

CH13 스프링부트 컨트롤러 기본 - 응답

<https://github.com/codingspecialist/Springboot-Controller>

1. 스프링 컨트롤러 응답 방식 4가지

- View 응답
- Text 응답
- Object 응답
- ResponseEntity 응답

1. **View** : JSP, Freemarker, Thymeleaf 등의 뷰(View)를 사용하여 HTML 페이지를 반환하는 방식입니다. 이 방식은 주로 HTML 페이지를 반환하는 웹 애플리케이션에서 사용됩니다.

```
@GetMapping("/example")
public String example() {
    return "loginForm"; // ViewResolver가 발동하여 파일을 찾아서 text/html이 응답된다.
}
```

2. **Text** : 텍스트 데이터를 반환하는 방식입니다. 이 방식은 주로 텍스트 파일을 반환하는 애플리케이션에서 사용됩니다.

```
@GetMapping("/example")
@ResponseBody
public String example() {
    return "Hello, world!"; // text/html이 응답된다.
}
```

3. **Object** : Java 객체를 반환하는 방식입니다. 이 방식은 주로 JSON 형식으로 변환하여 반환됩니다.

```
@GetMapping("/example")
public ExampleDto getExample() {
    ExampleDto exampleDto = exampleService.getExample();
    return exampleDto; // application/json이 응답된다.
}
```

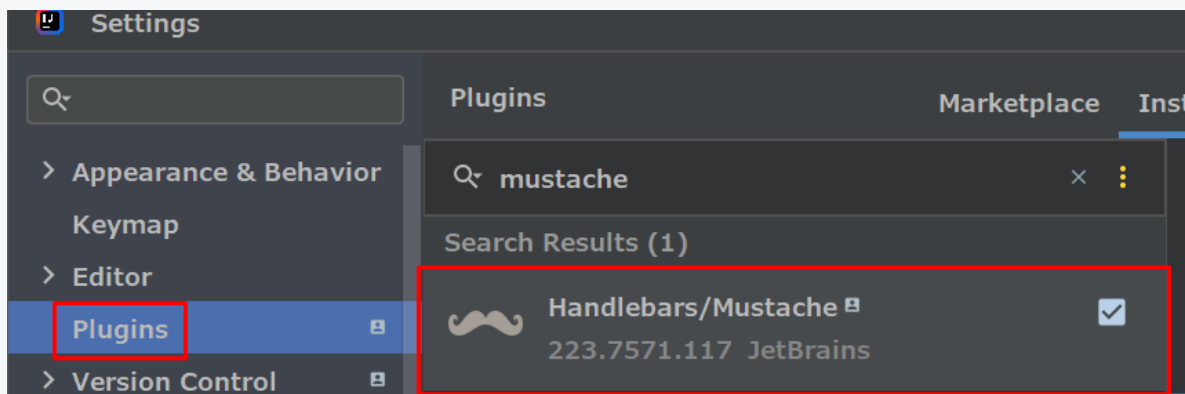
4. `ResponseEntity` : HTTP 응답 코드, 헤더, 본문 등을 직접 지정할 수 있는 방식입니다. 이 방식은 주로 RESTful API에서 HTTP 응답 코드와 메시지를 지정할 때 사용됩니다.

```
@GetMapping("/example/{id}")
public ResponseEntity<ExampleDto> getExample(@PathVariable Long id) {
    ExampleDto exampleDto = exampleService.getExample(id);
    if (exampleDto == null) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(exampleDto, HttpStatus.OK); // text/html과 http
status가 함께 응답된다.
}
```

2. View 응답

JSP 템플릿 엔진이 아닌, Mustache 템플릿 엔진을 사용하도록 하겠습니다.

우선 툴에서 mustache 파일을 인식할 수 있게 plugin을 설치합니다.



resources/templates/home.mustache

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <h1>Home</h1>
</body>
</html>

```

controller/ResponseController.java

@Controller 와 @RestController 는 각각 다른 목적을 가지며, 사용하는 방식도 다릅니다.
 @Controller 는 뷰(View)를 생성하거나 반환하는 컨트롤러를 정의할 때 사용하며, @RestController 는 데이터를 반환하는 RESTful 웹 서비스를 구현할 때 사용합니다.

```

package shop.mtcoding.conbasic.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class ResponseController {

    @GetMapping("/resp/v1")
    public String respV1(){
        return "home";
    }
}

```

새로운 mustache 파일이 생성되었기 때문에, 서버를 종료한 뒤 다시 실행해야 파일 인식이 됩니다.

Home

3. View 응답 - Model에 담기

HttpServletRequest 의 `setAttribute()` 메서드와 Model 의 `addAttribute()` 메서드는 모두 데이터를 저장하는 메서드이지만, 저장하는 위치와 범위에서 차이가 있습니다.

HttpServletRequest 의 `setAttribute()` 메서드는 HTTP 요청의 속성(attributes)을 설정하는 데 사용됩니다. 속성은 HTTP 요청을 처리하는 모든 컴포넌트에서 공유되며, 요청 범위(request scope) 내에서 유지됩니다. 즉, `setAttribute()` 메서드를 사용하여 설정한 속성은 해당 HTTP 요청을 처리하는 모든 컴포넌트에서 접근할 수 있습니다.

```
@GetMapping("/example")
public String example(HttpServletRequest request) {
    request.setAttribute("message", "Hello, world!");
    return "example";
}
```

Model 의 `addAttribute()` 메서드는 뷰(View)에서 사용할 데이터를 저장하는 데 사용됩니다. `addAttribute()` 메서드로 저장한 데이터는 뷰에서 참조할 수 있으며, 뷰 범위(view scope) 내에서 유지됩니다. 즉, `addAttribute()` 메서드로 저장한 데이터는 해당 요청에 대한 응답을 반환할 때까지 유지됩니다.

