Spring Boot 3.2.8 REST 서비스

inky4832@daum.net

Rest(ful) 서비스

1. RESTful 서비스

개념

REST란 'Representational State Transfer'의 약자로 하나의 URI는 하나의 고유한 리소스와 연결되며 이 리소스를 GET/POST/PUT/DELETE등 HTTP 메서드로 제어하자는 개념.

REST는 가장 적합한 형식이 무엇이든 간에 서버와 클라이언트간에 리소스의 상태를 전달하는 것으로서 WWW와 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다.

특징

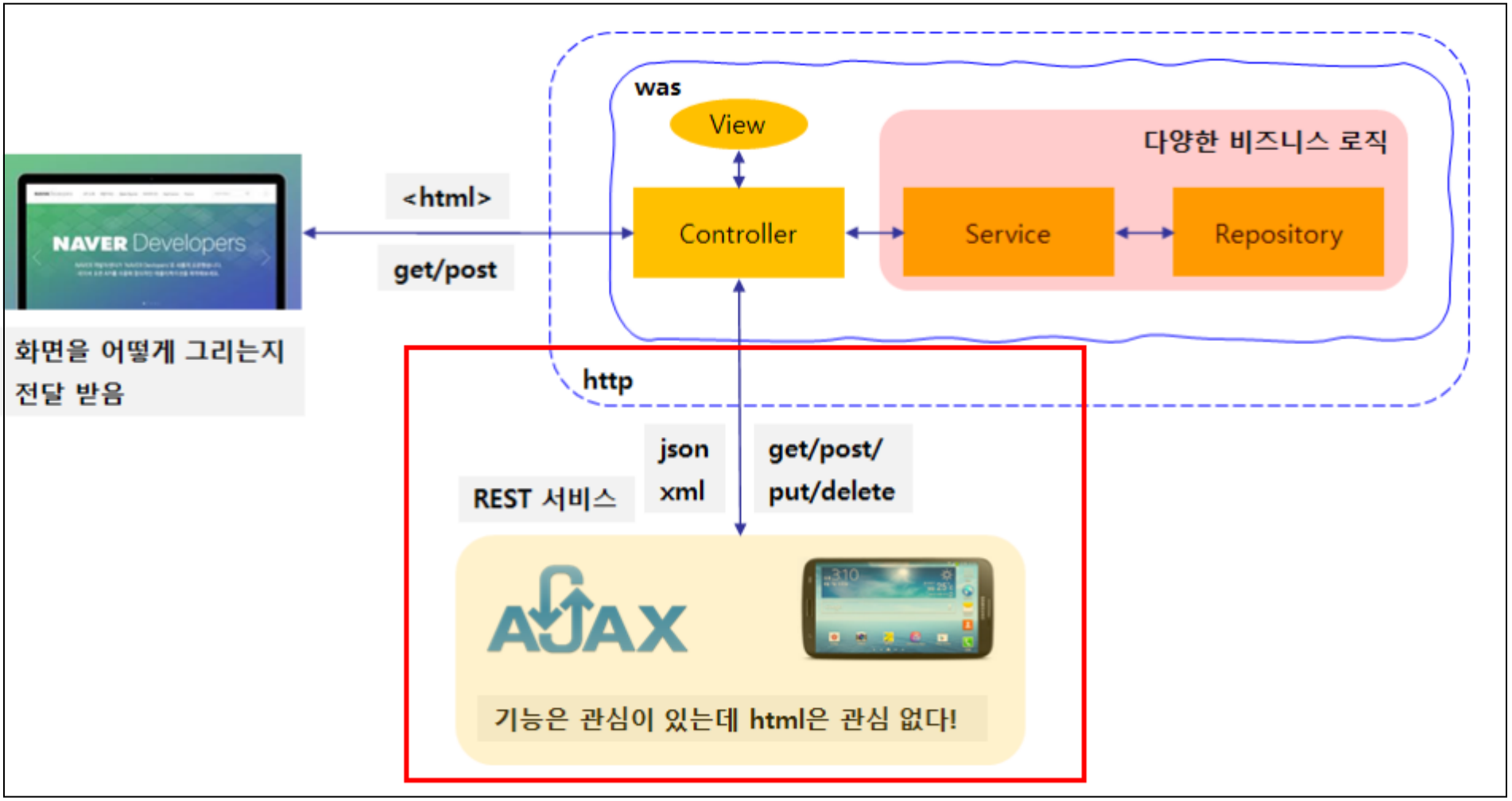
서버에 접근하는 클라이언트의 종류가 단순히 브라우저를 넘어 스마트폰, 다른 서비스등으로 다양해지면서 '화면에 대한 관심은 없고 데이터, 비즈니스 로직에만 관심 ' 있는 경우가 많아지는데 이때 사용하는 것이 REST 이다.

