Day 15 (2019-08-14)
API Documentation

Spring Rest Docs

목차

REST Docs 소개

Spring REST Docs는 테스트 코드 기반으로 RESTful 문서생성을 돕는 도구로 기본적으로 Asciidoctor를 사용하여 HTML를 생성합니다.
Spring MVC 테스트 프레임워크로 생성된 snippet을 사용해서 snippet이 올바르지 않으면 생성된 테스트가 실패하여 정확성을 보장해줍니다.

Rest Docs vs Swagger

우선 Rest Docs과 Swagger는 성격이 많이 다릅니다. Swagger는 RESTful 문서에 대한 명세 보다는 Postman Tool 처럼 특정 API를 쉽게 호출해볼수 있는 것에 초점이 맞춰져있다고 있지않습니다. 다시 말해 Swagger는 API 명세에 대한 기능은 어느정도 제공해주지만 그것은 효율적이지 못하다고 생각합니다.

```
@ApiOperation(value = "View a list of available products", response = Iterable.class)
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "Successfully retrieved list"),
        @ApiResponse(code = 401, message = "You are not authorized to view the resource"),
        @ApiResponse(code = 403, message = "Accessing the resource you were trying to reach is
forbidden"),
        @ApiResponse(code = 404, message = "The resource you were trying to reach is not
found")
}
)
@RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")
public Iterable list(Model model){
    Iterable productList = productService.listAllProducts();
    return productList;
}
```

- 코드 출처

Swagger는 RESTful에대한 문서명세를 위와같은 어노테이션 방법으로 제공해주고 이런 방법은 다음과 같은 단점이 있습니다.

- 실제 코드에 영향을 미치지 않지만 지속해서 추가됨으로써 실제 코드보다 문서 명세에 대한 코드가 더 길어져 전체적인 가독성이 떨어집니다.
- **해당 코드는 주석일뿐 로직에 영향을 미치지 않기 때문에 비즈니스 로직이 변경되더라도 문서를 갱신하지 않아 결국 문서와 코드가 일치하지 않게 됩니다.**
- **정상적인 Reponse에 대한 명세 일뿐 status 2xx 이외의 값에 대한 정의가 힘듭니다. 이것을 정의하더라도 주석 형태로 정의하게 되어 결국 시간이 지나면 문서와 코드가 일치하지 않게 됩니다.**

제 개인적인 결론은 RESTful API에 대한 명세관점에서는 둘은 비교 대상이 되기 힘들다고 생각합니다. **Rest Docs는 테스트 코드 기반으로 문서가 작성되기 때문에 문서와 실제 코드의 일치성이 높고 테스트코드로 문서가 표현되기 때문에 실제 코드에 어떠한 코드 추가도 필요가 없다는 장점이 있습니다.** Swaager이외에도 실제 자바 주석문으로 RESTful API를 명세하는 서비스도 있지만 결국 문서와 코드의 일치성 등 다양항문제로 현재로써는 Rest Docs가 가장 효율적인 RESTful API 명세 도구라고 생각합니다.

pom.xml 설정

```
<dependencies>
    ...
    <!-- (1) -->
    <dependency>
        <groupId>org.springframework.restdocs</groupId>
        <artifactId>spring-restdocs-mockmvc</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

```
<plugins>
    <plugin>
    <!-- (2) -->
        <groupId>org.asciidoctor</groupId>
        <artifactId>asciidoctor-maven-plugin</artifactId>
        <version>1.5.3</version>
        <executions>
            <execution>
                <id>generate-docs</id>
                <phase>prepare-package</phase>
                <goals>
                    <goal>process-asciidoc</goal>
                </goals>
                <configuration>
                    <backend>html</backend>
                    <doctype>book</doctype>
                </configuration>
            </execution>
        </executions>
        <dependencies>
        <!-- (3) -->
            <dependency>
                <groupId>org.springframework.restdocs</groupId>
                <artifactId>spring-restdocs-asciidoctor</artifactId>
                <version>2.0.2.RELEASE</version>
            </dependency>
        </dependencies>
    </plugin>