```
@GetMapping("/example")
public String example(Model model) {
    model.addAttribute("message", "Hello, world!");
    return "example";
}
```

따라서, HttpServletRequest 의 `setAttribute()` 메서드와 Model 의 `addAttribute()` 메서드는 저장하는 위치와 범위에서 차이가 있으며, 데이터를 저장하는 목적과 사용 방법도 다르게 사용됩니다.

HttpServletRequest 의 `setAttribute()` 메서드는 주로 HTTP 요청을 처리하는 컴포넌트 간에 데이터를 공유할 때 사용하며, Model 의 `addAttribute()` 메서드는 주로 뷰에서 참조할 데이터를 저장할 때 사용합니다.

```
@GetMapping("/resp/v2")
public String respV2(Model model){
    model.addAttribute("title", "제목1");
    return "main";
}
```

resources/templates/main.mustache 생성

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Main</h1>
    <hr/>
    <h3>{{title}}</h3>
</body>
</html>
```

mustache는 jsp와 다른 문법을 사용합니다.

jsp는 \${}

mustache는 {{}}

← → ↻ ⓘ localhost:8080/resp/v2

JUnit 유용한사이트 flutter 스프링부트

Main

제목1

4 Text 응답

스프링부트 컨트롤러에 메서드에서 String을 응답하면 text/html 이 응답되고 상태코드는 200이 응답된다.

그리고 문자 인코딩은 자동으로 UTF-8이 적용된다.

@Controller는 View를 응답하기 때문에 메서드에 return type 앞에 @ResponseBody를 붙여줘야 Text를 리턴하는 메서드로 변환된다.

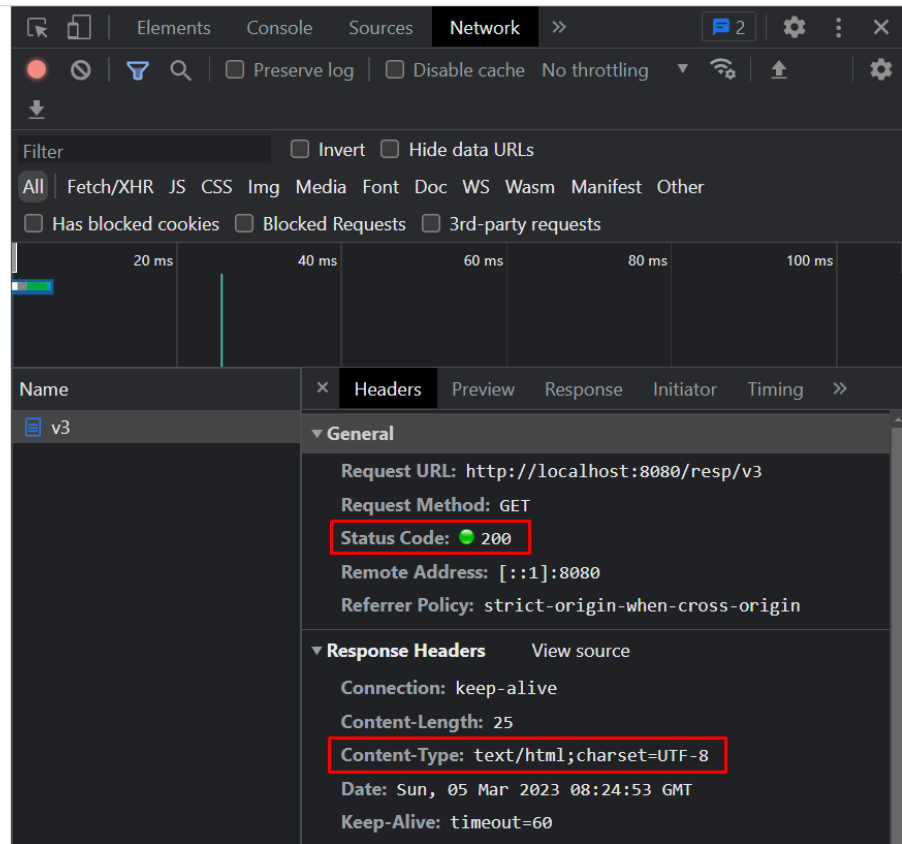
즉, @RestController는 @Controller와 @ResponseBody가 결합되어 만들어진 어노테이션이다.

```
@Controller
@ResponseBody
public @interface RestController {
```

controller/ResponseController.java 수정

```
@GetMapping("/resp/v3")
public @ResponseBody String respV3() {
    return "<h1>text/html 응답</h1>";
}
```

text/html 응답



5. Object 응답

스프링부트에서는 Object를 응답하면 `MessageConverter` 클래스가 일을 처리합니다. `MessageConverter`는 Object를 Json으로 변환하여 응답해줍니다.

이때 사용되는 라이브러리는 Jackson 이고, 아래 3개의 라이브러리가 필요합니다.



1. Jackson Databind

25,223 usages

com.fasterxml.jackson.core » [jackson-databind](https://com.fasterxml.jackson.core)

Apache

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

Last Release on Jan 29, 2023



2. Jackson Core

12,521 usages

com.fasterxml.jackson.core » [jackson-core](https://com.fasterxml.jackson.core)

Apache

Core Jackson processing abstractions (aka Streaming API), implementation for JSON

Last Release on Jan 29, 2023



3. Jackson Annotations

11,464 usages

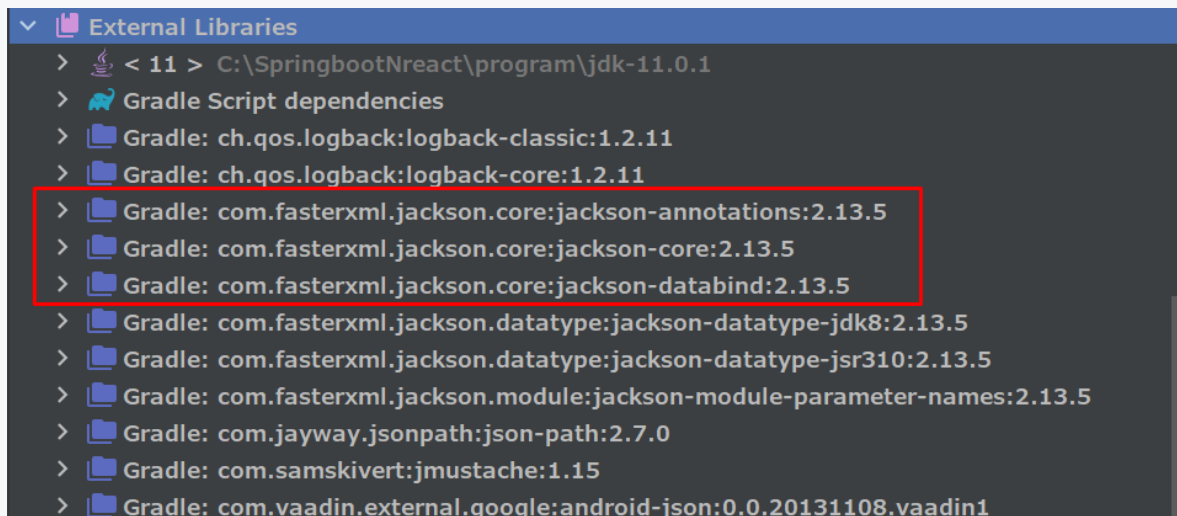
com.fasterxml.jackson.core » [jackson-annotations](https://com.fasterxml.jackson.core)