자바나 C와 같은 언어에서 어떤 기능을 제공하는 것을 API라고 하듯이 REST 형태로 사용자가 원하는 기능을 제공하는 것을 REST API라고 한다.

최근에 많이 활용되는 공공 API들이 대표적인 예이다.

또한 REST 방식으로 서비스를 제공하는 것을 'RESTful 하다' 라고 말한다.

아키텍쳐


was
View
다양한 비즈니스 로직
<html>
Controller
Service
Repository
get/post
화면을 어떻게 그리는지
전달 받음
http
json
xml
get/post/
put/delete
REST 서비스
AJAX
기능은 관심이 있는데 html은 관심 없다!

RESTless 컨트롤러 분석

구현 코드

```
@RequestMapping(value = "/delMember", method = RequestMethod.GET)
public String delete(Locale locale, Model model) {

    return "say";
}
```

코드 분석

- delete 메서드명
  => 컨트롤러가 리소스 지향이 아니라 액션 지향이라는 사실을 나타낸다.

- "/delMember "
  => 특별한 유즈케이스에 초점을 맞췄다는 사실을 암시한다.

http://localhost:8090/Context명/delMember?id=123

RESTful URL 특징

RESTless 계열과 반대로 RESTful은 HTTP가 리소스에 대한 모든 내용을 담고 있다.

DELETE http://localhost:8090/Context명/리소스명/123

GET http://localhost:8090/Context명/리소스명/123

위 URL에서 명확하지 않은 점은 무엇을 수행하는지에 대한 내용이다. 그 이유는 URL이 아무것도 수행하지 않기 때문이다. 대신에 리소스를 식별한다. 리소스명이 리소스의 위치를 나타낸다. 이 리소스로 무엇을 할지는 HTTP메소드가 결정한다.

쿼리 파라미터를 이용해 리소스를 식별하는 대신에 전체 기본 URL이 리소스를 식별한다. RESTful URL은 경로가 파라미터화 된다.

RESTless URL은 쿼리 파라미터에서 입력을 받는 반면에 RESTful URL의 입력은 URL 경로의 일부분이다.

REST 동사

앞서 언급했듯이 REST는 리소스 상태의 전달에 대한 것이다.
따라서 실제로 이러한 리소스에 액션을 취하기 위해 몇 가지 동사가 필요하다.
즉, 리소스의 상태를 전달하는 동사가 필요하다.

메소드 명	설명
GET	서비스에서 리소스 데이터를 조회한다. 리소스는 요청 URL에 의해 식별된다. (read)
POST	요청 URL을 리스닝하는 프로세서에 의해 처리되도록 서버에 데이터를 전송한다. (create)
PUT	요청 URL에 있는 서버에 리소스 데이터를 둔다. (update)
DELETE	요청 URL에 의해 식별되는 서버의 리소스를 삭제한다. (delete)

샘플

Http Method	동작	예
GET	전체 리소스에 대한 정보 획득	http://example.com/api/orders : 모든 주문 내역 조회
GET	특정 조건에 맞는 모든 정보 획득	http://example.com/api/orders?from=100 : 100번 주문부터 모든 주문 내역 조회
GET	특정 리소스에 대한 정보 획득	http://example.com/api/orders/123 : 123번 주문에 대한 상세 내역 조회
POST	새로운 리소스 저장	http://example.com/api/orders : 새로운 주문 생성. 파라미터는 request body로 전달
PUT	기존 리소스 업데이트	http://example.com/api/orders/123 : 123번 주문에 대한 수정. 파라미터는 request body로 전달
DELETE	기존 리소스 삭제	http://example.com/api/orders/123 : 123번 주문 삭제

