

2025년 상반기 K-디지털 트레이닝

## Swagger-ui를 활용한 문서 자동화

Swagger란 개발한 Rest API를 편리하게 문서화 해주고, 이를 통해서 관리 및 제 3의 사용자가 편리하게 API를 호출해보고 테스트 할 수 있는 프로젝트 이다.

Annotation	
@Api	클래스를 스웨거의 리소스로 표시
@ApiOperation	특정 경로의 오퍼레이션 HTTP 메소드 설명
@ApiParam	오퍼레이션 파라미터에 메타 데이터 설명
@ApiResponse	오퍼레이션의 응답 지정
@ApiModelProperty	모델의 속성 데이터를 설명
@ApiImplicitParam	메소드 단위의 오퍼레이션 파라미터를 설명
@ApiImplicitParams	

- 

## 번외 javadoc만드는 방법은 학습 필요함



## Swagger-ui를 활용한 문서 자동화

### ✓ Swagger 라이브러리의 종류 2가지

- Spring-Fox\*,
- Spring-Doc,

### ✓ 의존성

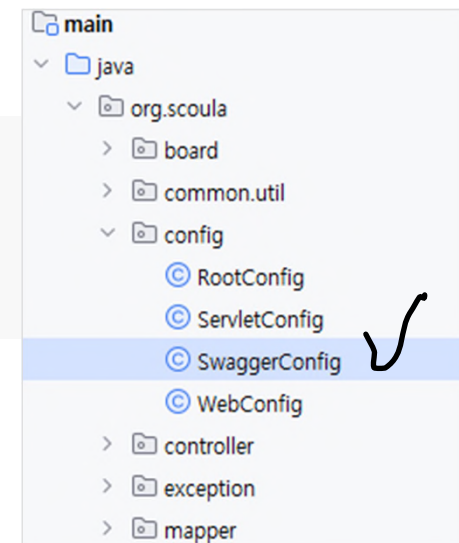
#### ○ build.gradle

implementation 'io.springfox:springfox-swagger2:2.9.2'  
implementation 'io.springfox:springfox-swagger-ui:2.9.2'

## config.SwaggerConfig.java

스웨거 전용 config

```
@Configuration  
@EnableSwagger2 ✓ 스웨거 툴을 활성화시키는 어노테이션  
public class SwaggerConfig {  
  
}
```



## config.SwaggerConfig.java

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    private final String API_NAME = "Board API";
    private final String API_VERSION = "1.0";
    private final String API_DESCRIPTION = "Board API 명세서";

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title(API_NAME)
            .description(API_DESCRIPTION)
            .version(API_VERSION)
            .build();
    }

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.withClassAnnotation(RestController.class))
            .paths(PathSelectors.any())
            .build()
            .apiInfo(apiInfo());
    }
}
```

내가 만든 설명할 api 정보

api라고하는 빈을 등록.

대상으로해서  
api를 만들겠다.

@RestController가 붙은 모든 컨트롤러를 대상으로 함

제한을 두지 않겠다.

api에 대한 기본 정보를 구성

## Swagger-ui를 활용한 문서 자동화

### config.WebConfig.java

swaggerconfig를 사용해라라고 등록

```
public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {
```

```
...
```

```
@Override
```

```
protected Class<?>[] getServletConfigClasses() {  
    return new Class[] { ServletConfig.class, SwaggerConfig.class };  
}
```

// 스프링의 FrontController인 DispatcherServlet이 담당할 url 매핑 패턴, /: 모든 요청에 대해 매핑

```
@Override
```

```
protected String[] getServletMappings() {
```

```
    return new String[]{
```

```
        "/",
```

```
        "/swagger-ui.html",
```

```
        "/swagger-resources/**",
```

```
        "/v2/api-docs",
```

```
        "/webjars/**"
```

```
    };
```

```
}
```

```
...
```

```
}
```

요청 경로 추가.  
문서를 보기 위한 인덱스 페이지  
스웨거의 정적파일

## config.ServletConfig.java

```
public class ServletConfig implements WebMvcConfigurer {
```

```
@Override
```

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {
```

```
    registry
```

```
        .addResourceHandler("/resources/**") // url이 /resources/로 시작하는 모든 경로
```

```
        .addResourceLocations("/resources/"); // webapp/resources/경로로 매핑
```

```
// Swagger UI 리소스를 위한 핸들러 설정
```

```
registry.addResourceHandler("/swagger-ui.html")
```

```
    .addResourceLocations("classpath:/META-INF/resources/");
```

```
// Swagger WebJar 리소스 설정
```

```
registry.addResourceHandler("/webjars/**")
```

```
    .addResourceLocations("classpath:/META-INF/resources/webjars/");
```

```
// Swagger 리소스 설정
```

```
registry.addResourceHandler("/swagger-resources/**")
```

```
    .addResourceLocations("classpath:/META-INF/resources/");
```

```
registry.addResourceHandler("/v2/api-docs")
```

```
    .addResourceLocations("classpath:/META-INF/resources/");
```

```
}
```

```
...
```

스웨거의 정적인 경로 추가해줘야함.

스웨거의 정적 리소스 4가지 추가.

개발과정에서는 meta-inf/resources  
폴더는 존재하지 않음.