    <plugin>
    <!-- (4) -->
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.7</version>
        <executions>
            <execution>
                <id>copy-resources</id>
                <phase>prepare-package</phase>
                <goals>
                    <goal>copy-resources</goal>
                </goals>
                <configuration>
                    <outputDirectory>
```

```
                    ${project.build.outputDirectory}/static/docs
                </outputDirectory>
                <resources>
                    <resource>
                        <directory>
                            ${project.build.directory}/generated-docs
                        </directory>
                    </resource>
                </resources>
            </configuration>
        </execution>
    </executions>
</plugin>
</plugins>
```

- (1) `restdocs` 의존성을 추가합니다. `scope`는 test로 지정합니다.
- (2) `asciidoctor` 플러그인을 추가합니다.
- (3) `spring-restdocs-asciidoctor` 의존성을 추가합니다. 해당 의존성이 추가되면 `snippets` 이 자동으로 구성됩니다.
- (4) `Asciidoctor` 플러그인 설정 입니다.
    - `outputDirectory` 문서가 출력되는 디렉토리 경로 입니다. 실제 `/target/classes/static/docs/` 경로에 문서가 생성됩니다.

Test Code

Test Code 설정

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MemberControllerTest {

    @Rule public JUnitRestDocumentation restDocumentation = new JUnitRestDocumentation(); //
(1)
    @Autowired private WebApplicationContext context;
    private MockMvc mockMvc; // (2)
    private RestDocumentationResultHandler document;

    // (3)
    @Before
    public void setUp() {
        this.document = document(
                "{class-name}/{method-name}",
                preprocessResponse(prettyPrint())
        );

        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.context)
                .apply(documentationConfiguration(this.restDocumentation))
                .alwaysDo(document)
                .build();
    }
}
```

- (1) Spring Rest Docs는 `JUnit` `TestNG` 등 다양한 테스트 프레임워크를 지원합니다. 본 예제는 `JUnit` 기반으로 테스트를 진행 하기 때문에 `JUnitRestDocumentation` 객체를 생성했습니다.
- (2) `MockMvc`, `WebTestClient`, `Rest Assured` 등 다양한 방식으로 Controller에 대한 테스트를 진행 할 수 있습니다. 본 예제는 `MockMvc` 기반으로 Controller 테스트를 진행하겠습니다.
- (3) `RestDocumentationResultHandler` 객체와 `MockMvc` 객체를 생성합니다.
  - `{class-name}/{method-name}` 설정은 해당 테스트 클래스의 이름 메서드 이름 기반으로 디렉토리 경로를 설정해서 `snippets`을 생성합니다. `target/generated-snippets/member-controller-test/get_member`에 저장됩니다.
  - `preprocessResponse(prettyPrint())` 설정을 통해서 해당 문서가 이쁘게 출력됩니다. 예를들어 JSON Response 값이 JSON 포멧팅에 맞게 출력됩니다.
  - `alwaysDo()` 메서드로 위에서 생성된 `RestDocumentationResultHandler` 객체를 의존성 주입해줍니다. 모든 `mockMvc` 테스트에 대한 `snippets`이 생성됩니다.

Member 조회 Test Code

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MemberControllerTest {
    ...
    @Test
    public void get_member() throws Exception {
        mockMvc.perform(get("/members/{id}", 1L)
                .accept(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andDo(document.document(
                        // (1)
                        pathParameters(
                                parameterWithName("id").description("Member's id")
                        ),
                        // (2)
                        responseFields(
                                fieldWithPath("email.value").description("The Member's email
address"),
                                fieldWithPath("address.city").description("The Member's
address city"),
                                fieldWithPath("address.street").description("The Member's
address street"),
                                fieldWithPath("address.zipCode").description("The Member's
address zipCode")
                        )
                ))
                .andExpect(jsonPath("$.email.value", is(notNullValue())))
                .andExpect(jsonPath("$.address.city", is(notNullValue())))
                .andExpect(jsonPath("$.address.street", is(notNullValue())))
                .andExpect(jsonPath("$.address.zipCode", is(notNullValue())))
        ;