응답 상태 코드값

상태 코드	설명
200 (OK)	OK
201 (Created)	요청이 성공적이었으며 그 결과로 새로운 리소스가 생성됨. POST|PUT 요청
204 (No Content)	요청에 대한 응답할 컨텐츠가 없을때, 예> PUT 요청해서 update하려고 했는데 컨텐츠가 없는 경우
400 (Bad Request)	이 응답은 잘못된 문법으로 인하여 서버가 요청을 이해할 수 없음을 의미
401 (Unauthorized)	HTTP 표준에서는 미승인(unauthorized)이지만 정확히는 미인증(unauthenticated)이다. 클라이언트에서는 요청한 응답을 받기 위해서는 반드시 인증해야 된다.
403 (Forbidden)	인증 안된 경우의 요청. 즉 클라이언트는 컨텐츠에 접근할 권한이 없음. 401과 다른점은 서버가 클라이언트가 누구인지 알고 있음
404 (Not Found)	
405 (Method Not Allowed)	
500 (Internal Server Error)	

URL 파라미터 (URI template)

파라미터화된 URL 경로를 가능하게 하기 위해서 스프링은 @PathVariable 어노테이션을 제공한다
@Pathvariable 은 취할 액션을 설명하는 대신에 리소스를 식별하는 URL에 대한 요청을 처리하는 컨트롤러 메소드 작성을 가능하게 한다.

구현

```
@RequestMapping(value="/cust/{userid}", method=RequestMethod.GET)
public ModelAndView sayEcho(@PathVariable String userid){
        return new ModelAndView("home");
}//end sayEcho
```

GET : http://localhost:8081context/cust/123 (123에 해당하는 고객 조회)
DELETE : http://localhost:8081/context/cust/123 (123에 해당하는 고객 삭제)

샘플

```
@GetMapping(value="/board/name/{xxx}")
public String aaa(@PathVariable("xxx") String name) {
    System.out.println(name);
    return "main";
}
@GetMapping(value="/board2/name/{xxx}/age/{yyy}" )
public String aaa2(@PathVariable("xxx") String name ,
                   @PathVariable("yyy") int age) {
    System.out.println(name+"\t"+ age);
    return "main";
}
@GetMapping(value="/board3/birthday/{xxx}" )
public String aaa3(@PathVariable("xxx")
                  @DateTimeFormat(iso = ISO.DATE)
                   Date d,
                   @RequestParam("price")
                   @NumberFormat(style=Style.CURRENCY,
                     pattern="###,###,##0.00")
                   BigDecimal price
                    ) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
    System.out.println(sdf.format(d));
    System.out.println(price);

    return "main";
```

7. @ResponseBody

개념

@ResponseBody가 메서드 레벨에 부여되면 메서드가 리턴하는 객체는 뷰를 통해 결과를 만들어내는 모델로 사용되는 대신에 메시지 컨버터를 통해 바로 HTTP 응답의 메시지 본문으로 전환된다.
기본적으로 @RequestBody 와 @ResponseBody는 XML 이나 JSON과 같은 메시지 기반의 커뮤니케이션을 위해 사용된다.

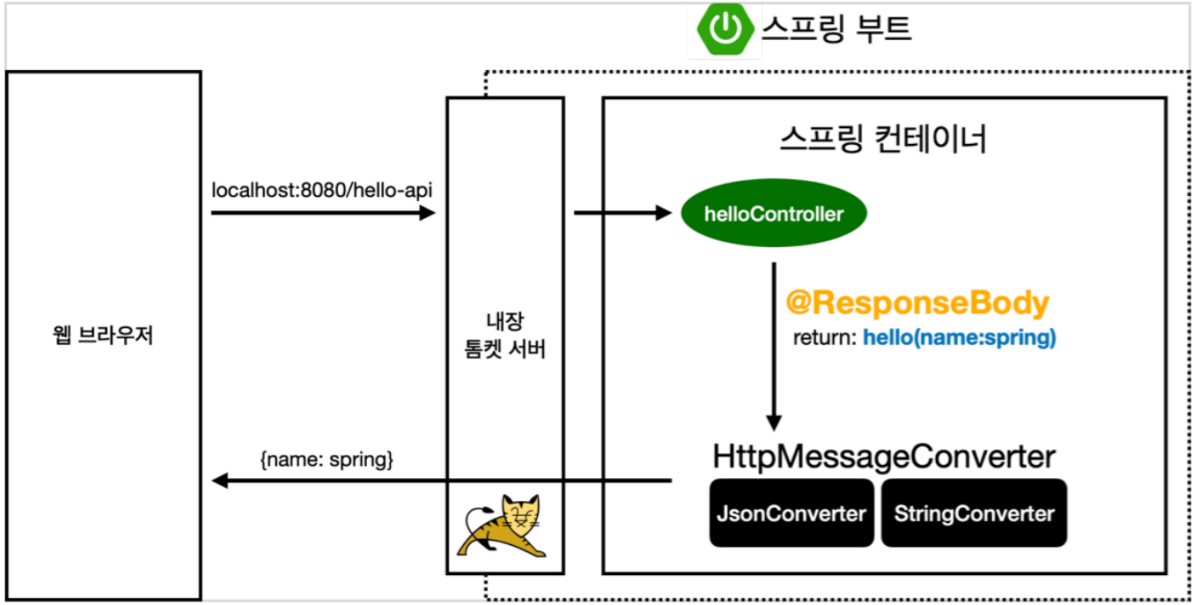
구현

```
@Controller
public class TestController {

  @RequestMapping("/hello")
   @ResponseBody
    public String hello(){
       return "HelloWorld";
    }
}
```

