다고 표현하는 경우가 많음

# REST API

- REST API는 REST 아키텍처를 기반으로 한 API(Application Programming Interface)
- REST API의 구성요소
  - GET : 리소스를 조회
  - POST : 새 리소스를 생성
  - PUT : 리소스를 갱신
  - DELETE : 리소스를 삭제
- REST API 사용의 장점
  - 간단함과 범용성: HTTP 프로토콜을 사용하므로, REST API는 이해하기 쉽고 사용하기 간편함
  - 언어와 플랫폼 독립적: 어떤 프로그래밍 언어나 플랫폼에서도 사용할 수 있음
  - 확장성과 유연성: 새로운 리소스나 메서드를 기존 시스템에 쉽게 추가할 수 있음

# JSON

- **JSON**
  - **JavaScript Object Notation은 데이터 교환을 위해 개발된 경량 데이터 형식**
  - **JSON은 언어 독립적이며, 대부분의 프로그래밍 언어에서 사용이 가능함**
  - **웹 개발에서 클라이언트<->서버 간의 데이터를 전송하는 표준 방식으로 많이 사용됨**
  - **{ }로 감싸져 있으면 k,v 쌍이 나와야 함. 순서는 상관없음**
  - **[ ]는 배열로 어떤 형태라도 들어갈 수 있으며 순서가 있음**

```
{
  "name": "Park Jonggyu",
  "age": 38,
  "isStudent": false,
  "address": {
    "street": "Chungryeol-ro",
    "city": "Busan"
  },
  "phoneNumbers": ["010-1234-5678", "070-0000-9999"]
}
```

```
// 객체를 JSON 문자열로 변환
const jsonObj = { name: "Jong", age: 38 };
const jsonString = JSON.stringify(jsonObj);
console.log(jsonString);

// JSON 형태의 문자열을 객체로 변환
const jsonString = '{"name":"Jong","age":38}';
const jsonObj = JSON.parse(jsonString);
console.log(jsonObj.name);
```

# 예제 (1/2)

- **REST API**

```
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json()); // JSON 파싱 미들웨어

// 초기 사용자 데이터
let users = [
    { id: 1, name: 'Kim' },
    { id: 2, name: 'Lee' },
    { id: 3, name: 'Park' }];

// 모든 사용자 정보 조회
app.get('/users', (req, res) => {
  res.status(200).json(users);
});

// 특정 사용자 정보 조회, :id는 변수처럼 사용한다는 의미
// 해당 위치에 id 값인 1, 2, 3등을 넣으면 됨 /users/1, /users/2 이런 식
app.get('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (user) {
    res.status(200).json(user);
  } else {
    res.status(404).send('User not found');
  }
});
```

