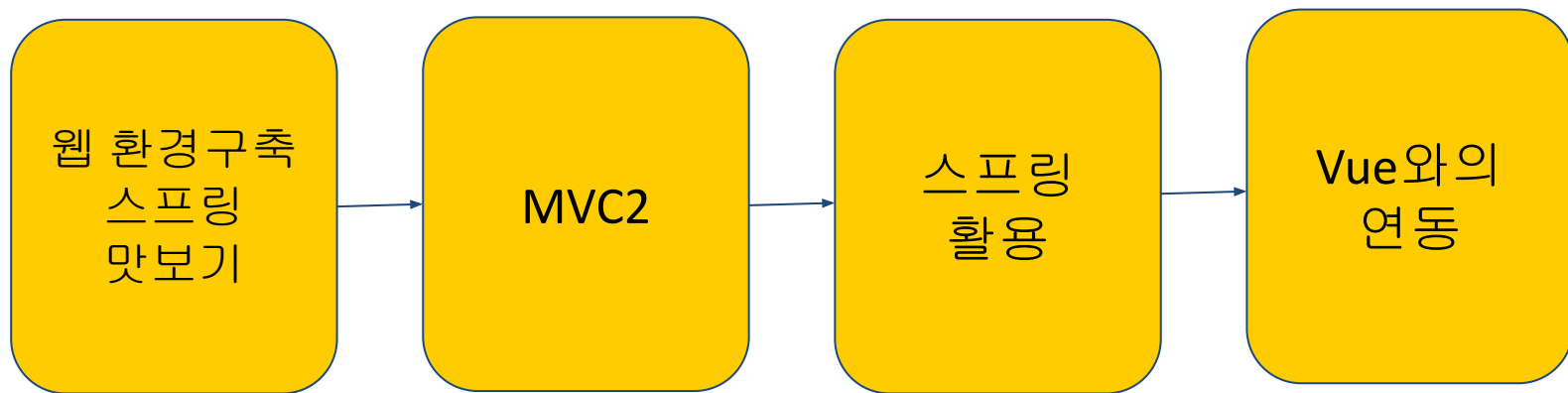


2024년 상반기 K-디지털 트레이닝

SPRING04 - Rest

[KB] IT's Your Life

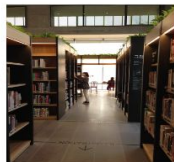




우리도 JSON 받아보자!

우리도 JSON 많이 받아보자!(리스트로)

우리도 JSON 많이 받아보자!(테이블로)

아이디	북마크이름	url	img
naver0	naver0	GO URL	img: 
naver1	naver1	GO URL	img: 
naver2	naver2	GO URL	img: 

우리도 JSON 받아보자!	우리도 JSON 많이 받아보자!(리스트로)
우리도 JSON 많이 받아보자!(테이블로)	
아이디: naver0 북마크이름: naver0 url: GO URL img:	
아이디: naver1 북마크이름: naver1 url: GO URL img:	
아이디: naver2 북마크이름: naver2 url: GO URL img:	

Rest



- REST(Representational State Transfer)는 웹 서비스 디자인 아키텍처의 하나
- Roy Fielding이 2000년에 그의 박사 학위 논문에서 소개한 개념
- REST는 네트워크의 리소스를 정의하고 이 리소스에 대한 주소(URL)를 지정하여 상태를 전송하는 방법을 제공
- RESTful 시스템은 HTTP를 사용하여 이러한 리소스를 처리하고 조작
- 단어의 유래
 - **Representational:** 리소스의 "표현"을 전송한다는 의미
 - **State:** 리소스의 상태를 주고받는다라는 의미
 - **Transfer:** 클라이언트와 서버 간에 상태 정보를 주고받는다라는 의미

- REST의 핵심 개념

- **자원(Resource):** 모든 것이 자원으로 간주. 자원은 특정 데이터나 정보에 해당하며, URI(Uniform Resource Identifier)를 통해 식별
- **표현(Representation):** 자원의 현재 상태는 XML, JSON, HTML 등 다양한 형식으로 표현
- **상태 전이(State Transfer):** 클라이언트와 서버 간에 자원의 상태를 주고받으며, 클라이언트가 서버에 요청을 보내면 서버는 자원의 상태를 전달
- **HTTP 메서드:** REST는 주로 HTTP 프로토콜을 사용하며, 주로 다음과 같은 HTTP 메서드를 활용
 - **GET:** 자원 조회
 - **POST:** 자원 생성
 - **PUT:** 자원 업데이트
 - **DELETE:** 자원 삭제
- **무상태(Stateless):** 각 요청은 독립적이며, 서버는 요청 간에 상태를 유지하지 않음. 따라서 각 요청에는 필요한 모든 정보가 포함되어야 함.

- REST의 핵심 개념

- **Idempotency (멱등성):** 여러 번 실행해도 결과가 동일한 연산의 특성, 예를 들어, GET, PUT, DELETE 요청은 멱등성을 가져야 함.
- **Hypermedia as the Engine of Application State (HATEOAS):** 클라이언트가 동적으로 서버에서 제공하는 링크를 따라 자원에 접근하거나 상태를 변경할 수 있게 함.
- **Client-Server Architecture (클라이언트-서버 아키텍처):** 클라이언트와 서버가 서로 독립적으로 개발되고 배포될 수 있는 구조를 의미. 클라이언트는 사용자 인터페이스와 관련된 작업을 수행하고, 서버는 데이터 저장 및 처리를 담당

- REST를 사용하는 이유

1. **단순성**: HTTP 프로토콜을 기반으로 하기 때문에 기존의 웹 기술과 잘 통합
2. **확장성**: RESTful 서비스는 쉽게 확장 가능하며, 클라이언트와 서버가 독립적으로 개발
3. **유연성**: 다양한 데이터 형식을 사용할 수 있어 유연한 데이터 전송이 가능
4. **표준화**: HTTP 표준을 따르기 때문에 RESTful API는 다양한 클라이언트와 쉽게 상호 작용

- REST API

- REST 원칙을 따르는 애플리케이션 프로그래밍 인터페이스를 의미
- RESTful API와 비교할 때 보다 일반적인 개념

- RESTful API

- REST 원칙을 따르는 애플리케이션을 구현하는 방식에 대한 구체적인 지침을 의미
- REST API의 구현을 통해 일관된 인터페이스를 제공하는 것을 목표로 함.
- 특징
 - **자원 기반 접근**: 자원은 URI를 통해 접근할 수 있으며, 각 자원은 고유한 URI를 가짐.
 - **HTTP 메서드 사용**: 자원에 대한 CRUD 작업(Create, Read, Update, Delete)을 수행하기 위해 HTTP 메서드(GET, POST, PUT, DELETE)를 사용
 - **표현형식**: JSON 또는 XML과 같은 표준 표현 형식을 사용하여 자원의 상태를 클라이언트와 서버 간에 교환
 - **HATEOAS**: 자원 상태와 관련된 링크를 제공하여 클라이언트가 동적으로 서버와 상호작용