@ResponseBody가 없다면, String타입의 리턴값은 뷰이름(HelloWorld.jsp)으로 인식될 것이다.
하지만 @ResponseBody가 있기 때문에 HttpServletResponse의 출력 스트림으로 동작된다.

HttpMessageConverter



HTTP의 BODY에 문자 내용을 직접 반환.
viewResolver 대신에 HttpMessageConverter가 동작
기본문자처리: StringHttpMessageConverter
기본객체처리: MappingJackson2HttpMessageConverter
Byte처리 등등 기타 여러 HttpMessageConverter가 기본으로 등록되어 있음

참고: 클라이언트의 HTTP Accept 헤더정보와 서버의 Controller 반환 타입 정보 둘을 조합해서 HttpMessageConverter가 선택된다.

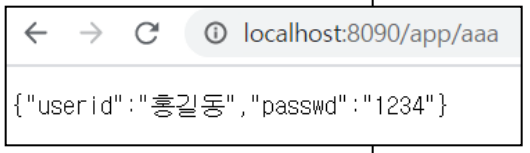
샘플

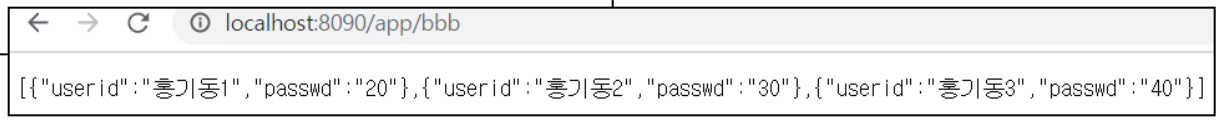
```java
@Controller
public class MainController {

    @RequestMapping("/main")
    public String main() {
        return "main";
    }

    @RequestMapping("/aaa")
    @ResponseBody
    public Login aaa() {
        Login login = new Login();
        login.setUserid("홍길동");
        login.setPasswd("1234");
        return login;
    }

    @RequestMapping("/bbb")
    @ResponseBody
    public ArrayList<Login> bbb() {
        ArrayList<Login> list = new ArrayList<Login>();
        list.add(new Login("홍기동1", "20"));
        list.add(new Login("홍기동2", "30"));
        list.add(new Login("홍기동3", "40"));
        return list;
    }
```