    }
}
```

- (1) Path Parameter에 대한 정의 입니다. `parameterWithName("id").description("Member's id")` 설정을 통해서 간단하게 문서화가 가능합니다.
- (2) `responseFields()` 메서드로 reponse 값들에 대한 문서 정의가 가능합니다.

실제 Document 모습입니다.

Member 조회

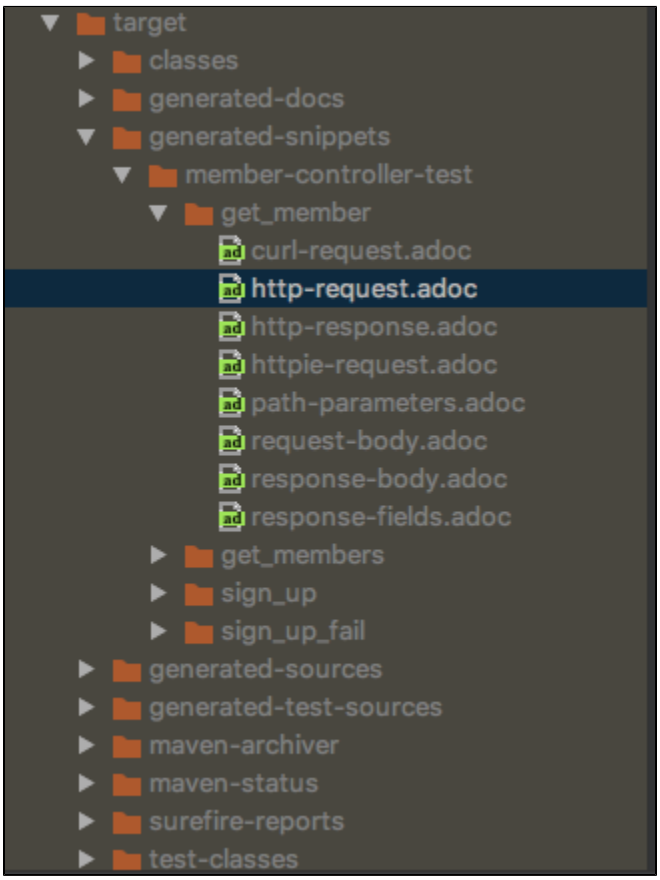
Table 1. /members/{id}

Parameter	Description
id	Member's id

```
HTTP/1.1 200 OK
Content-Length: 148
Content-Type: application/json;charset=UTF-8