- 둘 다 REST 원칙을 기반으로 하지만, **RESTful API는 REST API의 구현 세부 사항을 명확하게 따르는 것을 의미**

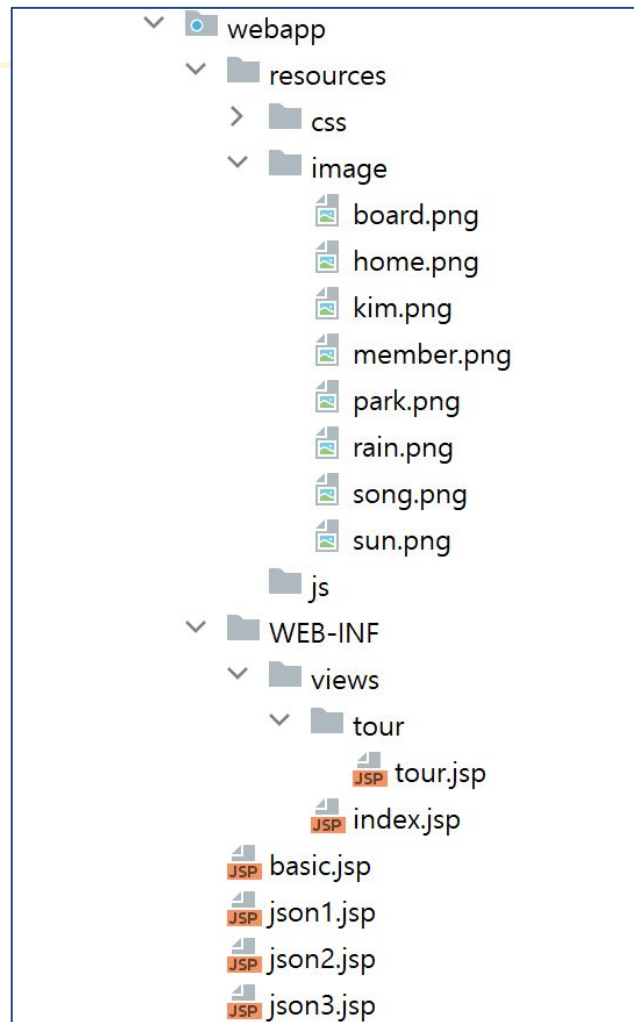
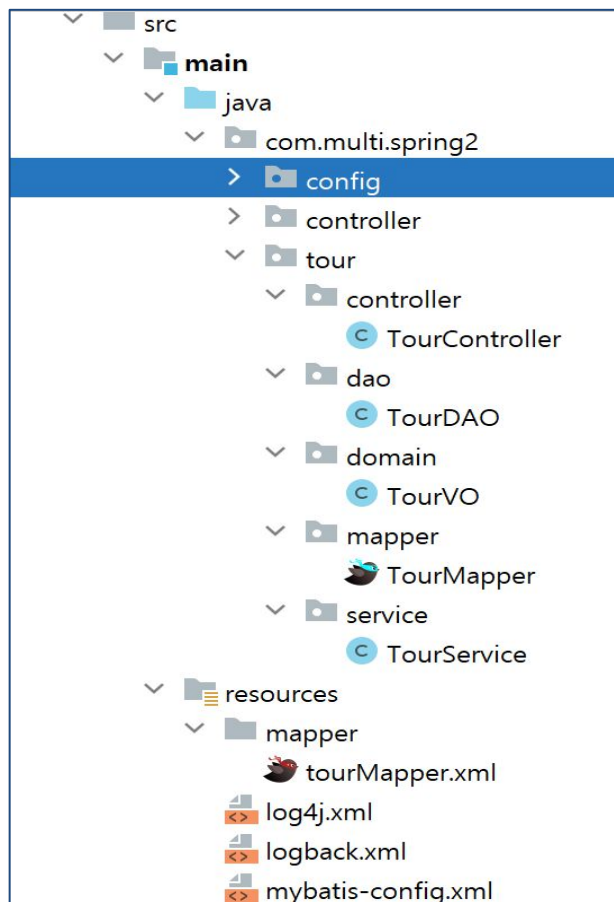
```
@RestController
@RequestMapping("/api")
class ApiController {
    private Map<Integer, String> data = new HashMap<>();

    @GetMapping("/data/{id}")
    public String getData(@PathVariable int id) {
        return data.getOrDefault(id, "No data found");
    }

    @PostMapping("/data")
    public String addData(@RequestParam int id,
                        @RequestParam String value) {
        data.put(id, value);
        return "Data added";
    }
}
```

```
@PutMapping("/data/{id}")
public String updateData(@PathVariable int id,
                        @RequestParam String value) {
    data.put(id, value);
    return "Data updated";
}

@DeleteMapping("/data/{id}")
public String deleteData(@PathVariable int id) {
    data.remove(id);
    return "Data deleted";
}
}
```



Maven Repository: jackson-databind

mvnrepository.com/search?q=jackson-databind

Repository

Found 1488 results

Sort: **relevance** | popular | newest

1. **Jackson Databind** 20,166 usages
 com.fasterxml.jackson.core » jackson-databind Apache

General data-binding functionality for Jackson: works on core streaming API

Last Release on Sep 30, 2021

JSON.simple » 1.1
 A simple Java toolkit for JSON

License	Apache 2.0
Categories	JSON Libraries
Tags	json format
HomePage	http://code.google.com/p/json-simple/
Date	Aug 12, 2009
Files	pom (1 KB) jar (15 KB) View All
Repositories	Central AdobePublic Archive Mulesoft Redhat GA WSQ2 Public Xceptance
Ranking	#225 in MvnRepository (See Top Artifacts) #9 in JSON Libraries
Used By	1,946 artifacts

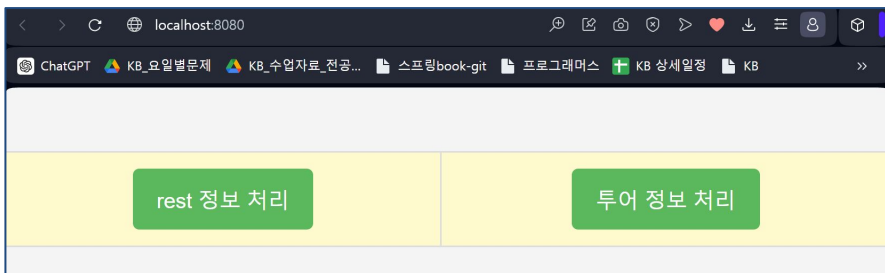
Note: There is a new version for this artifact

New Version 1.1.1

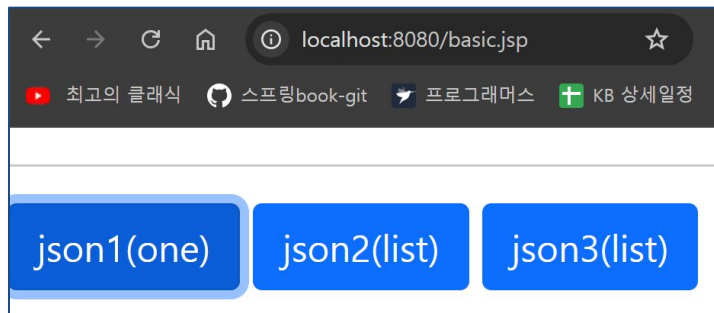
Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
<dependency>
  <groupId>com.googlecode.json-simple</groupId>
  <artifactId>json-simple</artifactId>
  <version>1.1</version>
</dependency>
```

```
implementation 'org.springframework:spring-core:5.3.37'
implementation "org.springframework:spring-context:5.3.37"
implementation "org.springframework:spring-webmvc:5.3.37"
implementation 'javax.inject:javax.inject:1'
implementation 'org.springframework:spring-web:5.3.37'
implementation 'org.springframework:spring-jdbc:5.3.37'
implementation 'com.fasterxml.jackson.core:jackson-databind:2.15.2' // Jackson 라이브러리 추가
implementation 'com.googlecode.json-simple:json-simple:1.1.1'
```



```
<%@ page contentType="text/html; charset=UTF-8 " pageEncoding="UTF-8 "
%>
<!DOCTYPE html>
<html>
<head>
<title>Spring</title>
<link rel="stylesheet" href="resources/css/out.css" >
</head>
<body>
<h1></h1>
<br/>
<table>
<tbody>
<tr>
<td style="background: lemonchiffon; text-align: center">
<a href="basic.jsp">
<button>rest 정보 처리</button>
</a>
</td>
<td style="background: lemonchiffon; text-align: center">
<a href="tour">
<button>투어 정보 처리</button>
</a>
</td>
</tr>
</tbody>
</table>
</body>
</html>
```



```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Axios Example</title>
  <!-- Bootstrap 5 CSS -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.m
in.css" rel="stylesheet">
  <!-- Bootstrap Icons CSS -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons
.css" rel="stylesheet">
  <script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
</head>
<body>

<hr color="red">
<button id="b1" class="btn btn-primary">json1 (one) </button>
<button id="b2" class="btn btn-primary">json2 (list) </button>
<button id="b3" class="btn btn-primary">json3 (list) </button>
<div class="container mt-5">
  <div id="result" class="row"></div>
</div>
```

```
<%@ page import="org.json.simple.JSONObject" %>
<%@ page language="java"
contentType="text/html; charset=UTF-8" %>
<%
    JSONObject json = new JSONObject();
    json.put("today", "rain");
    json.put("temp", "20");
    json.put("hu", "90");
%>
<%= json.toJSONString() %>
```

json1.jsp

```
<%@ page import="org.json.simple.JSONObject" %>
<%@ page import="org.json.simple.JSONArray" %>
<%
    //hashmap(key:value)
    JSONObject json = new JSONObject();
    json.put("today", "rain");
    json.put("temp", "20");
    json.put("hu", "90");
    json.put("location", "seoul");