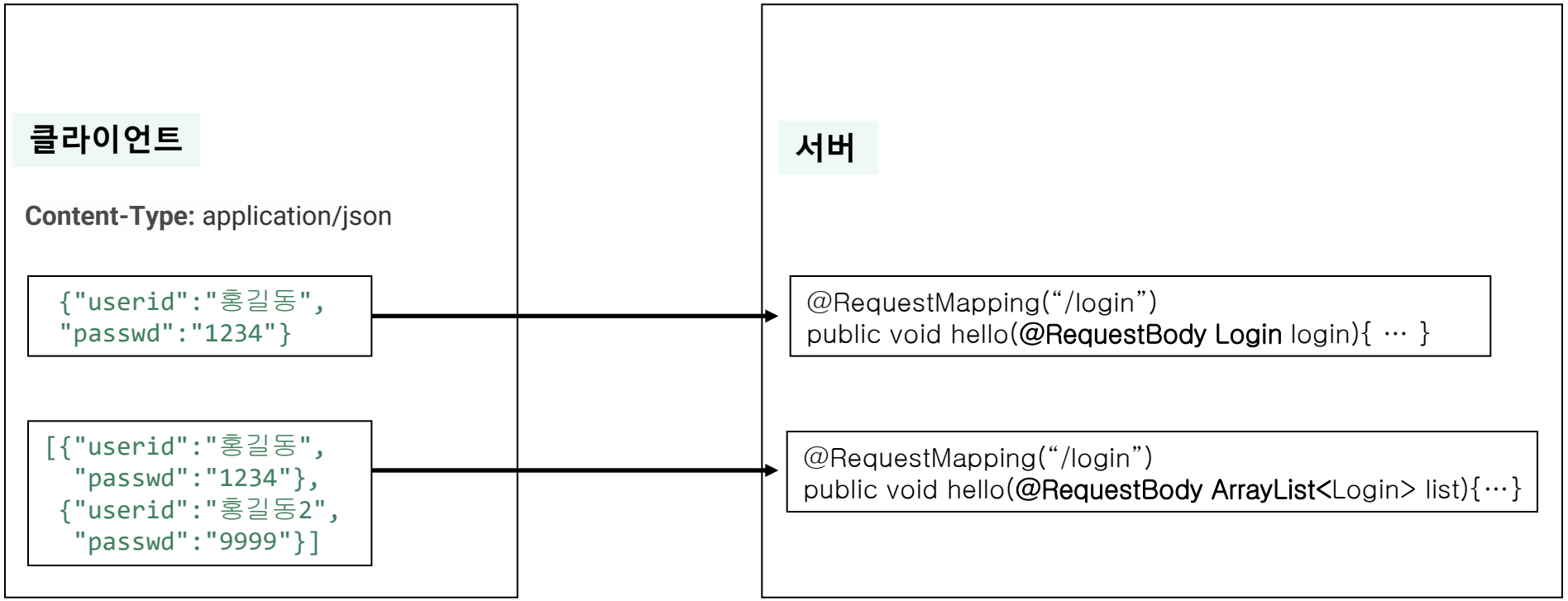
@Controller + @ResponseBody

@RestController

개념

@RequestBody가 메서드 파라미터에 지정되면 JSON 형식의 요청 파라미터가 자바 객체(빈, 컬렉션)에 직접 저장 가능하도록 변환되어 진다.

구현



개념

REST API를 구현할 때 클라이언트에게 정확한 응답상태를 반환하는 것은 매우 중요하다.
ResponseEntity 를 사용하면 200, 404 같은 응답 상태값 및 반환값을 한꺼번에 저장할 수 있는 메서드를 사용할 수 있다.

구현

```
ResponseEntity.created(null).build();
ResponseEntity.badRequest().build();
ResponseEntity.internalServerError().build();
ResponseEntity.noContent().build();
ResponseEntity.notFound().build();
ResponseEntity.ok().build();
ResponseEntity.status(200).build();
```

개념

REST API 컨슈머는 REST API를 이해해야 된다.
즉 노출되고 있는 여러 리소스가 무엇인지 알아야 됨 (resource 파악)
수행되는 여러 작업에 대해서도 알아야 됨. (action 파악)
요청 및 응답 구조는 물론 요청형식도 알아야 됨. (Request/Response Structure)
컨슈머가 원하는 응답은 무엇인지 제약이나 검증은 있는지 따져봐야 됨.
(Constrains/Validations)
결국 REST API 컨슈머는 REST API를 잘 이해하고 있어야 제대로 사용할 수 있다.

기능

개발자가 만든 REST API 서비스를 설계, 빌드, 문서화할 수 있도록 하는 오픈소스 프로젝트로서 대표적인 기능은 다음과 같다.

