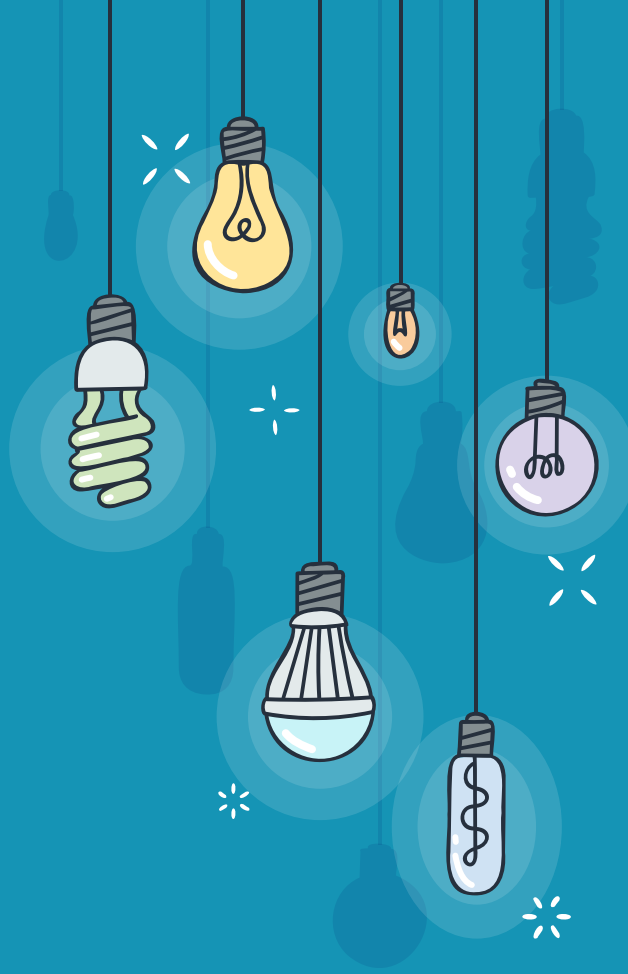
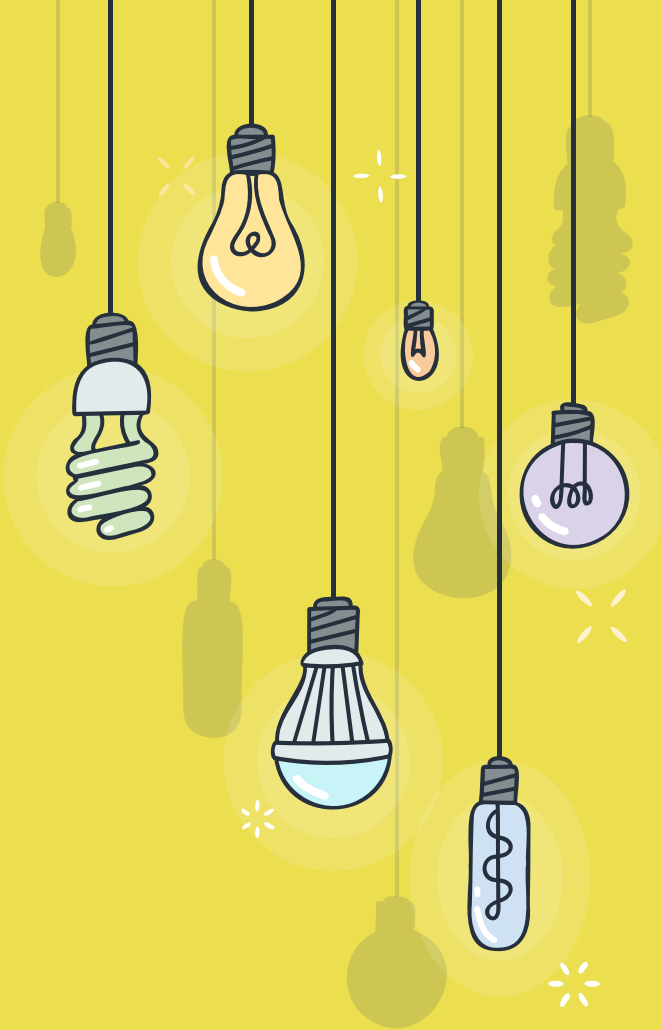
Spring boot

1. REST API의 개요

2. RESTFUL API의 개요



1 REST API 개요



REST API

- REST API(Representational State Transfer)란?
  - 자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것
    1. HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고,
    2. HTTP Method(POST, GET, PUT, DELETE, PATCH 등)를 통해
    3. 해당 자원(URI)에 대한 CRUD(Create, Read, Update, Delete) Operation을 적용하는 것을 의미

REST API의 장단점

- 장점
  - HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구축할 필요 없음
  - HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있다.
  - HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용 가능
  - REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악
  - 서버와 클라이언트의 역할을 명확하게 분리한다.