    JSONObject json2 = new JSONObject();
    json2.put("today", "sun");
    json2.put("temp", "15");
    json2.put("hu", "70");
    json2.put("location", "busan");

    //arraylist(element)
    JSONArray array = new JSONArray();
    array.add(json);
    array.add(json2);
%>
<%= array.toJSONString() %>
```

json2.jsp

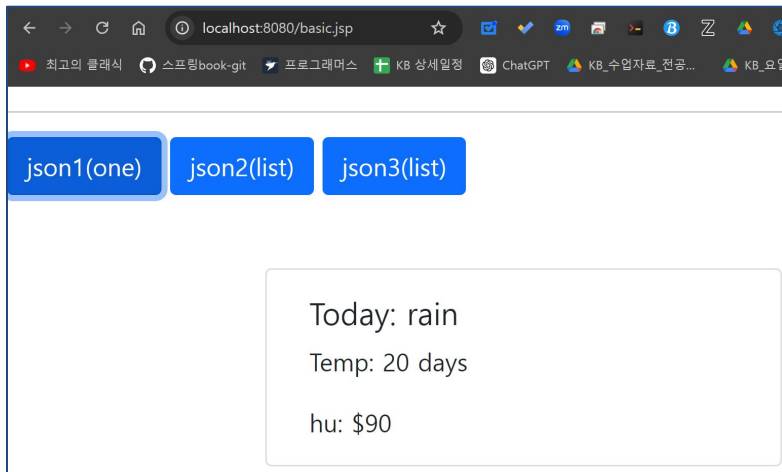
json3.jsp

```
<%@ page import="org.json.simple.JSONObject" %>
<%@ page import="org.json.simple.JSONArray" %>
<%
    //hashmap(key:value)
    JSONObject json = new JSONObject();
    json.put("name", "kim");
    json.put("age", 20);
    json.put("pic", "kim.png");

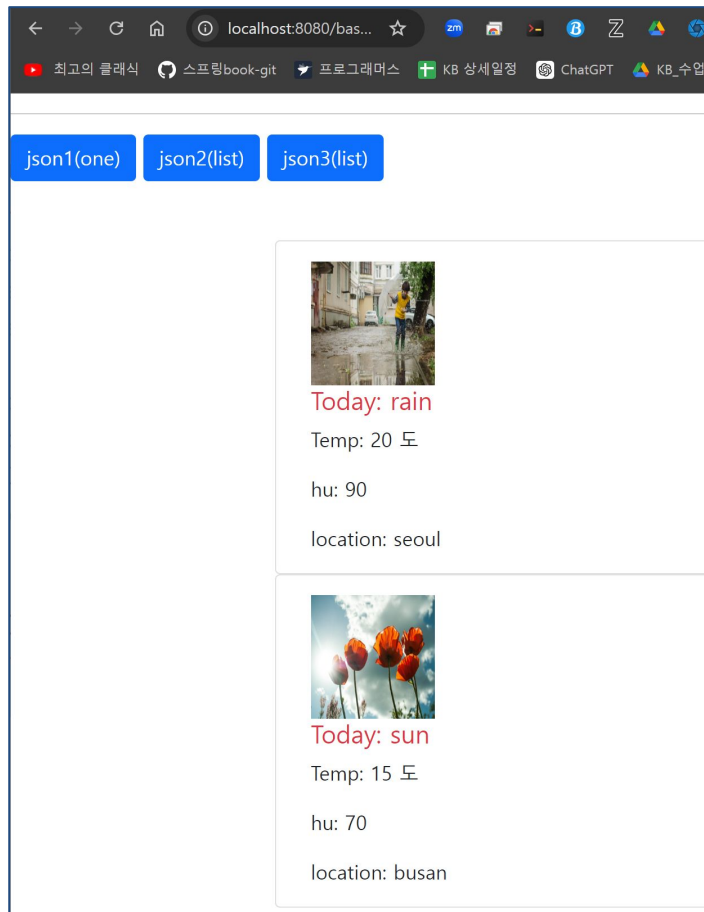
    JSONObject json2 = new JSONObject();
    json2.put("name", "park");
    json2.put("age", 30);
    json2.put("pic", "park.png");

    JSONObject json3 = new JSONObject();
    json3.put("name", "song");
    json3.put("age", 40);
    json3.put("pic", "song.png");

    //arraylist(element)
    JSONArray array = new JSONArray();
    array.add(json);
    array.add(json2);
    array.add(json3);
%>
<%= array.toJSONString() %>
```

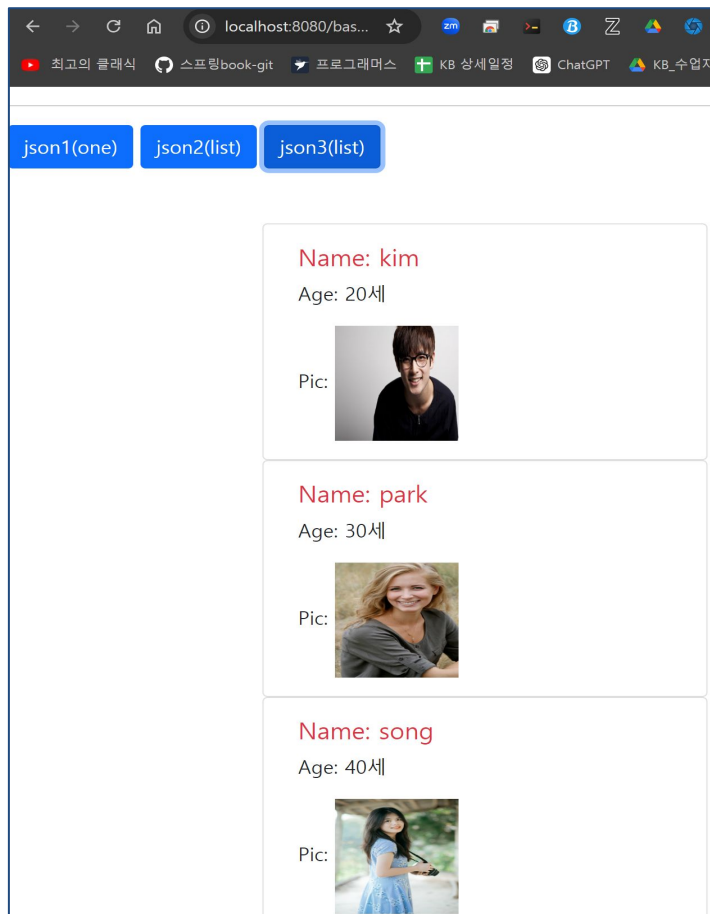
```
document.getElementById('b1').addEventListener('click', function() {  
    // Axios로 JSON 데이터를 호출  
    axios.get('json1.jsp')  
        .then(function(response) {  
            // 데이터를 받아서 result div에 출력  
            let json = response.data;  
            console.log(json)  
            var resultDiv = document.getElementById('result');  
  
            // 기존 내용을 지우기  
            resultDiv.innerHTML = '';  
  
            let jsonElement = document.createElement('div');  
            jsonElement.className = 'card col-12 col-md-6 offset-md-3';  
            jsonElement.innerHTML = '<div class="card-body">' +  
                '<h5 class="card-title">Today: ' + json.today + '</h5>' +  
                '<p class="card-text">Temp: ' + json.temp + ' days</p>' +  
                '<p class="card-text">hu: $ ' + json.hu + '</p>' +  
                '</div>';  
            resultDiv.appendChild(jsonElement);  
        })  
        .catch(function(error) {  
            console.error('Error fetching the tour:', error);  
        });  
}); //b1
```



```
document.getElementById('b2').addEventListener('click', function() {
  // Axios로 JSON 데이터를 호출
  axios.get('json2.jsp')
    .then(function(response) {
      // 데이터를 받아서 result div에 출력
      let jsonArray = response.data;
      let resultDiv = document.getElementById('result');

      // 기존 내용을 지우기
      resultDiv.innerHTML = '';
      let img_result = "<img src=resources/image/rain.png width='100' height='100'><br>";