{
  "email" : {
    "value" : "test0@test.com"
  },
  "address" : {
    "city" : "city0",
    "street" : "street0",
    "zipCode" : "zipCode0"
  }
}
```

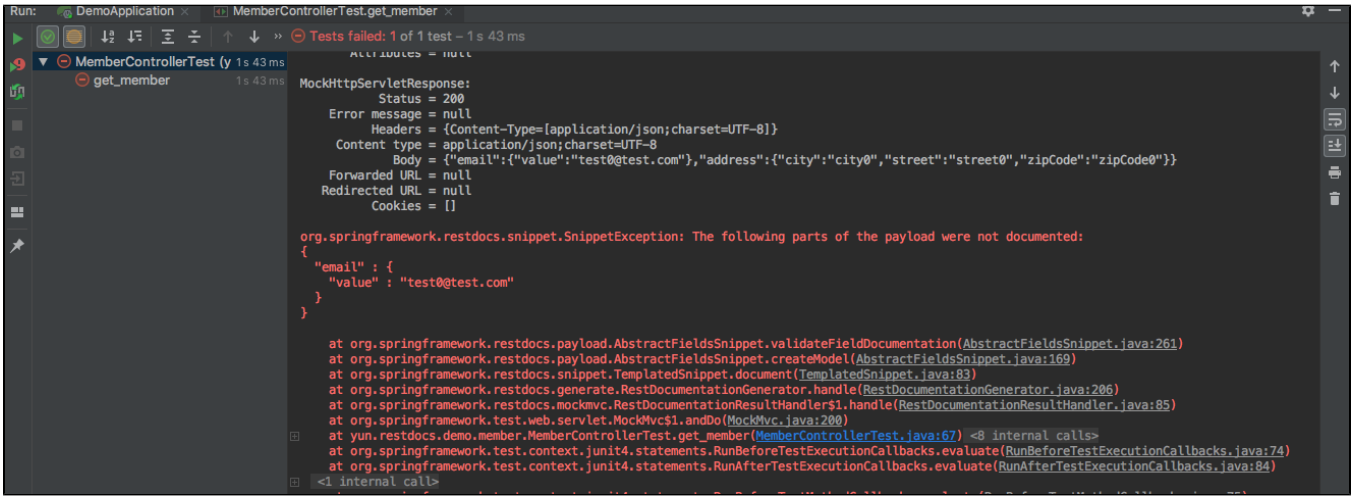
Path	Type	Description
email.value	String	The Member's email address
address.city	String	The Member's address city
address.street	String	The Member's address street
address.zipCode	String	The Member's address zipCode

기본적으로 snippets은 6개가 생성됩니다.



기본 `snippets` 6개 이외에도 `path-parameters.adoc`, `response-fields.adoc` 이 추가로 생성된 것을 확인 할 수 있습니다. 이것은 (1), (2) 에서 추가로 문서 작성 코드를 통해서 작성된 것입니다. 해당 `snippets`은 만들지 않더라도 기본적인 문서는 갖출 수 있습니다. **하지만 해당 코드를 작성하면 문서와 코드의 일치 성이 높아집니다.**

만약 `fieldWithPath("email.value").description("The Member's email address")` 코드를 주석하고 테스트 코드를 실행하게 되면



위의 그림과 같이 해당 페이로드가 문서화 되지 않았다는 메시지오 함께 테스트 코드가 실패하게 됩니다. 물론 `response-body.adoc`으로 매번 response 값이 갱신 되기 때문에 문서와의 일치 성은 보장 받을 수 있습니다. 해당 필드 값에 대한 타입과 정의까지 철저 하게 관리하고 싶으시다면 작성하는것도 좋다고 생각합니다.

Document 구조 adoc 파일 작성

테스트코드 기반으로 생성된 `snippets`을 실제 문서로 만들기 위한 `.adoc` 파일을 만들어야합니다. 문서의 뼈대 같은 구조를 잡아주는 작업 입니다.

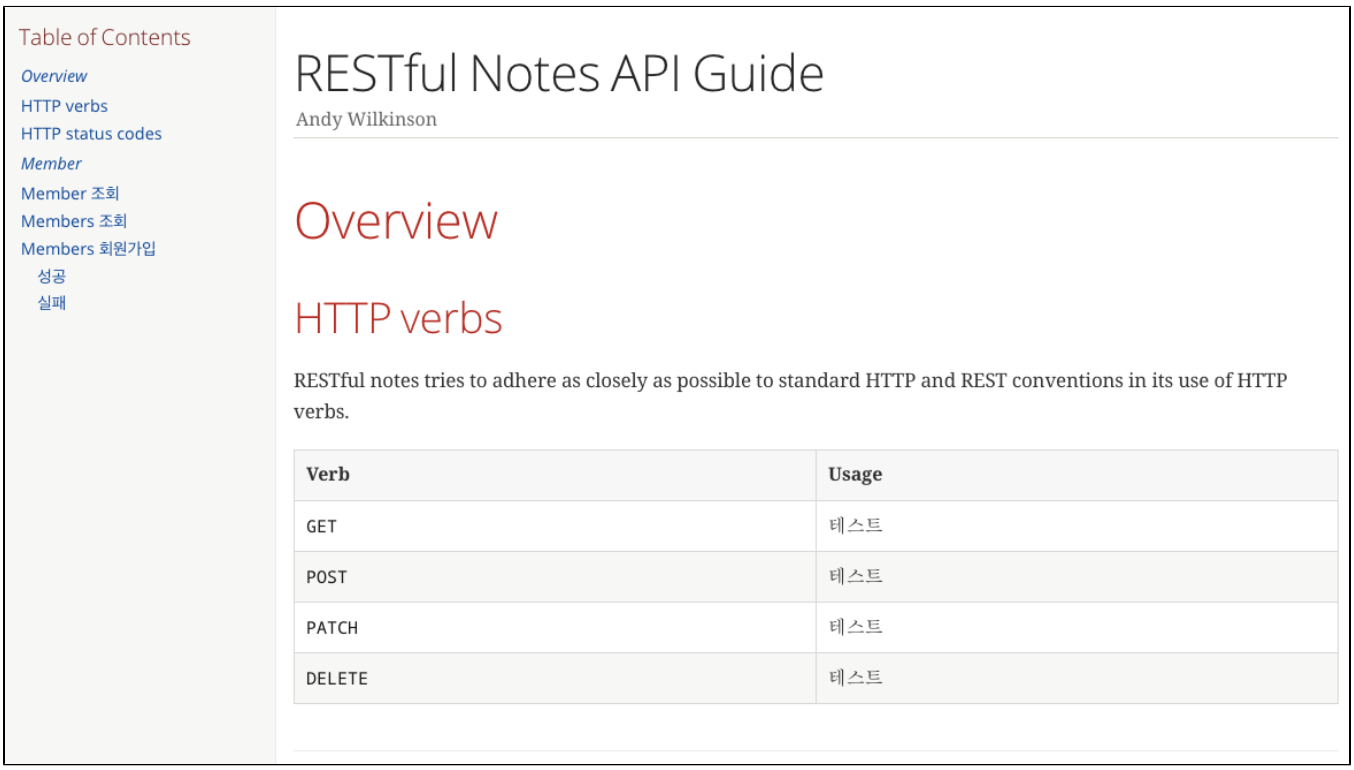
`src/main/asciidoc/` 경로에 `api-guide.adoc` 파일을 생성하고 테스트 코드를 통해서 생성된 `snippets` 파일들을 아래와 같이 추가합니다. **`api-guide.adoc` 파일의 일부분 이라서 전체 파일을 보시는 것이 좋습니다.**