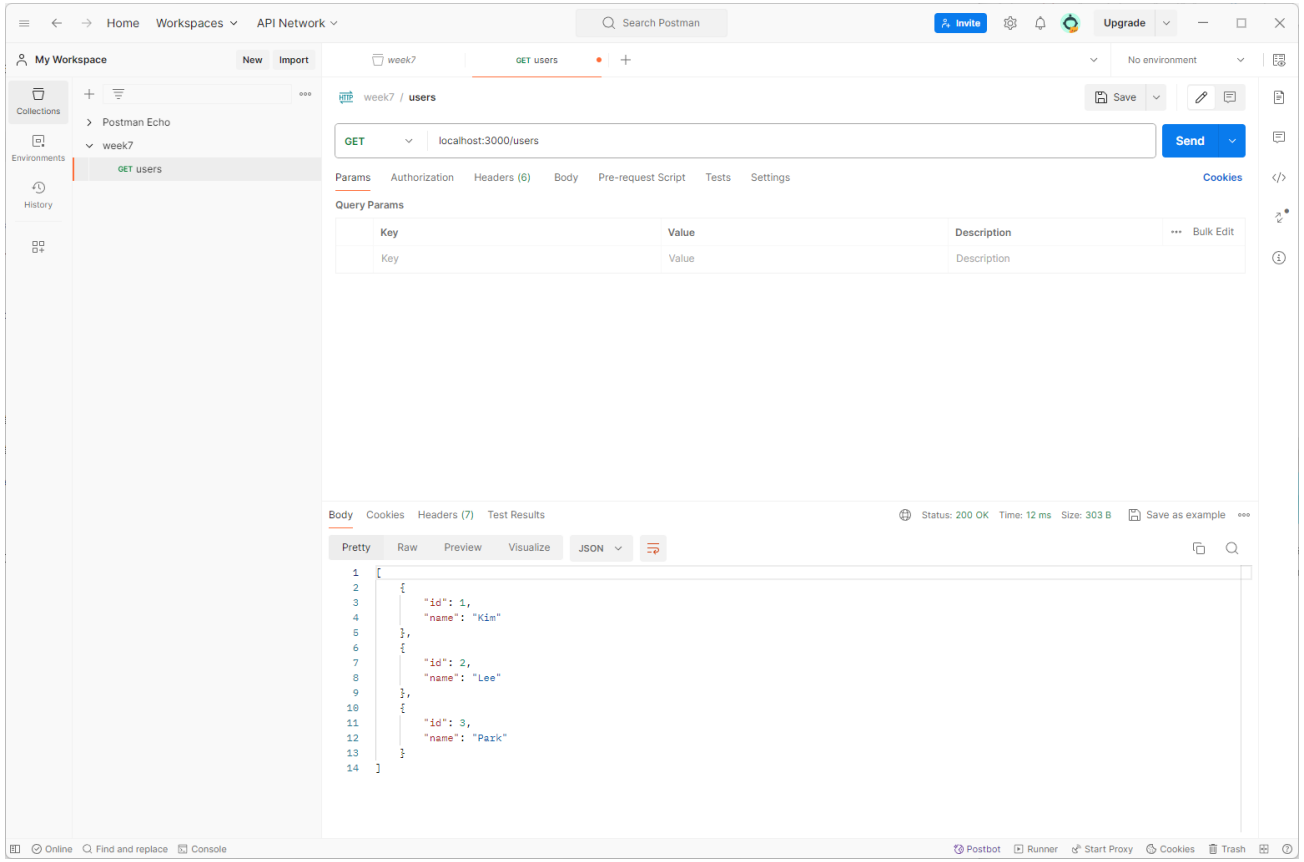
<1/2>

```
// 사용자 생성
app.post('/users', (req, res) => {
  const user = {
    id: users.length + 1,
    name: req.body.name
  };
  users.push(user);
  res.status(201).json(user);
});

// 사용자 정보 수정
app.put('/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (user) {
    user.name = req.body.name;
    res.status(200).json(user);
  } else {
    res.status(404).send('User not found');
  }
});

// 사용자 삭제
app.delete('/users/:id', (req, res) => {
  users = users.filter(u => u.id !== parseInt(req.params.id));
  res.status(204).send();
});

// 서버 시작
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

<2/2>

# 예제 (2/2)

- **POSTMAN**
  - **API 개발 및 활용에 유용한 도구**
  - **간단한 사용법은 수업시간 예제와 함께 설명**



# 실습1

- REST API로 회원가입과 로그인을 구현해보세요. (난이도1)
- 회원의 정보를 파일로 저장하고, 로그인도 파일에서 내용을 찾는 것으로 구현해보세요. (난이도3)

# 실습2 - db

- 아래 예제 코드 일부를 참고하여 DB 연동을 해보세요.
- 앞의 회원가입을 DB를 사용해서 구현해보세요. (난이도2)

```
// npm install mysql
var mysql = require('mysql');
var db = mysql.createConnection({
    host:'localhost',
    user:'jong',
    password:'1234!@#$qwerQWER',
    database:'students'
});

db.connect();
```

```
db.query('select * from student', function(err, result){
            if (err){
                console.log(err);
            }
            console.log(result);
            res.writeHead(200, {"Content-type":"text/html"});
            res.end(result[0].name);
        });
```

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| num   | int(11)     | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| point | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> select * from student;
+------+---------+-------+
| num  | name    | point |
+------+---------+-------+
|    1 | jonggyu |    88 |
|    2 | gildong |    70 |
+------+---------+-------+
2 rows in set (0.00 sec)
```