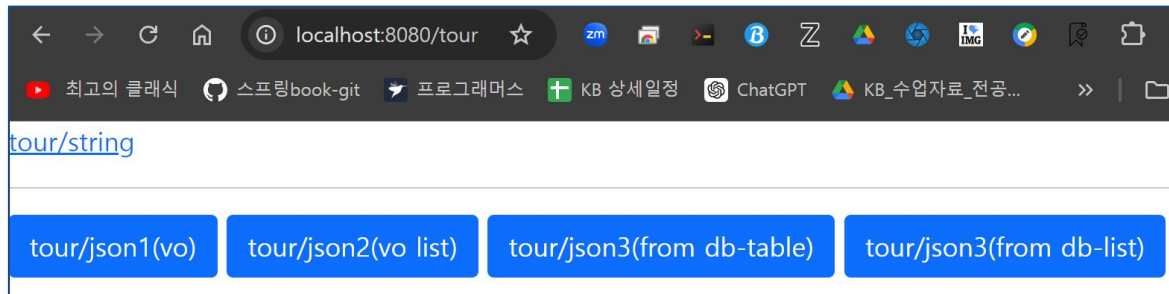
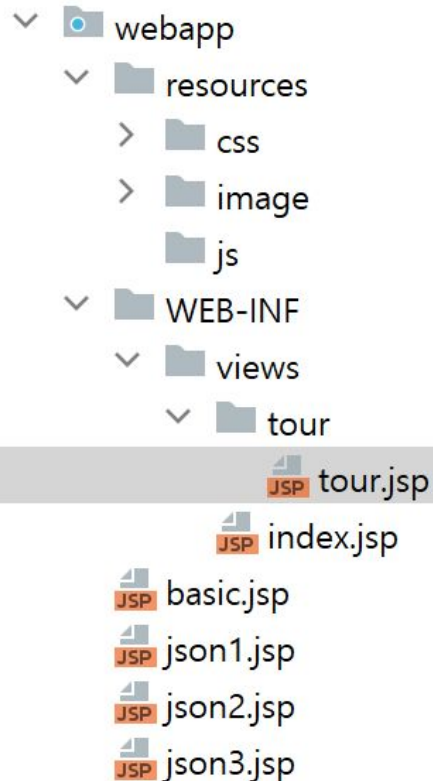
      jsonArray.forEach(function(json) {
        let jsonElement = document.createElement('div');
        jsonElement.className = 'card col-12 col-md-6 offset-md-3';
        if(json.today == 'sun') {
          img_result = "<img src=resources/image/sun.png width='100' height='100'><br>";
        }
        jsonElement.innerHTML = '<div class="card-body"> + img_result +
          '<h5 class="card-title text-danger">Today: ' + json.today + '</h5> ' +
          '<p class="card-text">Temp: ' + json.temp + ' 도</p> ' +
          '<p class="card-text">hu: ' + json.hu + '</p> ' +
          '<p class="card-text">location: ' + json.location + '</p> ' +
          '</div>';
        resultDiv.appendChild(jsonElement);
      });
    })
    .catch(function(error) {
      console.error('Error fetching the tours.; error');
    });
}); //b2
```



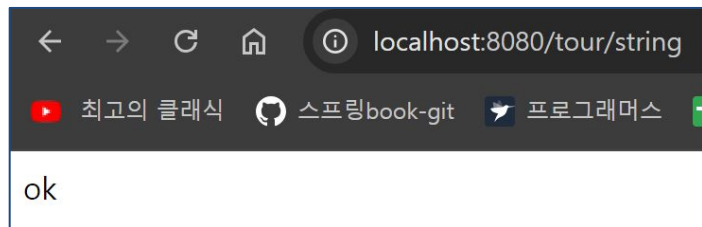
```
document.getElementById('b3').addEventListener('click', function() {
    // Axios로 JSON 데이터를 호출
    axios.get('json3.jsp')
        .then(function(response) {
            // 데이터를 받아서 result div에 출력
            var jsonArray = response.data;
            var resultDiv = document.getElementById('result');

            // 기존 내용을 지우기
            resultDiv.innerHTML = '';

            jsonArray.forEach(function(json) {
                const jsonElement = document.createElement('div');
                jsonElement.className = 'card col-12 col-md-6 offset-md-3';
                jsonElement.innerHTML = '<div class="card-body">' +
                    '<h5 class="card-title text-danger">Name: ' + json.name
+ '</h5>' +
                    '<p class="card-text">Age: ' + json.age + '세</p>' +
                    '<p class="card-text">Pic: <img src=resources/image/' +
json.pic + ' width=100 height="100"></p>' +
                    '</div>';
                resultDiv.appendChild(jsonElement);
            });
        })
        .catch(function(error) {
            console.error('Error fetching the tours:', error);
        });
    }); //b3
```



```
<head>
  <meta charset="UTF-8" >
  <meta name="viewport" content="width=device-width, initial-scale=1.0" >
  <title>Axios Example</ title>
  <!-- Bootstrap 5 CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" >
  <!-- Bootstrap Icons CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css"
rel="stylesheet" >
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js" ></script>
</head>
<body>
  <a href="tour/string" >tour/string</ a> <br>
  <hr color="red">
  <button id="b1" class="btn btn-primary" >tour/json1(vo)</ button>
  <button id="b2" class="btn btn-primary" >tour/json2(vo list)</ button>
  <button id="b3" class="btn btn-primary" >tour/json3(from db-table)</ button>
  <button id="b4" class="btn btn-primary" >tour/json3(from db-list)</ button>
  <div class="container mt-5" >
    <div id="result" class="row"></div>
  </div>
```



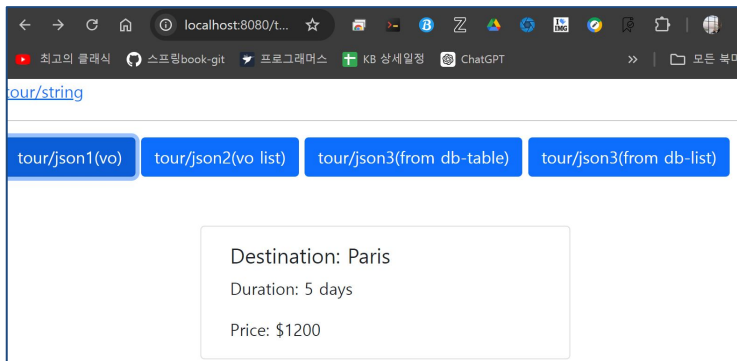
```
@Controller
@RequestMapping("/tour")
public class TourController {

    private final TourService tourService;

    @Autowired
    public TourController(TourService tourService) {
        this.tourService = tourService;
    }

    @GetMapping
    public String tour() {
        return "tour/tour";
    }

    @GetMapping("/string")
    @ResponseBody
    public String getTour() {
        return "ok";
    }
}
```

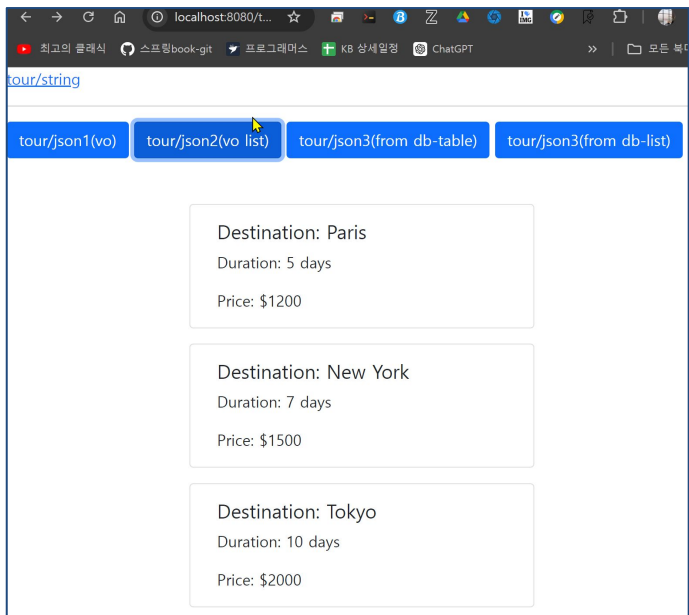


```
@GetMapping("/json1")
@ResponseBody
public TourVO getJson1() {
    TourVO tour = new TourVO();
    tour.setDestination("Paris");
    tour.setDuration(5);
    tour.setPrice(1200.0);
    return tour;
}
```

```
document.getElementById('b1').addEventListener('click', function() {
    // Axios로 JSON 데이터를 호출
    axios.get('/tour/json1')
        .then(function (response) {
            // 데이터를 받아서 result div에 출력
            var tour = response.data;
            console.log(tour);
            var resultDiv = document.getElementById('result');