```
= Member

== Member

include::{snippets}/member-controller-test/get_member/path-parameters.adoc[]
include::{snippets}/member-controller-test/get_member/http-response.adoc[]
include::{snippets}/member-controller-test/get_member/response-fields.adoc[]
include::{snippets}/member-controller-test/get_member/curl-request.adoc[]
include::{snippets}/member-controller-test/get_member/http-request.adoc[]
include::{snippets}/member-controller-test/get_member/httpie-request.adoc[]
include::{snippets}/member-controller-test/get_member/request-body.adoc[]
include::{snippets}/member-controller-test/get_member/response-body.adoc[]
```

최종 Document

```
mvn install
mvn spring-boot:run
```

테스트 코드를 실행하고 스프링 부트를 실행 이후 http://localhost:8080/docs/api-guide.html를 접속하면 Rest Docs 기반의 문서를 확인할 수 있습니다.



마무리

Member 조회 이외의 테스트 코드들도 있으니 프로젝트의 전체 코드를 보시는 것이 좋을 거 같습니다. 위에서 언급했듯이 문서와 코드의 지속적인 일치성, 비즈니스 로직을 가리는 과도한 문서화의 코드들이 침입하지 않는 점 등 Rest Docs는 정말 좋은 문서 도구라고 생각합니다. 반드시 테스트 코드를 작성해야 한다는 것도 테스트 코드에 익숙하지 않은 상태에서는 조금은 어렵겠지만 반강제적으로라도 테스트 코드를 작성하게 도와주는 점도 있어 이 또한 큰 장점입니다.

출처

https://github.com/dlxotn216/springboot-restdocs

Swagger

Java의 Annotation을 기반으로 문서화

```
@ApiOperation(value = "View a list of available products", response = Iterable.class)
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "Successfully retrieved list"),
        @ApiResponse(code = 401, message = "You are not authorized to view the resource"),
        @ApiResponse(code = 403, message = "Accessing the resource you were trying to reach is forbidden"),
        @ApiResponse(code = 404, message = "The resource you were trying to reach is not found")
}
)
@RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")
public Iterable list(Model model){
    Iterable productList = productService.listAllProducts();
    return productList;
}
```

apiDoc

Javadoc과 비슷한 스타일의 주석을 작성한 후 export를 통해 API 문서화 및 Test stub을 생성할 수 있습니다.

사용법

(1) npm을 이용

Install