REST API의 장단점

단점

- 표준이 존재하지 않아 정의가 필요하다.
- HTTP Method 형태가 제한적이다.
- 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
- 구형 브라우저에서 호환이 되지 않아 지원해주지 못하는 동작이 많다

REST API의 규칙

- REST API 설계시 규칙
  - URI는 동사보다는 명사를, 대문자보다는 소문자를 사용한다

Bad) http://springboot.study.com/Running/
Good) http:// springboot.study.com /run/

  - 마지막에 슬래시 (/)를 포함하지 않는다.

Bad) http://springboot.study.com/test/
Good) http://springboot.study.com/test

REST API의 규칙

- 언더바 대신 하이픈을 사용한다.

Bad) http://springboot.study.com/test_blog
Good) http://springboot.study.com/test-blog

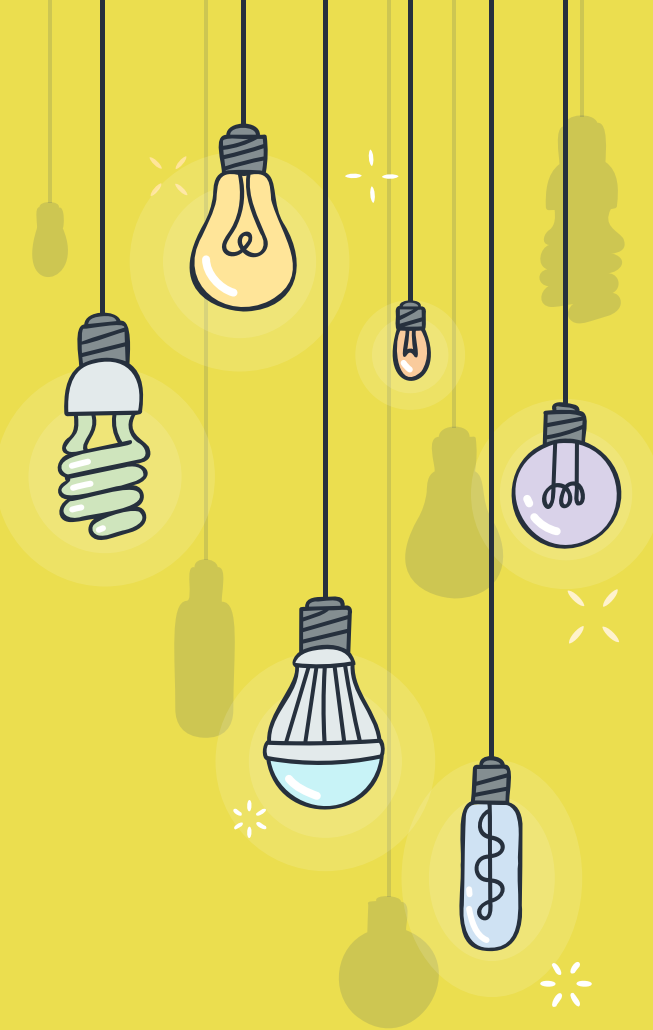
- 파일확장자는 URI에 포함하지 않는다.

Bad) http://springboot.study.com/photo.jpg
Good) http://springboot.study.com/photo

- 행위를 포함하지 않는다.

Bad) http://springboot.study.com/delete-post/1
Good) http://springboot.study.com/post/1

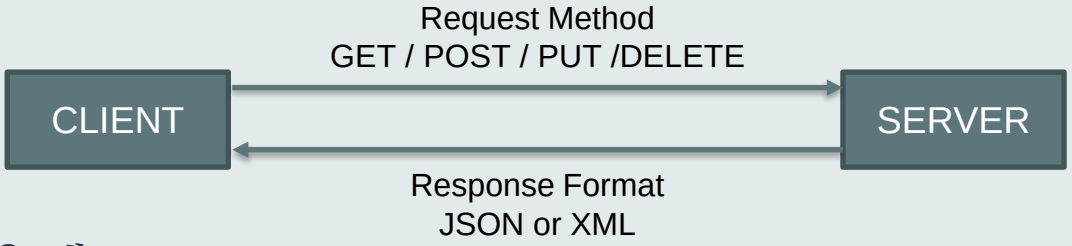
2 RESTFUL API 개요



RESTFUL API 개요

- RESTFUL API 개요
  - REST의 원리를 따르는 시스템을 의미
    - REST API의 설계 규칙을 올바르게 지킨 시스템을 RESTful하다 말할 수 있으며
    - REST API의 설계 규칙을 올바르게 지키지 못한 시스템은 REST API를 사용하였지만 RESTful 하지 못한 시스템이라고 할 수 있다
  - 일반적으로 JSON이나 XML 형식의 데이터를 반환