            // 기존 내용을 지우기
            resultDiv.innerHTML = '';

            var tourElement = document.createElement('div');
            tourElement.className = 'card col-12 col-md-6 offset-md-3';
            tourElement.innerHTML = '<div class="card-body"> +
                '<h5 class="card-title">Destination: ' + tour.destination + '</h5>' +
                '<p class="card-text">Duration: ' + tour.duration + ' days</p>' +
                '<p class="card-text">Price: $ + tour.price + '</p>' +
                '</div>';
            resultDiv.appendChild(tourElement);
        })
        .catch(function (error) {
            console.error('Error fetching the tour:; error);
        });
}); //b1
```

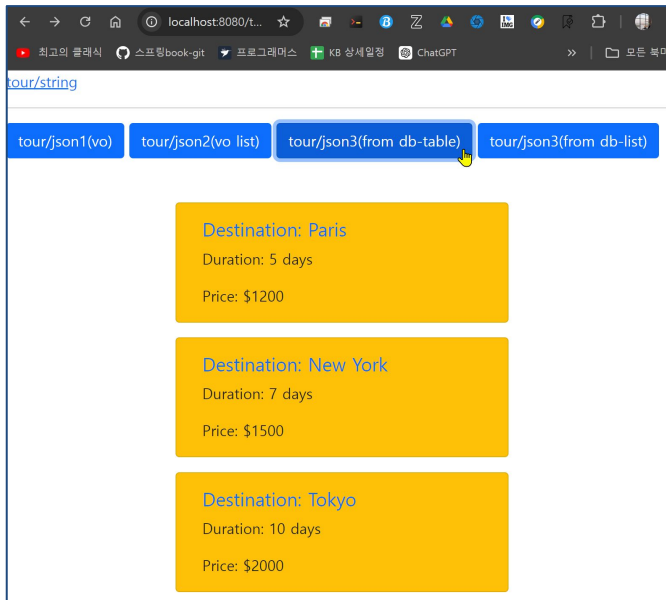


```
@GetMapping("/json2")
@ResponseBody
public List<TourVO> getTours() {
    return Arrays.asList(
        new TourVO("Paris", 5, 1200.0),
        new TourVO("New York", 7, 1500.0),
        new TourVO("Tokyo", 10, 2000.0)
    );
}
```

```
document.getElementById('b2').addEventListener('click', function() {
    // Axios로 JSON 데이터를 호출
    axios.get('/tour/json2')
        .then(function(response) {
            // 데이터를 받아서 result div에 출력
            var tours = response.data;
            var resultDiv = document.getElementById('result');

            // 기존 내용을 지우기
            resultDiv.innerHTML = '';

            tours.forEach(function(tour) {
                var tourElement = document.createElement('div');
                tourElement.className = 'card col-12 col-md-6 offset-md-3 mb-3';
                tourElement.innerHTML = '<div class="card-body">' +
                    '<h5 class="card-title">Destination: ' + tour.destination + '</h5>' +
                    '<p class="card-text">Duration: ' + tour.duration + ' days</p>' +
                    '<p class="card-text">Price: $ ' + tour.price + '</p>' +
                    '</div>';
                resultDiv.appendChild(tourElement);
            });
        })
        .catch(function(error) {
            console.error('Error fetching the tours: ' + error);
        });
}); //b2
```

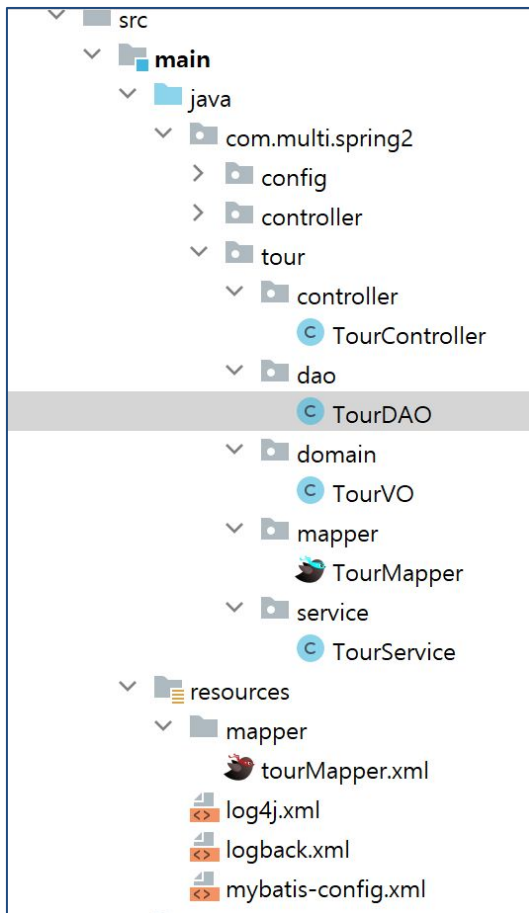


```
@GetMapping("/json3")
@ResponseBody
public List<TourVO> getTours2() {
    return tourService.all();
}
```

```
document.getElementById('b3').addEventListener('click', function() {
    // Axios로 JSON 데이터를 호출
    axios.get('/tour/json3')
        .then(function (response) {
            // 데이터를 받아서 result div에 출력
            var tours = response.data;
            var resultDiv = document.getElementById('result');

            // 기존 내용을 지우기
            resultDiv.innerHTML = '';

            tours.forEach(function(tour) {
                var tourElement = document.createElement('div');
                tourElement.className = 'card col-12 col-md-6 offset-md-3 mb-3 bg-warning';
                tourElement.innerHTML = '<div class="card-body">' +
                    '<h5 class="card-title text-primary">Destination: ' + tour.destination + '</h5>' +
                    '<p class="card-text">Duration: ' + tour.duration + ' days</p>' +
                    '<p class="card-text">Price: $' + tour.price + '</p>' +
                    '</div>';
                resultDiv.appendChild(tourElement);
            });
        })
        .catch(function (error) {
            console.error('Error fetching the tours:', error);
        });
}); //b3
```

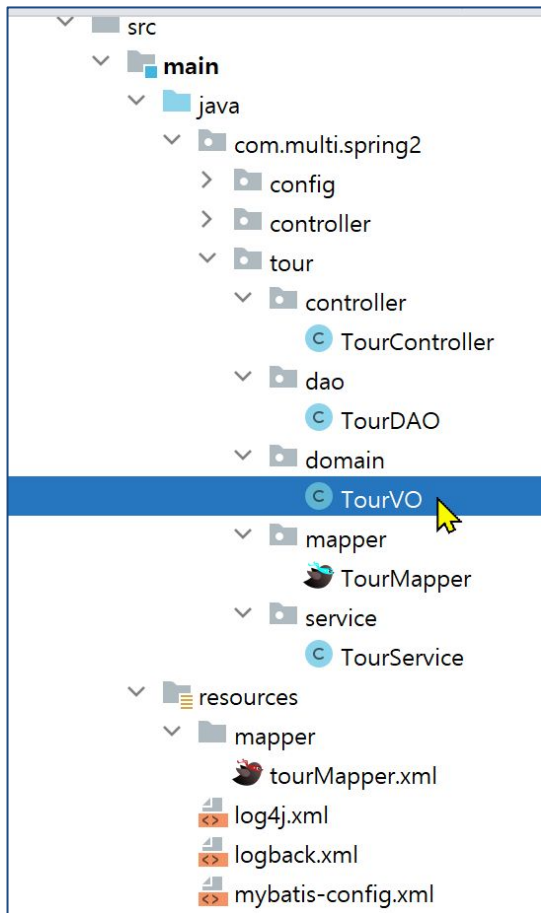