Apache

Core annotations used for value types, used by Jackson data binding package.

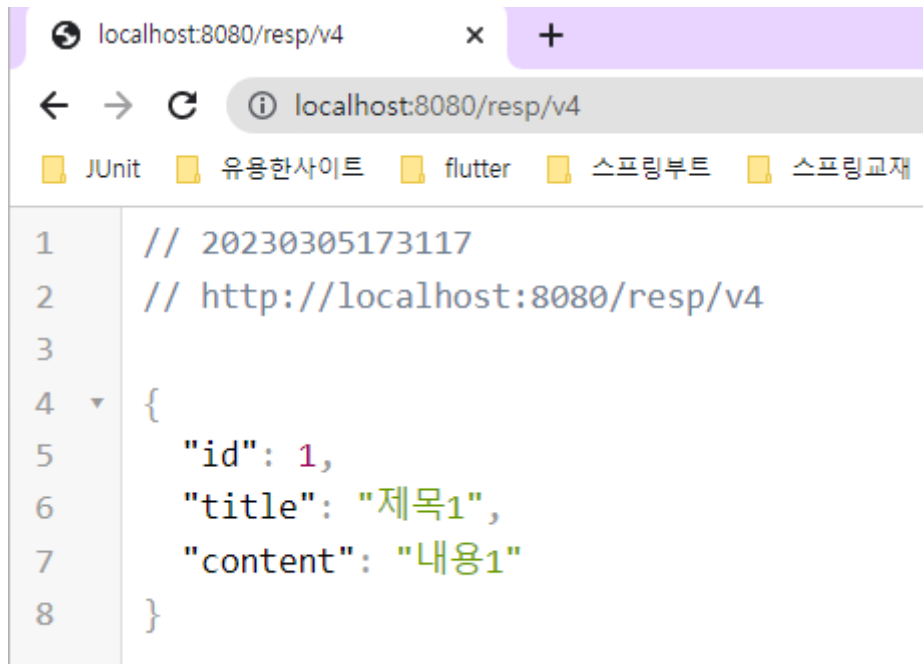
Last Release on Jan 29, 2023

인텔리J에서 External Libraries를 살펴보면 확인 가능합니다.



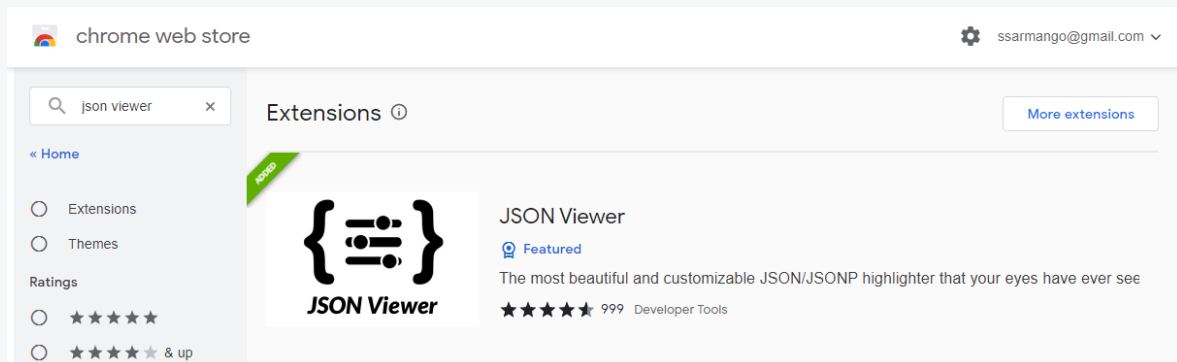
controller/ResponseController.java 수정

```
@GetMapping("/resp/v4")
public @ResponseBody BoardRespDto respV4() {
    return new BoardRespDto(1, "제목1", "내용1");
}
```

```
1 // 20230305173117
2 // http://localhost:8080/resp/v4
3
4 {
5   "id": 1,
6   "title": "제목1",
7   "content": "내용1"
8 }
```

브라우저에서 위와 같이 JSON으로 변환된 데이터가 예쁘게 출력되지 않는다면, 크롬 확장프로그램에서 Json Viewer를 설치하면 된다.



5. ResponseEntity 응답

스프링부트의 기본 응답 코드는 200 이다. 내부적으로 예외가 발생하지 않는 이상 200을 응답한다. ResponseEntity를 이용하면 상태코드를 자유롭게 응답할 수 있다.

물론 HttpServletResponse에 직접 header값을 추가해줄 수 도 있다.

```
@GetMapping("/example")
public void example(HttpServletResponse response) throws IOException {
    response.setStatus(HttpStatus.OK.value());
}
```

하지만 ResponseEntity를 활용하는 것이 좋다.