사용법



사용 예

HTTP Method	URI	Operation
GET	/member/users	모든 사용자의 List를 리턴
GET	/member/users/u01	ID가 u01인 특정 사용자 리턴
POST	/member/users	새로운 사용자 생성
PUT	/member/users/u02	ID가 u02인 특정 사용자를 업데이트
DELETE	/member/users/u03	ID가 u03인 특정 사용자를 삭제
DELETE	/member/users	전체 사용자 모두 삭제

사용법

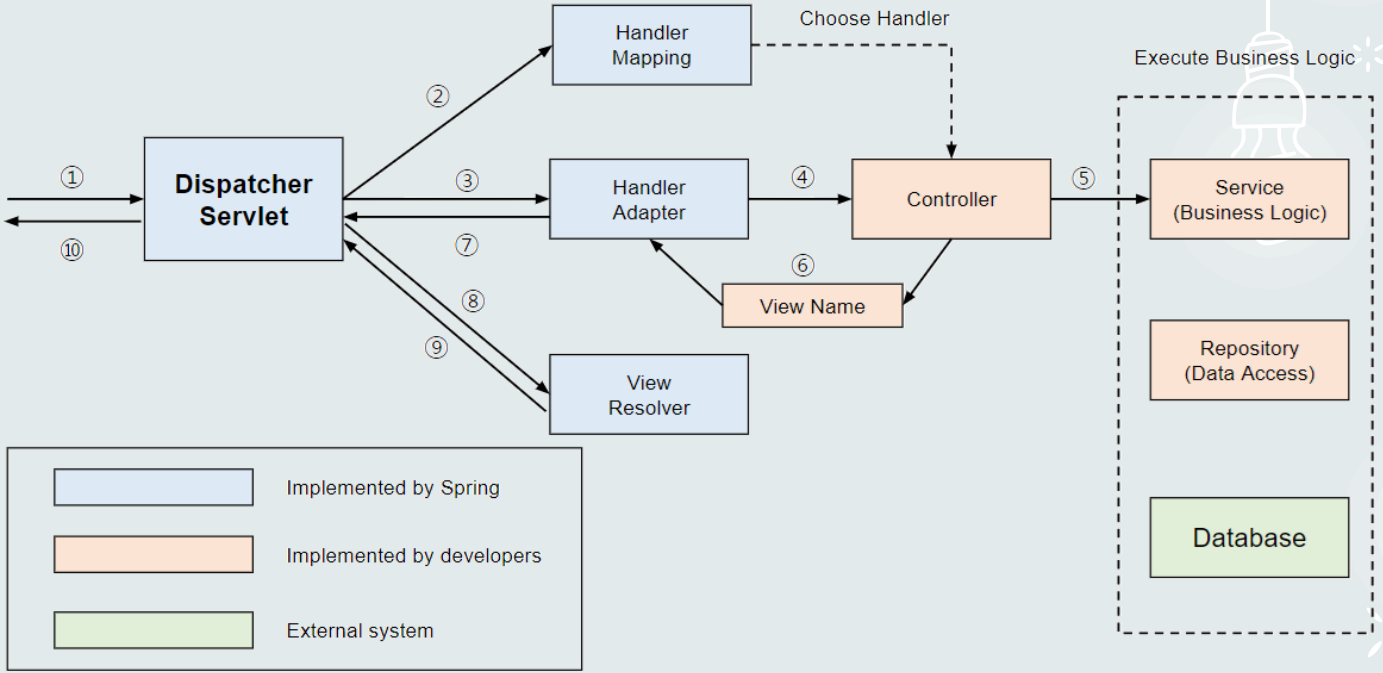
- @RestController 어노테이션
  - Spring4 부터 지원
  - @Controller + @ResponseBody
    - View를 갖지 않는 REST Data(JSON/SML)를 반환
- Path Variable 사용
  - RESTful 웹 어플리케이션에서 URL 경로에 있는 변수 값을 추출하는 데 사용

Url 경로 : https://example.com/products/123
이면,

```
@GetMapping("/products/{id}")
public Product getProductById(@PathVariable Long id) {
    return productService.getProductById(id);
}
```