```
@Repository
@Slf4j
public class TourDAO {

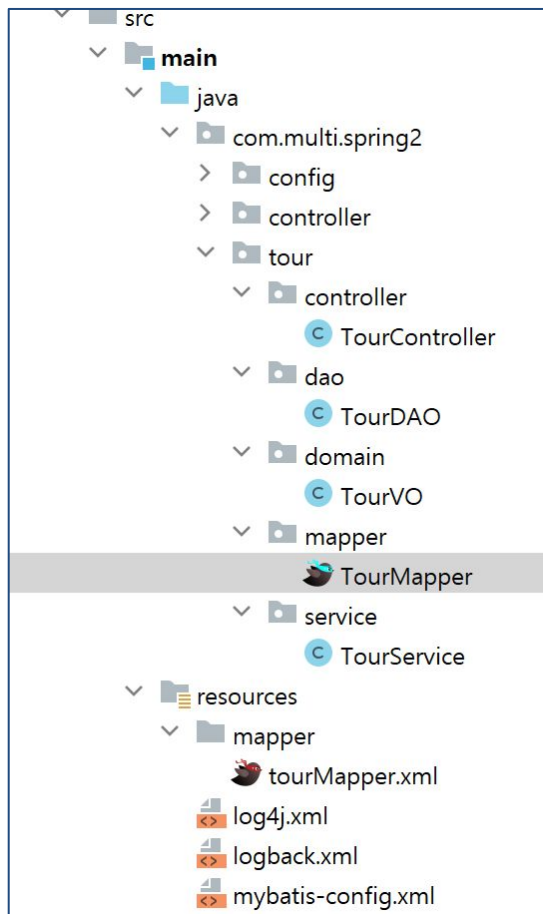
    //private static final Logger logger =
    LoggerFactory.getLogger(MemberDAO.class);
    private final SqlSessionTemplate sqlSessionTemplate;

    @Autowired
    public TourDAO(SqlSessionTemplate sqlSessionTemplate) {
        this.sqlSessionTemplate = sqlSessionTemplate;
    }

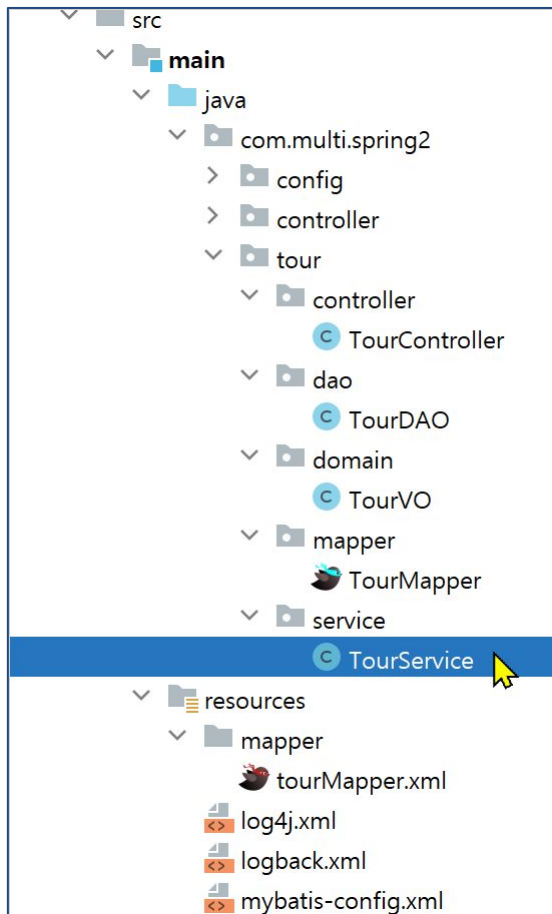
    public List<TourVO> all() {
        return
        sqlSessionTemplate.getMapper(TourMapper.class).all();
    }
}
```



```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class TourVO {
    private String destination;
    private int duration; // in days
    private double price;
}
```



```
@Mapper
public interface TourMapper {
    List<TourVO> all();
}
```

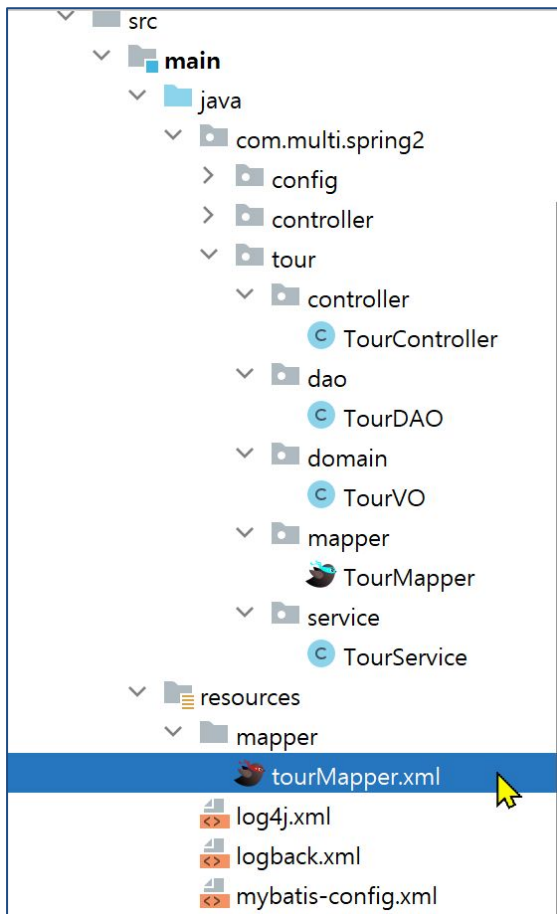


```
@Service
@Slf4j
public class TourService {

    private TourDAO tourDAO;

    @Autowired
    public TourService(TourDAO tourDAO) {
        this.tourDAO = tourDAO;
    }

    public List<TourVO> all() {
        log.debug("---->> ");
        return tourDAO.all();
    }
}
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper
    namespace="com.multi.spring2.tour.mapper.TourMapper" >

    <select id="all"

        resultType="com.multi.spring2.tour.domain.TourVO" >
        SELECT * FROM tour;
    </select>

</mapper>
```

localhost:8080/t...
최고의 클래식 스프링book-git 프로그래머스 KB 상세일정 ChatGPT >> 모든 북마크

tour/string

tour/json1(vo) tour/json2(vo list) tour/json3(from db-table) tour/json3(from db-list)

#	Destination	Duration	Price
1	Paris	5 days	\$1200
2	New York	7 days	\$1500
3	Tokyo	10 days	\$2000
4	London	4 days	\$1100
5	Sydney	8 days	\$1800
6	Rome	6 days	\$1300
7	Berlin	5 days	\$1150
8	Dubai	7 days	\$1600
9	Hong Kong	9 days	\$1750
10	Amsterdam	5 days	\$1250

```

@GetMapping("/json3")
@ResponseBody
public List<TourVO> getTours2() {
    return tourService.all();
}

```

```

document.getElementById('b4').addEventListener('click', function() {
    // Axios로 JSON 데이터를 호출
    axios.get('/tour/json3')
        .then(function (response) {
            // 데이터를 받아서 result div에 테이블 형태로 출력
            var tours = response.data;
            var resultDiv = document.getElementById('result');

            // 기존 내용을 지우기
            resultDiv.innerHTML = '';

            var table = document.createElement('table');
            table.className = 'table table-striped';

            var thead = document.createElement('thead');
            thead.innerHTML = '<tr>' +
                '<th scope="col">#</th>' +
                '<th scope="col">Destination</th>' +
                '<th scope="col">Duration</th>' +
                '<th scope="col">Price</th>' +
                '</tr>';
            table.appendChild(thead);

            var tbody = document.createElement('tbody');
            tours.forEach(function(tour, index) {
                var row = document.createElement('tr');
                row.innerHTML = '<th scope="row">' + (index + 1) + '</th>' +
                    '<td><i class="bi bi-geo-alt-fill me-2"></i>' + tour.destination + '</td>' +
                    '<td>' + tour.duration + ' days</td>' +
                    '<td>$' + tour.price + '</td>';
                tbody.appendChild(row);
            });
            table.appendChild(tbody);

            resultDiv.appendChild(table);
        })
        .catch(function (error) {
            console.error('Error fetching the tours:', error);
        });
}); //b4

```

@RestController

- 목적: @Controller와 @ResponseBody를 결합한 것
- 사용: RESTful 웹 서비스를 만들 때 사용
- 동작: 이 애노테이션이 붙은 클래스의 모든 메서드는 기본적으로 데이터를 직접 반환
(주로 JSON이나 XML 형태로)

```
@RestController
@RequestMapping("/tour2")
public class TourController2 {

    private final TourService tourService; 2 usages

    @Autowired
    public TourController2(TourService tourService) { this.tourService = tourService; }

    @GetMapping
    public ModelAndView tour() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("tour2/tour2");
        return mav;
    }

    @GetMapping("/string")
    public String getTour() { return "ok"; }
```

RestController에서
view를 호출하려면
ModelAndView사용해야
함.