```
npm install apidoc -g
```

Write apiDoc comment

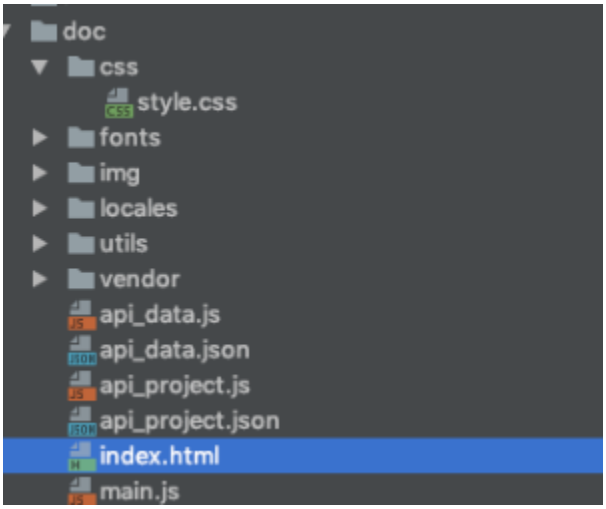
```
/**
 * @api {get} /user/:id Request User information
 * @apiName GetUser
 * @apiGroup User
 *
 * @apiParam {Number} id Users unique ID.
 *
 * @apiSuccess {String} firstname Firstname of the User.
 * @apiSuccess {String} lastname  Lastname of the User.
 */
...
 /**
     *
     * @api {post} /templates Template  API
     * @apiGroup Template
     * @apiName CreateTemplate
     *
     * @apiParam {String} templateName        (Unique)
     * @apiParam {Number} fileKey              (   File Key)
     * @apiParamExample {json} Request-Example:
     *      {
     *          "templateName": "MyTemplate 001",
     *          "fileKey": 1
     *      }
     *
     * @apiSuccess (201) {String} status                             'success'
     * @apiSuccess (201) {Object} result                              body
     * @apiSuccess (201) {Object} result.template                      Template
     * @apiSuccess (201) {Number} result.template.templateVersionKey
     * @apiSuccess (201) {String} result.template.templateName
     * @apiSuccess (201) {String} result.template.templateType
     * @apiSuccess (201) {String} result.template.fileName
     * @apiSuccess (201) {String} result.template.fileDownloadURI      URI
     * @apiSuccessExample {json} Success-Response:
     *     HTTP/1.1 201 OK
     *      {
     *          "result":{
     *               "template":{
     *                  "templateVersionKey":1,
     *                  "templateName":"My Template 01",
     *                  "templateType":"PDF",
     *                  "fileName":"test.html",
     *                  "fileDownloadURI":"/api/v2/files/2/downloads"
     *              }
     *          },
     *          "message":"Request was success",
```

```
*          "responseTimezone":"Asia/Seoul",
*          "responseLanguage":"en","status":"success"
*      }
*
* @apiSampleRequest  /templates
*
* @apiError MethodArgumentNotValidException
* @apiErrorExample Error-Response:
*     HTTP/1.1 400 Bad Request
*     {
*       "error": {
*          "errorCode":"REQUIRED_PARAMETER",
*          "errorDetails":"Required parameter [templateName] is not present"
*        }
*     }
*
* @apiError NotAcceptableTemplateFileException     template file
* @apiErrorExample Error-Response:
*     HTTP/1.1 400 Bad Request
*     {
*       "error": {
*          "errorCode":"NOT_ACCEPTABLE_TEMPLATE_FILE_EXTENSION",
*          "errorDetails":"[test.abc] is not acceptable template file extension. [html,xml] are available"
*       }
*     }
*
* @apiError DuplicatedDataException                template name
* @apiErrorExample Error-Response:
*     HTTP/1.1 409 Conflict
*     {
*       "error": {
*          "errorCode":"DUPLICATED_DATA",
*          "errorDetails":"field name: templateName [My Template 01] can not be duplicated"
*       }
*     }
*
*/
```

Run

apidoc -i src/ -o doc/

결과




apiDocl.html