`ResponseEntity` 클래스를 사용하여 HTTP 응답을 반환할 수도 있습니다. `ResponseEntity` 는 HTTP 응답 본문뿐만 아니라 상태 코드, 헤더 등의 정보를 모두 포함한 객체입니다.

```
@GetMapping("/example")
public ResponseEntity<Void> example() {
    return ResponseEntity.ok().build();
}
```

`@Controller` 에서 `ResponseEntity` 를 사용하면 `@ResponseBody` 어노테이션을 사용하지 않아도 됩니다. `ResponseEntity` 는 HTTP 응답 본문뿐만 아니라 HTTP 응답 상태 코드, 헤더 등의 정보를 모두 포함한 객체입니다. 따라서, `ResponseEntity` 를 반환하면 스프링은 HTTP 응답을 생성하여 클라이언트에게 반환하게 됩니다.

controller/ResponseController.java 수정

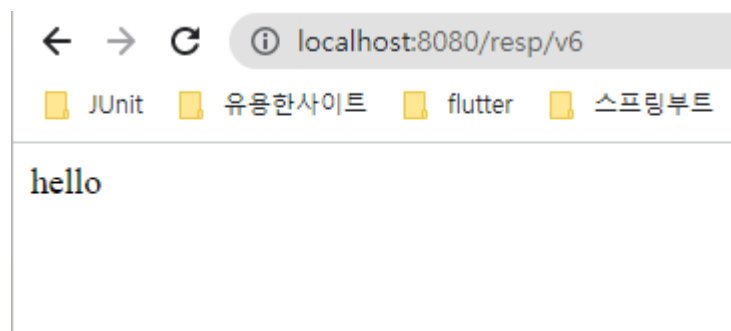
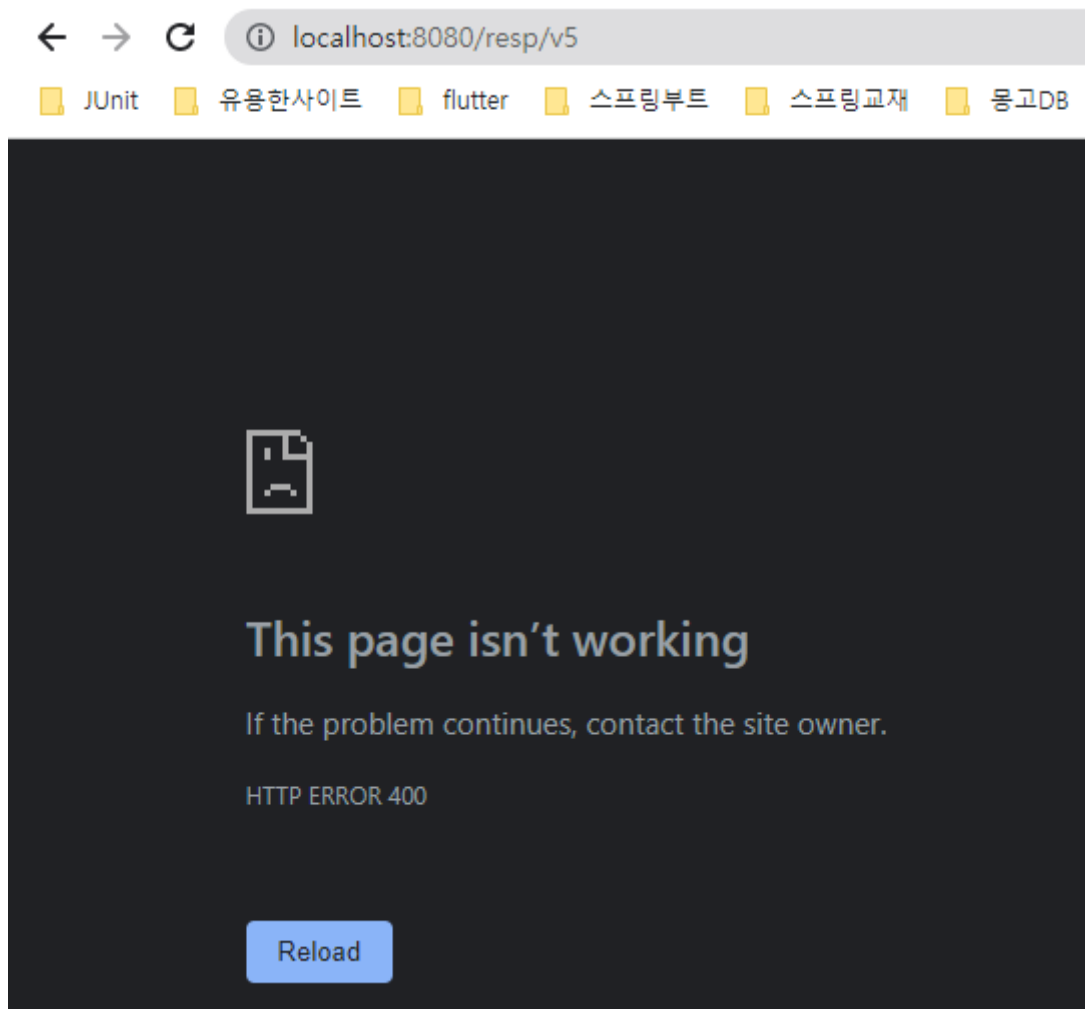
```
@GetMapping("/resp/v5")
public ResponseEntity<Void> respV5() {
    return ResponseEntity.badRequest().build();
}

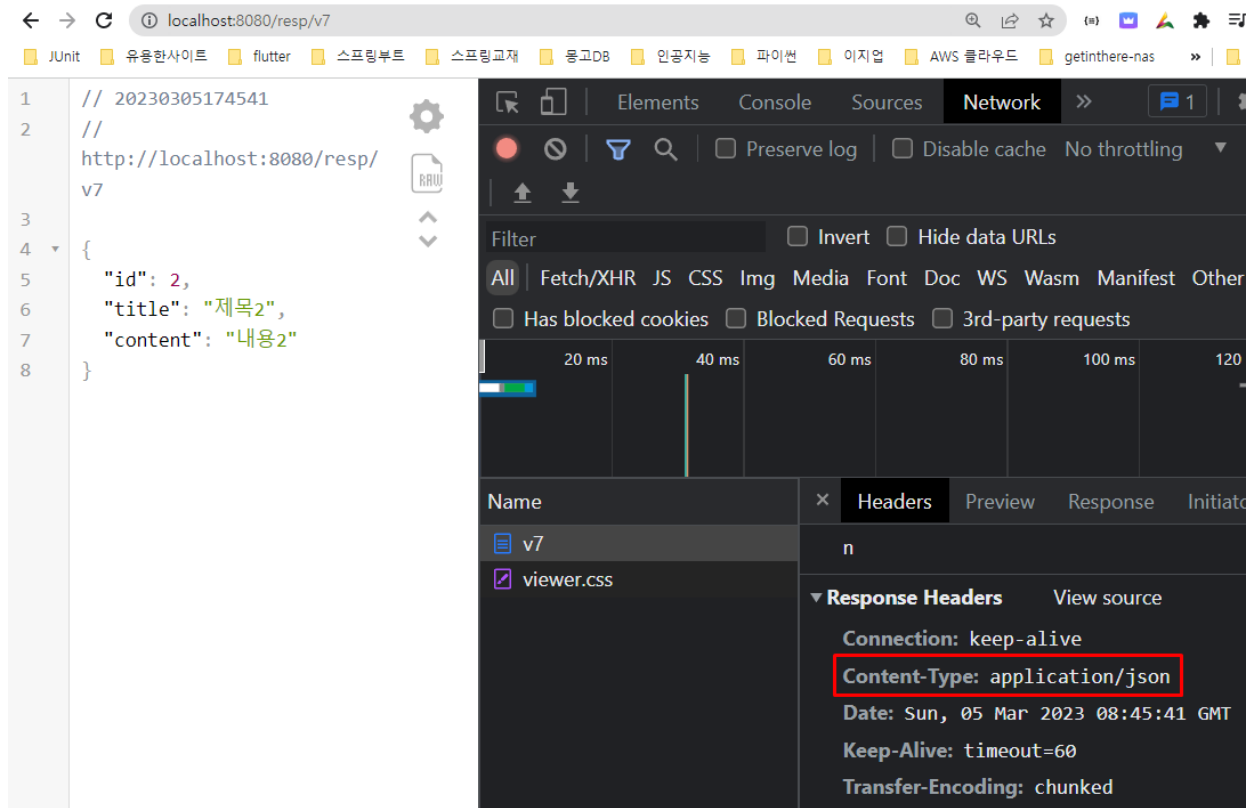
@GetMapping("/resp/v6")
public ResponseEntity<String> respV6() {
    return ResponseEntity.ok().body("hello");
}

@GetMapping("/resp/v7")
public ResponseEntity<?> respV7() {
    BoardRespDto boardRespDto = new BoardRespDto(2, "제목2", "내용2");
    return ResponseEntity.ok().body(boardRespDto);
}
```