-API 디자인
-API 빌드
-API 문서화
-API 테스팅 (*)
-API 표준화

SpringBoot 2 의존성

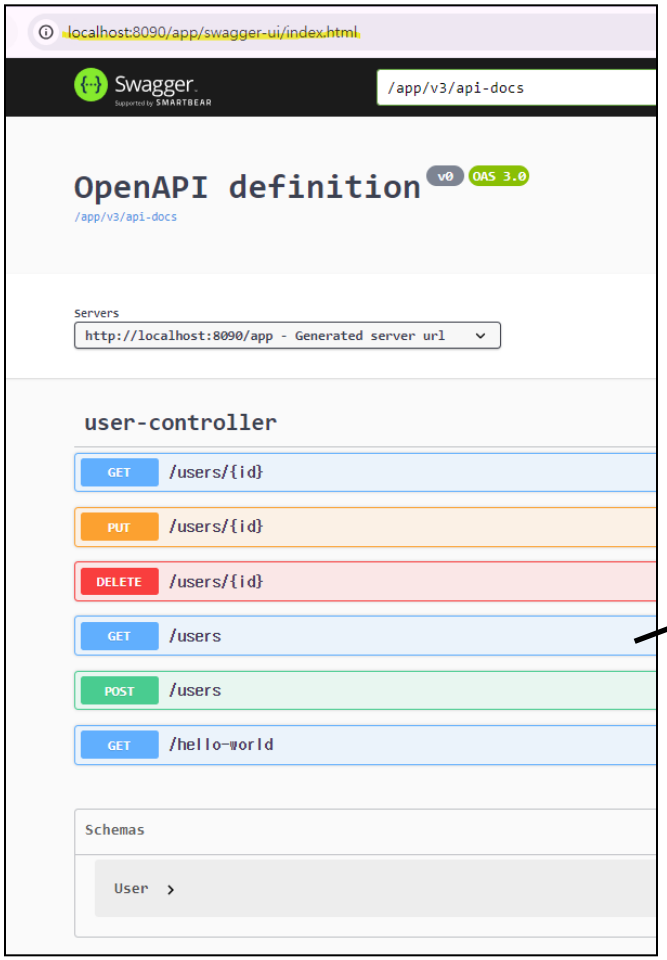
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.8.0</version>
</dependency>
```

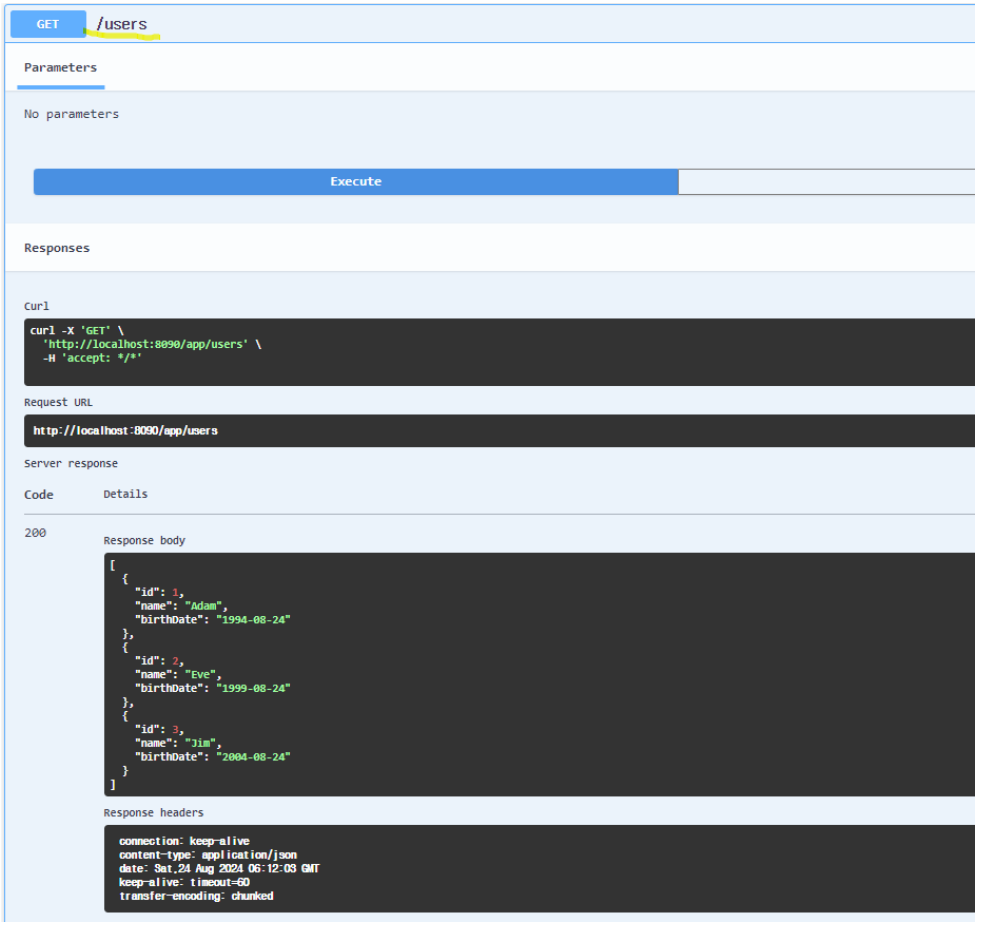
SpringBoot 3 의존성

```
 <dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.6.0</version>
 </dependency>
```

실행

http://localhost:8090/app/swagger-ui/index.html





Thank you