```
@GetMapping("/json1")
public TourV0 getJson1() {
    TourV0 tour = new TourV0();
    tour.setDestination("Paris");
    tour.setDuration(5);
    tour.setPrice(1200.0);
    return tour;
}

@GetMapping("/json2")
public List<TourV0> getTours() {
    return Arrays.asList(
        new TourV0( destination: "Paris", duration: 5, price: 1200.0),
        new TourV0( destination: "New York", duration: 7, price: 1500.0),
        new TourV0( destination: "Tokyo", duration: 10, price: 2000.0)
    );
}

@GetMapping("/json3")
public List<TourV0> getTours2() { return tourService.all(); }
```

Log

- Spring Framework는 다양한 로깅 라이브러리와 통합이 잘 되어 있음
- 기본적으로 SLF4J를 사용하여 로깅을 처리
- 일반적으로는 Logback이나 Log4j2를 많이 사용
 - implementation 'org.springframework.boot:spring-boot-starter-log4j2'
 - application.properties 예시
 - logging.level.root=INFO
 - logging.level.com.example=DEBUG
 - logback-spring.xml

```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

- 로그를 파일에 출력하려면 설정 파일에 파일 아웃풋을 위한 **appender**를 추가해야 함.

```
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>logs/spring-boot.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>logs/spring-boot-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
      <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
        <maxFileSize>10MB</maxFileSize>
      </timeBasedFileNamingAndTriggeringPolicy>
      <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

- 로깅을 위한 코드는 아래와 같이 작성

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

@Service
public class MyService {

    private static final Logger logger = LoggerFactory.getLogger(MyService.class);

    public void performAction() {
        logger.info("Action is being performed");
        try {
            // Some business logic
        } catch (Exception e) {
            logger.error("An error occurred", e);
        }
    }
}
```

- 로깅 성능 최적화
- 로그를 너무 많이 출력하면 성능에 악영향을 미칠 수 있음. 최적화 방법을 고려해야 함.
 - 필요한 로그 레벨만 출력하도록 설정.
 - 로그 파일의 크기와 보관 기간을 제한.
 - 비동기 로깅을 통해 로그 출력이 메인 프로세스를 차단하지 않도록 설정(Logback의 경우 AsyncAppender 사용).

- SLF4J(Simple Logging Facade for Java)는 자바에서 다양한 로깅 프레임워크(Logback, Log4j, Java Util Logging 등)를 사용할 수 있게 하는 일종의 추상화 계층
- SLF4J를 사용하면 특정 로깅 프레임워크에 종속되지 않고, 런타임 시에 사용할 로깅 프레임워크를 선택할 수 있음.
 - 로깅 추상화
 - SLF4J는 다양한 로깅 프레임워크에 대해 통일된 **API**를 제공
 - 따라서, 애플리케이션 코드에서 **SLF4J API**만 사용하면 나중에 로깅 프레임워크를 교체하거나 설정을 변경할 때 코드 수정이 필요 없음.
 - 경량
 - SLF4J 자체는 매우 가볍고, 로깅 프레임워크와의 의존성을 최소화
 - 런타임 바인딩
 - SLF4J는 런타임에 사용할 로깅 프레임워크를 바인딩할 수 있음. 예를 들어, Logback, Log4j 2 등을 선택할 수 있음.

- SLF4J를 사용하기 위해서는 먼저 **SLF4J API**를 의존성으로 추가하고, 원하는 로깅 구현체 (Logback, Log4j 등)를 바인딩으로 추가해야 함.

```
implementation 'org.slf4j:slf4j-api:2.0.0'  
implementation 'ch.qos.logback:logback-classic:1.2.11'
```

- 로그 메시지 작성 : SLF4J는 다양한 로그 레벨을 제공
 - `logger.trace("Trace level message");`
 - `logger.debug("Debug level message");`
 - `logger.info("Info level message");`
 - `logger.warn("Warning level message");`
 - `logger.error("Error level message");`
- SLF4J는 파라미터를 이용한 로그 메시지 작성도 지원하여 성능과 코드 가독성을 높일 수 있음

```
int userId = 12345;  
String status = "active";  
  
logger.info("User with ID {} has status {}", userId, status);
```

- SLF4J는 단독으로 로그를 기록하지 않으며, 실제로 로그를 기록하는 것은 Logback, Log4j 2 등
 - 이를 위해 **SLF4J**는 바인딩 라이브러리를 통해 특정 로깅 구현체와 연결됨.
 - Logback과 바인딩: **logback-classic**을 추가하면 SLF4J는 Logback을 사용함.
 - Log4j 2와 바인딩: SLF4J와 Log4j 2를 함께 사용하려면 **log4j-slf4j-impl** 의존성을 추가해야 함.

- Logback과 Log4j는 둘 다 자바 애플리케이션에서 사용되는 대표적인 로깅 프레임워크
- 프레임워크는 유사한 점이 많지만, 몇 가지 차이점도 존재
- 공통점
 1. SLF4J 지원: Logback과 Log4j 모두 SLF4J(Simple Logging Facade for Java)를 지원. SLF4J는 로깅 API로, 런타임에 실제 로깅 프레임워크를 선택할 수 있게 해줌.
 2. 로깅 레벨: 두 프레임워크 모두 동일한 로깅 레벨을 지원(TRACE, DEBUG, INFO, WARN, ERROR, FATAL)
 3. 구성 파일: XML, JSON, YAML 등 다양한 형식의 설정 파일을 지원. 또한, 환경별로 다양한 설정을 적용할 수 있는 유연성을 제공
 4. 다양한 Appender: 콘솔, 파일, 네트워크, 데이터베이스 등 다양한 출력 매체를 지원하는 Appender들을 제공
 5. 필터 및 레이아웃: 로그 메시지를 출력하기 전 다양한 필터링 및 포매팅을 지원

- 차이점

특징	Logback	Log4j 2
성능	<ul style="list-style-type: none"> - Logback은 기존의 Log4j에 비해 성능 개선이 이루어짐. - 메모리 및 속도 측면에서 효율적 	<ul style="list-style-type: none"> - Log4j 2는 기존 Log4j 1.x의 성능 문제를 해결 - Logback과 유사한 수준으로 성능을 최적화
설정 파일	주로 logback.xml 또는 logback-spring.xml로 설정	주로 log4j2.xml, log4j2.properties, log4j2.yaml 등 다양한 형식의 설정 파일을 지원
재구성	Logback은 설정 파일이 변경되면 자동으로 재로딩을 지원	<ul style="list-style-type: none"> - Log4j 2도 설정 파일 변경 시 자동으로 재구성을 지원 - 설정 변경 시점에 대한 고급 옵션이 더 다양
비동기 로깅	AsyncAppender를 통해 비동기 로깅을 지원	<ul style="list-style-type: none"> - Log4j 2는 AsyncLogger를 통해 성능이 더 향상된 비동기 로깅을 지원 - LMAX Disruptor를 사용하여 더 빠르고 낮은 대기 시간의 비동기 로깅을 제공
커뮤니티 및 유지보수	<ul style="list-style-type: none"> - Logback은 SLF4J의 창시자인 Ceki Gülcü에 의해 개발 - 꾸준한 업데이트와 유지보수 	<ul style="list-style-type: none"> - Log4j 2는 Apache 재단에서 관리하며, 대규모 커뮤니티와 활발한 개발 - Log4j 1.x와의 호환성을 개선하고 있음
프로젝트 복잡도	<ul style="list-style-type: none"> - Logback은 Spring Boot에서 기본적으로 사용 - 설정이 비교적 간단. 	<ul style="list-style-type: none"> - Log4j 2는 더욱 다양한 기능을 제공 - 대규모 프로젝트나 특정 요구사항이 있는 프로젝트에 적합

- 언제 각각을 사용하면 좋은가?

- **Logback을 사용할 때:**

- **Spring Boot와의 통합**: Spring Boot는 기본적으로 Logback을 사용, 특별한 이유가 없다면 기본 설정을 사용하는 것이 유리. Spring Boot와 자연스럽게 통합되며 설정이 간단
- **단순한 설정**: 로그 설정이 비교적 간단하고, 기본적인 로깅 요구사항을 충족시킬 수 있을 때 Logback이 적합
- **성능과 안정성**: 성능이 중요하고, 안정적인 로깅 솔루션이 필요할 때 적합

- **Log4j 2를 사용할 때:**

- **고급 기능 필요**: 고급 비동기 로깅, 커스텀 필터 및 플러그인 등 Logback보다 더 많은 기능이 필요할 경우 Log4j 2가 유리
- **대규모 애플리케이션**: 로그를 대규모로 수집 및 분석해야 하거나, 매우 높은 성능과 낮은 대기 시간이 필요한 환경에서는 Log4j 2의 비동기 로깅 기능이 더 적합
- **커스터마이징**: 로그 출력이나 처리 과정에 대해 더 세밀한 설정과 커스터마이징이 필요할 때 Log4j 2가 더 나은 선택이 될 수 있음.