`ResponseEntity`는 응답의 body값의 타입을 제네릭으로 지정해야 합니다. 하지만 그 타입이 동적이면 미리 알 수 없는 경우들이 있습니다.

예를 들어, A라는 요청이 들어오면 `String`을 반환하고, B라는 요청이 들어오면 `Integer`를 반환해야 한다면 반환 타입을 미리 지정해둘 수 없습니다. 이럴 때는 제네릭의 와일드카드를 사용합니다. 와일드카드를 사용하면 메서드가 return될 때 반환타입이 동적으로 결정됩니다.





6. Redirection

Spring Boot에서 `redirect:` 는 HTTP 요청을 다른 URI로 리디렉션하는 데 사용됩니다. 즉, `redirect:` 를 사용하면 클라이언트의 요청을 다른 URI로 전달하게 됩니다.

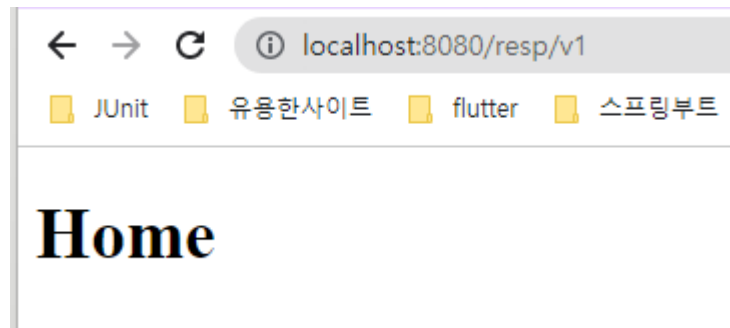
`redirect:` 는 `String` 형태의 URI를 반환하는 컨트롤러 메서드에서 사용됩니다.

```
@GetMapping("/example")
public String example() {
    return "redirect:/new-page";
}
```

참고로 `redirect`는 `@Controller`에서만 사용 가능합니다.

controller/ResponseController.java

```
@GetMapping("/resp/v8")
public String respV8() {
    return "redirect:/resp/v1";
}
```



7. 단위 테스트

Controller의 책임은 아래와 같다.

- 값 전달 받기 (QueryString, PathVariable, Body)
- 유효성 검사하기 (Post, Put 요청일 경우에만, 즉 Body에 데이터가 있을 경우)
- 인증 체크하기 (아직 배우지 않음)
- 비즈니스(mvc에서 m)에게 일 위임하기 (아직 배우지 않음)
- 값 응답하기 (View, ResponseEntity)

7.1 본코드 작성하기

dto/UserRespDto.java

```
package shop.mtcoding.conbasic.dto;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class UserRespDto {
    private int id;
    private String username;
    // password는 응답하지 않음.
    private String tel;

    public UserRespDto(int id, String username, String tel) {
        this.id = id;
        this.username = username;
        this.tel = tel;
    }

    @Override
    public String toString() {
```

```

        return "UserRespDto{" +
            "id=" + id +
            ", username='" + username + '\'' +
            ", tel='" + tel + '\'' +
            '}';
    }
}

```

resources/templates/detail.mustache

```

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Detail</h1>
    <hr/>
    <h3>번호 : {{user.id}}</h3>
    <h3>아이디 : {{user.username}}</h3>
    <h3>전화번호 : {{user.tel}}</h3>
</body>
</html>

```

controller/UserController.java

주소에 api가 붙은 메서드는 Text 혹은 Json을 응답하는 메서드이고, api가 없는 메서드는 View를 응답하는 메서드이다.

```

package shop.mtcoding.conbasic.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