@Controller vs @RestController

- Controller는 주로 View를 반환하기 위해 사용



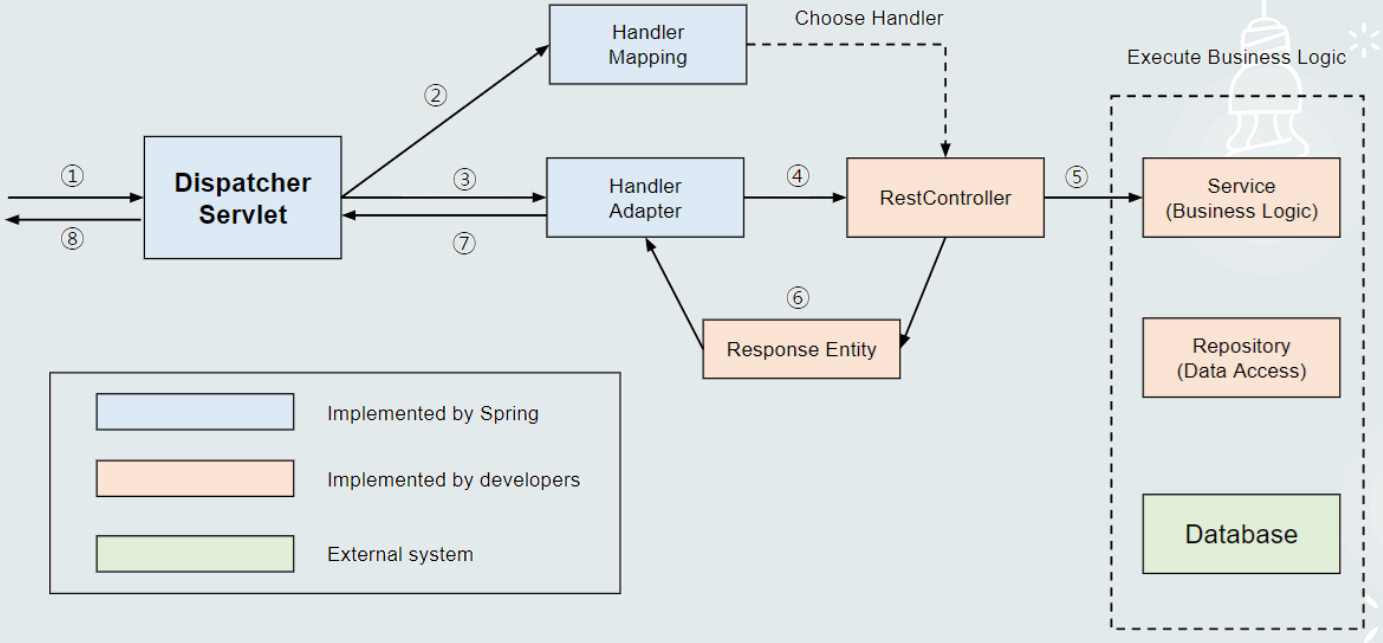
@Controller vs @RestController

Controller의 View처리 과정

1. Client는 URI 형식으로 웹 서비스에 요청을 보낸다.
2. DispatcherServlet이 요청을 처리할 대상을 찾는다.
3. HandlerAdapter을 통해 요청을 Controller로 위임
4. Controller는 요청을 처리한 후에 ViewName을 반환
5. DispatcherServlet은 ViewResolver를 통해 ViewName에 해당하는 View를 찾아 사용자에게 반환

@Controller vs @RestController

- RestController의 주용도는 Json 형태로 객체 데이터를 반환하는 것



@Controller vs @RestController

- RestController의 Data처리 과정

1. Client는 URI 형식으로 웹 서비스에 요청을 보낸다.
2. DispatcherServlet이 요청을 처리할 대상을 찾는다.
3. HandlerAdapter을 통해 요청을 Controller로 위임
4. RestController는 요청을 처리한 후에 객체를 반환
5. 반환되는 객체는 Json으로 Serialize되어 사용자에게 반환

@Controller vs @RestController

```
@RestController
public class TestRestController {

    @Autowired
    TestRepository repository;

    @GetMapping("/mypage")
    public ResponseEntity<Member> findUserWithResponseEntity(@RequestParam("id") String id){
        return ResponseEntity.ok(repository.findById(id).get());
    }
}
```

- findById는 Member객체를 그대로 반환
- 이러한 경우의 문제는 클라이언트가 예상하는 HttpStatus를 설정해줄 수 없다는 것
- 예를 들어 어떤 객체의 생성 요청이라면 201 CREATED를 기대할 것이지만 객체를 그대로 반환하면 HttpStatus를 설정해줄 수 없다.
- 그래서 객체를 상황에 맞는 ResponseEntity로 감싸서 반환해준다

Thanks!

- Any questions?