- Lombok은 로깅과 관련해서도 자주 사용되며, 특히 Log4j와 관련하여 중요한 역할
- Lombok은 로깅과 관련된 작업도 간단하게 할 수 있도록 다양한 어노테이션을 제공
 - 어노테이션을 사용하면 클래스에 로거(Logger)를 자동으로 생성해 줌.
 - `private static final Logger logger = LoggerFactory.getLogger(MyService.class);` 코드 필요 없음.
- 개발자는 Log4j를 사용할 때 매번 로거를 직접 초기화하는 코드를 작성할 필요 없이, Lombok 어노테이션을 통해 로거를 자동으로 생성할 수 있음.

- 주요 Lombok 어노테이션
 - **@Log4j**: Log4j 1.x 버전을 사용하는 로거를 자동 생성. 예를 들어, 클래스에 **@Log4j** 어노테이션을 붙이면 Lombok이 **org.apache.log4j.Logger** 타입의 **log** 변수를 자동으로 생성

```
import lombok.extern.log4j.Log4j;
```

```
@Log4j
```

```
public class MyService {  
    public void performTask() {  
        log.info("Task started");  
        // 비즈니스 로직  
        log.info("Task finished");  
    }  
}
```

- 주요 Lombok 어노테이션
 - **@Log4j2**: Log4j 2.x 버전을 사용하는 로거를 자동 생성. 이 어노테이션을 사용하면 `org.apache.logging.log4j.Logger` 타입의 `log` 변수를 자동으로 생성

```
import lombok.extern.log4j.Log4j2;
```

```
@Log4j2
```

```
public class MyService {  
    public void performTask() {  
        log.info("Task started");  
        // 비즈니스 로직  
        log.info("Task finished");  
    }  
}
```

- 주요 Lombok 어노테이션
 - **@Slf4j** : SLF4J를 기반으로 하는 로거를 생성

```
import lombok.extern.slf4j.Slf4j;

@Slf4j
public class MyService {
    public void performTask() {
        log.info("Task started");

        try {
            log.debug("Processing data...");
        } catch (Exception e) {
            log.error("Error occurred", e);
        }

        log.info("Task finished");
    }
}
```

```
// Logging
implementation 'org.slf4j:slf4j-api:1.7.32'
runtimeOnly 'org.slf4j:jcl-over-slf4j:1.7.32'
runtimeOnly 'org.slf4j:slf4j-log4j12:1.7.32'

// Logback Classic
implementation 'ch.qos.logback:logback-classic:1.2.11'
implementation 'ch.qos.logback:logback-core:1.2.11'
implementation 'commons-logging:commons-logging:1.2'

compileOnly 'org.projectlombok:lombok:1.18.28'
annotationProcessor 'org.projectlombok:lombok:1.18.28'
```

```
<configuration>
  <property name="LOG_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss} - %msg%n"/>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>${LOG_PATTERN}</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <file>C:/kb_spring/app.log</file>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="DB" class="com.multi.spring2.config.CustomDBAppender"/>

  <root level="debug">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
    <appender-ref ref="DB" />
  </root>
</configuration>
```



```
<log4j:configuration>
```

```
<!-- Appenders -->
```

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
```

```
<param name="Target" value="System.out"/>
```

```
<layout class="org.apache.log4j.PatternLayout">
```

```
<param name="ConversionPattern" value="%-5p %c - %m%n"/>
```

```
</layout>
```

```
</appender>
```

```
<!-- 파일에 로그 출력 -->
```

```
<appender name="FILE" class="org.apache.log4j.FileAppender">
```

```
<param name="File" value="c:\\kb_spring\\lapp.log"/>
```

```
<param name="Append" value="true"/>
```

```
<layout class="org.apache.log4j.PatternLayout">
```

```
<param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} - %m%n"/>
```

```
</layout>
```

```
</appender>
```

```
<!-- Application Loggers -->
```

```
<logger name="com.multi">
```

```
<level value="debug"/>
```

```
<appender-ref ref="FILE"/>
```

```
</logger>
```

```
-
```

```
<!-- Root Logger -->
```

```
<root>
```

```
<level value="debug"/>
```

```
<appender-ref ref="console"/>
```

```
<appender-ref ref="FILE"/>
```

```
</root>
```

```
</log4j:configuration>
```

```
public class CustomDBAppender extends AppenderBase<ILoggingEvent> {
    private BasicDataSource dataSource;

    @Override
    public void start() {
        dataSource = new BasicDataSource();
        dataSource.setUrl("jdbc:mysql://localhost:3306/shop2");
        dataSource.setUsername("root");
        dataSource.setPassword("1234");
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        super.start();
    }

    @Override
    protected void append(ILoggingEvent event) {
        String sql = "INSERT INTO logs (timestamp, level, logger, message, thread, exception) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection connection = dataSource.getConnection();
            PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setLong(1, event.getTimestamp());
            stmt.setString(2, event.getLevel().toString());
            stmt.setString(3, event.getLoggerName());
            stmt.setString(4, event.getFormattedMessage());
            stmt.setString(5, event.getThreadName());
            stmt.setString(6, event.getThrowableProxy() != null ? event.getThrowableProxy().getMessage() : null);
            stmt.executeUpdate();
        } catch (SQLException e) {
            addError("Failed to insert log entry", e);
        }
    }
}
```

```
use shop2;
```

```
CREATE TABLE `logs` (  
  `id` bigint NOT NULL AUTO_INCREMENT,  
  `timestamp` bigint NOT NULL,  
  `level` varchar(10) NOT NULL,  
  `logger` varchar(255) NOT NULL,  
  `message` text NOT NULL,  
  `thread` varchar(255) NOT NULL,  
  `exception` text,  
  `created_at` timestamp NULL DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=30124 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```




Table Name: logs
 Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
 Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
timestamp	BIGINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
level	VARCHAR(10)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
logger	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
message	TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
thread	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
exception	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
created_at	TIMESTAMP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP

```
@Service
@Slf4j
public class TourService {

    private TourDAO tourDAO;

    @Autowired
    public TourService(TourDAO tourDAO) {
        this.tourDAO = tourDAO;
    }

    public List<TourVO> all() {
        log.debug("--->> ");
        return tourDAO.all();
    }
}
```

```
@Repository
@Slf4j
public class TourDAO {

    //private static final Logger logger =
    LoggerFactory.getLogger(MemberDAO.class);

    private final SqlSessionTemplate sqlSessionTemplate;

    @Autowired
    public TourDAO(SqlSessionTemplate sqlSessionTemplate) {
        this.sqlSessionTemplate = sqlSessionTemplate;
    }

    public List<TourVO> all() {
        return sqlSessionTemplate.getMapper(TourMapper.class).all();
    }
}
```

1 • **SELECT** * **FROM** shop2.logs;

ult Grid Filter Rows: Edit: Export/Import: Wrap Cell Content: Fetch rows:			
id	timestamp	level	logger
36	1723165910827	DEBUG	org.springframework.jndi.JndiTemplate
37	1723165910833	DEBUG	org.springframework.jndi.JndiLocatorDelegate
38	1723165910839	DEBUG	org.springframework.jndi.JndiTemplate
39	1723165910850	DEBUG	org.springframework.jndi.JndiPropertySource
40	1723165910856	DEBUG	org.springframework.jndi.JndiTemplate
41	1723165910862	DEBUG	org.springframework.jndi.JndiLocatorDelegate
42	1723165910868	DEBUG	org.springframework.jndi.JndiTemplate
43	1723165910873	DEBUG	org.springframework.jndi.JndiPropertySource
44	1723165910886	DEBUG	org.springframework.web.context.support.AnnotationConfigWebApplicationContext
45	1723165910892	DEBUG	org.springframework.web.context.support.AnnotationConfigWebApplicationContext
46	1723165910913	DEBUG	org.springframework.beans.factory.support.DefaultListableBeanFactory

1. Rest API
2. Restful API
3. @ResponseBody
4. axios요청
5. json, json array
6. log
 - a. slf4j
 - b. lombok log4j, log4j-jdbc
 - c. logback
7. appender
 - console, file, db