```

```

import org.springframework.web.bind.annotation.ResponseBody;
import shop.mtcoding.conbasic.dto.UserRespDto;

import java.net.URI;

@Controller
public class UserController {

    @PostMapping("/api/users")
    public ResponseEntity<?> addUser(String username, String password, String tel){
        // 1. 유효성 검사
        if(username == null || username.isEmpty()){
            return ResponseEntity.badRequest().body("<h1>username 입력하세요</h1>");
        }
        if(password == null || password.isEmpty()){
            return ResponseEntity.badRequest().body("<h1>password 입력하세요</h1>");
        }
        if(tel == null || tel.isEmpty()){
            return ResponseEntity.badRequest().body("<h1>tel 입력하세요</h1>");
        }

        // 2. 비즈니스 로직 처리 (아직 배우지 않았음)
        int id = 1; // DB에 저장된 PK값을 받아옴

        // 3. 저장된 User 정보의 URI 응답 (201 - 새로운 리소스가 서버에 추가되었을 때)
        URI location = URI.create("/api/users/"+id);
        return ResponseEntity.created(location).build();
    }

    @GetMapping("/api/users/{id}")
    public ResponseEntity<?> getUser(@PathVariable int id){
        // 1. 인증 검사 (아직 배우지 않았음)

        // 2. 비즈니스 로직 처리 (아직 배우지 않았음)
        UserRespDto userRespDto = new UserRespDto(1, "ssar", "0102222"); // DB에 저장된
        User 정보를 받아옴

        // 3. 조회된 데이터를 응답(200)
        return ResponseEntity.ok().body(userRespDto);
    }

    @GetMapping("/users/{id}")
    public String detail(@PathVariable int id, Model model){
        // 1. 인증 검사 (아직 배우지 않았음)

        // 2. 비즈니스 로직 처리 (아직 배우지 않았음)
    }
}

```

```

        UserRespDto userRespDto = new UserRespDto(1, "ssar", "0102222"); // DB에 저장된
        User 정보를 받아옴

        // 3. 받아온 Data를 Model에 담기 (request Scope에 담는 것과 동일)
        model.addAttribute("user", userRespDto);

        // 4. View 응답
        return "detail";
    }

}

```

POST http://localhost:8080/api/users

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data **x-www-form-urlencoded** raw binary GraphQL

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> username	ssar	
<input checked="" type="checkbox"/> password	1234	
<input checked="" type="checkbox"/> tel	0102222	

Key Value Description

Body Cookies **Headers (5)** Test Results Status: 201 Created

KEY	VALUE
Location	/api/users/1
Content-Length	0
Date	Sun, 05 Mar 2023 09:06:34 GMT
Keep-Alive	timeout=60
Connection	keep-alive

POST http://localhost:8080/api/users

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE
<input checked="" type="checkbox"/>	username	ssar
<input type="checkbox"/>	password	1234
<input checked="" type="checkbox"/>	tel	0102222
	Key	Value

Body Cookies Headers (4) Test Results

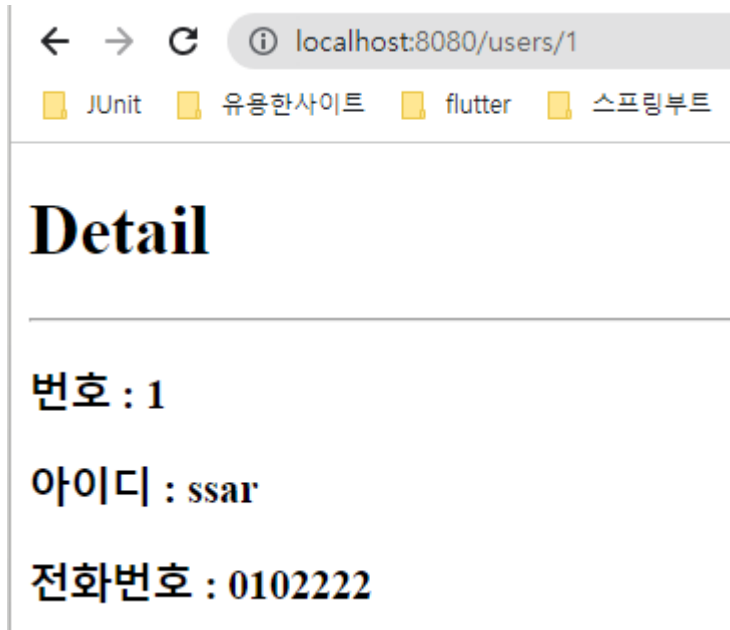
Pretty Raw Preview Visualize Text

1 `<h1>password 입력하세요</h1>`

← → ↻ ⓘ localhost:8080/api/users/1

JUnit 유용한사이트 flutter 스프링부트 스프링교재

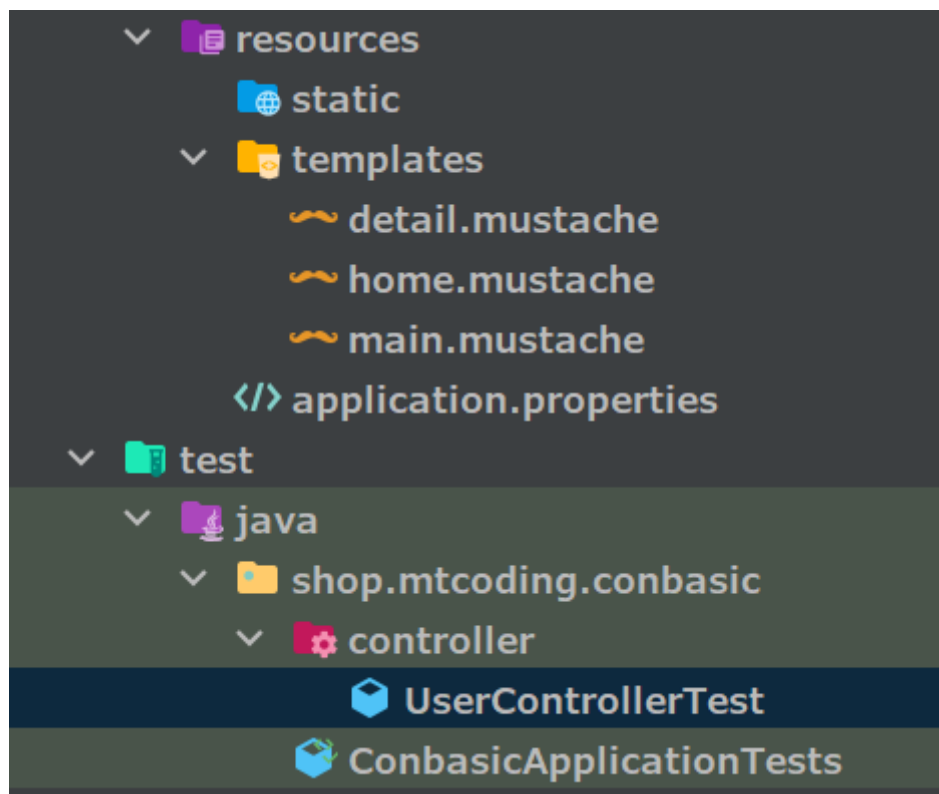
```
1 // 20230305181158
2 // http://localhost:8080/api/users/1
3
4 {
5   "id": 1,
6   "username": "ssar",
7   "tel": "0102222"
8 }
```



7.2 테스트코드 작성하기

컨트롤러가 잘 만들어졌는지를 확인하려면, 직접 브라우저 혹은 포스트맨을 열어서 테스트를 해봐야 한다. 이는 매우 번거로운 일이다. 테스트 코드를 작성해보자.

해당 테스트의 목적은 값을 잘 전달 받는지, 유효성 검사는 잘하는지, 응답은 잘되는지를 체크하는 것이다.



7.2.1 addUser 테스트

test/java/shop.mtcoding.conbasic/controller/UserControllerTest.java

테스트 코드에서 `given-when-then` 패턴은 테스트 메서드를 구성하는 세 가지 섹션입니다.

- `given` : 테스트 실행 전에 필요한 데이터를 설정합니다.
- `when` : 실제로 테스트할 동작을 실행합니다.
- `then` : 동작 실행 후 결과를 검증합니다.

이러한 패턴은 테스트 메서드가 코드 블록의 섹션으로 나뉘어 가독성이 높아지고, 어떤 테스트를 수행하는지 쉽게 이해할 수 있도록 도와줍니다.

아래의 예제 코드를 현재 모두 이해하려고 하지 마세요. 단 한가지만 기억하세요!!

MockMvc를 통해서 Controller에 Request 요청을 해볼 수 있다. 그리고 그 결과를 받아서 값을 검증해볼 수 있다.

```
package shop.mtcoding.conbasic.controller;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import org.springframework.test.web.servlet.ResultMatcher;
import org.springframework.test.web.servlet.request.MockMvcHttpRequestBuilder;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;

@WebMvcTest // Filter, DispatcherServlet, Controller, MockMvc를 IoC 컨테이너에 로드한다.
(웹 계층 관련된 모든 객체)
public class UserControllerTest {

    @Autowired // DI
    private MockMvc mockMvc;

    @Test // @Test를 붙이면 해당 메서드를 테스트할 수 있다.
    public void addUser_test() throws Exception{ // test 메서드는 파라미터를 전달받을 수 없다.

        // given (파라미터로 요청되는 데이터를 임의로 준비한다)
        String requestBody = "username=ssar&password=1234&tel=0102222"; // x-www-form-urlencoded
```

```

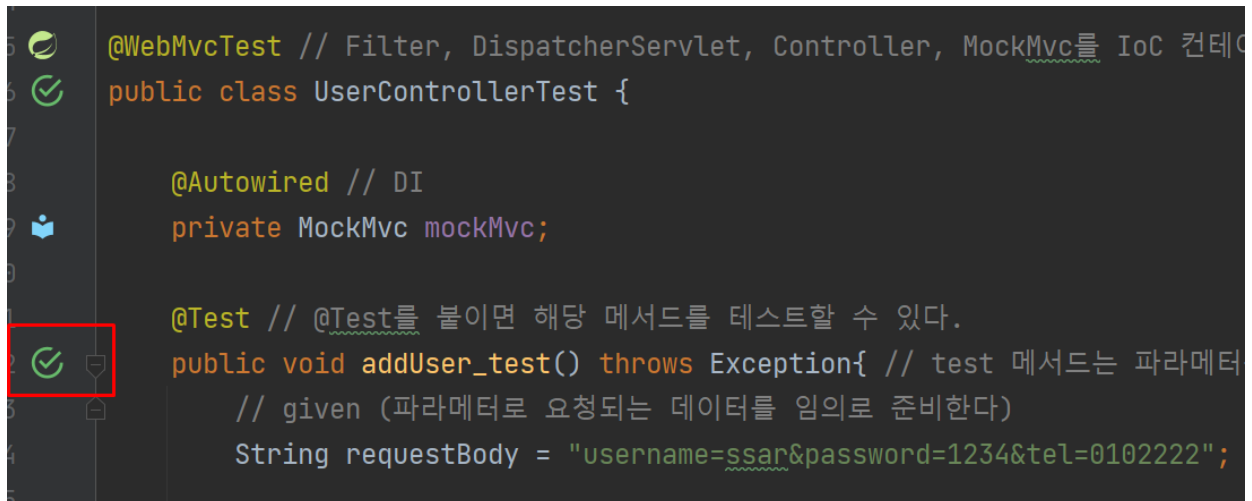
// when (본코드 테스트를 수행한다)
MockHttpServletRequestBuilder builders = MockMvcRequestBuilders
    .post("/api/users")
    .content(requestBody)
    .contentType(MediaType.APPLICATION_FORM_URLENCODED);
ResultActions actions = mockMvc.perform(builders);

// then (본코드 테스트 결과를 검증한다)
ResultMatcher isCreated = MockMvcResultMatchers.status().isCreated();
actions.andExpect(isCreated);

ResultMatcher location = MockMvcResultMatchers.header().string("Location",
"/api/users/1");
actions.andExpect(location);
}
}

```

테스트를 수행해봅니다.



```

@WebMvcTest // Filter, DispatcherServlet, Controller, MockMvc를 IoC 컨테이너에 등록
public class UserControllerTest {

    @Autowired // DI
    private MockMvc mockMvc;

    @Test // @Test를 붙이면 해당 메서드를 테스트할 수 있다.
    public void addUser_test() throws Exception{ // test 메서드는 파라미터
        // given (파라미터로 요청되는 데이터를 임의로 준비한다)
        String requestBody = "username=ssar&password=1234&tel=0102222";
    }
}

```

username의 값을 지우고 실패를 유도해봅니다.

```
String requestBody = "username=&password=1234&tel=0102222"; // x-www-form-urlencoded
```

MockHttpServletResponse:

Status = 400

Error message = null

Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"33"]

Content type = text/plain;charset=UTF-8

Body =

username 입력하세요

Forwarded URL = null Redirected URL = null Cookies = []

Status expected:<201> but was:<400>

Expected :201

Actual :400

테스트를 한 뒤 코드를 정상적으로 돌려두겠습니다.

```
String requestBody = "username=ssar&password=1234&tel=0102222"; // x-www-form-urlencoded
```

7.2.2 getUser 테스트

test/java/shop.mtcoding.conbasic/controller/UserControllerTest.java 추가

```
@Test
public void getUser_test() throws Exception{
    // given
    int id = 1;

    // when
    MockHttpServletRequestBuilder builders = MockMvcRequestBuilders
        .get("/api/users/"+id);
    ResultActions actions = mockMvc.perform(builders);

    // then
    ResultMatcher isOk = MockMvcResultMatchers.status().isOk();
    actions.andExpect(isOk);

    ResultMatcher isSameId = MockMvcResultMatchers.jsonPath("id").value("1");
    ResultMatcher isSameUsername =
    MockMvcResultMatchers.jsonPath("username").value("ssar");
    ResultMatcher isSameTel = MockMvcResultMatchers.jsonPath("tel").value("0102222");
    actions.andExpect(isSameId);
    actions.andExpect(isSameUsername);
    actions.andExpect(isSameTel);
}
```

전화번호를 틀리게 해서 테스트 수행

```
MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Content-Type:"application/json"]
    Content type = application/json
    Body = {"id":1,"username":"ssar","tel":"0102222"}
    Forwarded URL = null
    Redirected URL = null
    Cookies = []

JSON path "tel" expected:<010222> but was:<0102222>
Expected :010222
Actual   :0102222
```

정상적으로 테스트 수행

```
Tests passed: 1 of 1 test - 374 ms
:: Spring Boot :: (v2.7.9)

2023-03-05 18:56:02.904 INFO 9076 --- [ Test worker] s.m.c.controller.UserControllerTest : Starting UserControll
2023-03-05 18:56:02.905 INFO 9076 --- [ Test worker] s.m.c.controller.UserControllerTest : No active profile set
2023-03-05 18:56:03.669 INFO 9076 --- [ Test worker] o.s.b.t.m.w.SpringBootMockServletContext : Initializing Spring T
2023-03-05 18:56:03.670 INFO 9076 --- [ Test worker] o.s.t.web.servlet.TestDispatcherServlet : Initializing Servlet
2023-03-05 18:56:03.670 INFO 9076 --- [ Test worker] o.s.t.web.servlet.TestDispatcherServlet : Completed initializat
2023-03-05 18:56:03.689 INFO 9076 --- [ Test worker] s.m.c.controller.UserControllerTest : Started UserControll
BUILD SUCCESSFUL in 3s
4 actionable tasks: 2 executed, 2 up-to-date
오후 6:56:04: Execution finished 'test --tests "shop.mtcoding.conbasic.controller.UserControllerTest.getUser_test"'.

```

7.2.3 detail 테스트 (model 값 검증)

객체 안에 값을 비교할 때에는 toString을 재 정의하여 비교하는 것이 좋다.
객체를 비교하게 되면, 같은 값을 가지고 있더라도, 주소를 비교하기 때문이다.

actions.andExpect()를 사용하고 싶지만 이 친구를 사용하면 값을 하나씩 꺼내서 비교해야 하기 때문에 객체의 값을 비교할 때에는 org.assertj.core.api.Assertions 라이브러리를 사용하여 검증하는 것이 좋다.

test/java/shop.mtcoding.conbasic/controller/UserControllerTest.java 추가

```
@Test
public void detail_test() throws Exception {
    // given
    int id = 1;

    // when
    MockHttpServletRequestBuilder builders = MockMvcRequestBuilders
        .get("/users/" + id);
}
```

```

ResultActions actions = mockMvc.perform(builders);

// then
ResultMatcher isOk = MockMvcResultMatchers.status().isOk();
actions.andExpect(isOk);

UserRespDto userRespDto = (UserRespDto)
actions.andReturn().getModelAndView().getModel().get("user");
Assertions.assertThat(userRespDto.toString()).isEqualTo(new UserRespDto(1, "ssar",
"0102222").toString());
}

```

toString()을 붙이지 않고 테스트 수행

```

expected: "UserRespDto{id=1, username='ssar', tel='0102222'} (UserRespDto@3962ec84)"
but was: "UserRespDto{id=1, username='ssar', tel='0102222'} (UserRespDto@60b616c8)"

```

주소 값으로 비교하기 때문에 값 비교를 할 수 없다. 그렇다면 값을 하나씩 getter로 꺼내서 비교해봐도 된다. 하지만 값을 하나씩 꺼내서 비교하는 것은 toString()을 재정의하는 것보다 상대적으로 번거로운 일이다.

toString()을 붙이고 정상적으로 테스트 수행

 **Test Results**
376 ms

8. 정리

@Controller는 View를 응답하거나, Redirect할 수 있다.

@RestController는 Object를 응답하면 Json이 응답되고, String을 응답하면 text/html이 응답된다.

ResponseEntity를 사용하면 @Controller에서도 Json 혹은 String을 응답할 수 있다. 그리고 상태코드도 응답할 수 있다.

테스트를 사용하면 컨트롤러의 책임만 Check 해볼 수 있다. 추후에 Service, Repository 레이어가 생기더라도 Controller에서 문제가 난다면 UserControllerTest에서 문제가 발견될 것이다.