런타임때 스웨거에의해서 자동으로  
만들어진 폴더임



## Swagger-ui를 활용한 문서 자동화

✓ <http://localhost:8080/swagger-ui.html>

swagger Select a spec default

**Board API** 1.0  
[ Base URL: localhost:8080/ ]  
<http://localhost:8080/v2/api-docs>  
Board API 명세서

**board-controller** Board Controller

- GET /api/board getList
- POST /api/board create
- GET /api/board/{no} get
- PUT /api/board/{no} update
- DELETE /api/board/{no} delete

아까 apiInfo()로 설정한 내용들

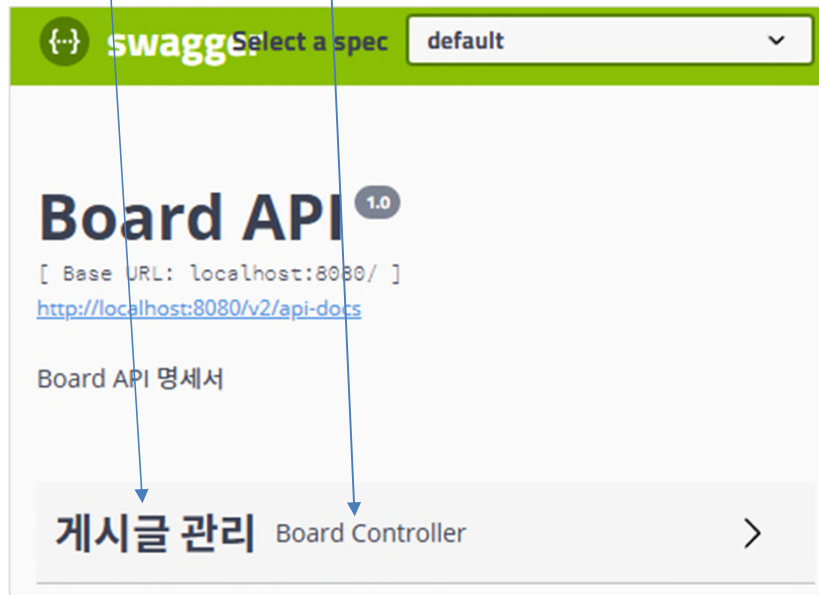
RestController 어노테이션이 붙은 클래스를 발견하고  
표현. 이름은 케밥케이스로 표현.  
펼치면 어떤 메소드가 어떤 방식으로 url과 매핑되어있는지.

### ✓ RestController에 정보 설정하기

- @Api(tags="API 타이틀")

@Api(tags = "게시글 관리")

public class BoardController { ... }



디폴트로는 레스트컨트롤러명의 케밥케이스버전

### controller.BoardController.java

```
@RestController
@RequestMapping("/api/board")
@RequiredArgsConstructor
@Log4j2
@Api(tags = "게시글 관리")
public class BoardController {
    private final BoardService service;
    ...
}
```

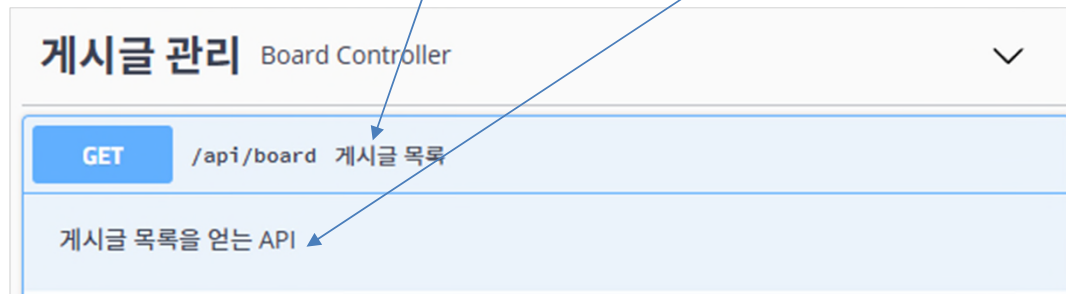
### ✓ API 엔드포인트 설명

- `@ApiOperation(value = "api 명", notes = "설명")` ✓

`@ApiOperation(value = "게시글 목록", notes = "게시글 목록을 얻는 API")`

`@GetMapping("")`

`public ResponseEntity<List<BoardDTO>> getList() { ... }`



### controller.BoardController.java

```
...
@Api(tags = "게시글 관리") ✓
public class BoardController {
    private final BoardService service;

    @ApiOperation(value = "게시글 목록", notes = "게시글 목록을 얻는 API") ✓
    @GetMapping("") — 이게 엔드포인트가 되겠다
    public ResponseEntity<List<BoardDTO>> getList() {
        return ResponseEntity.ok(service.getList());
    }
    ...
}
```

### ✓ API 엔드포인트 상세 설명

#### ○ @ApiParam

- 엔드포인트 파라미터 설명
- @PathVariable, @RequestBody 앞에 설정
- 주요 속성
  - value 속성: 엔드포인트의 간략한 설명,
  - required 속성: 필수 여부
  - example 속성: 파라미터의 예시 값 제공

너무 좋은데

## Swagger-ui를 활용한 문서 자동화

### controller.BoardController.java

```
@ApiOperation(value = "상세정보 얻기", notes = "게시글 상세 정보를 얻는 API")
@GetMapping("/{no}")
public ResponseEntity<BoardDTO> get(
    @ApiParam(value = "게시글 ID", required = true, example = "1")
    @PathVariable Long no) {
    return ResponseEntity.ok(service.get(no));
}
```

숫자 타입인 경우 example을 숫자값으로 지정해줘야 함  
지정하지 않은 경우 NumberFormatException 예외발생



GET /api/board/{no} 상세정보 얻기

게시글 상세 정보를 얻는 API

Parameters

Name Description

no \* required ✓

integer (\$int64) ✓

(path)

게시글 ID

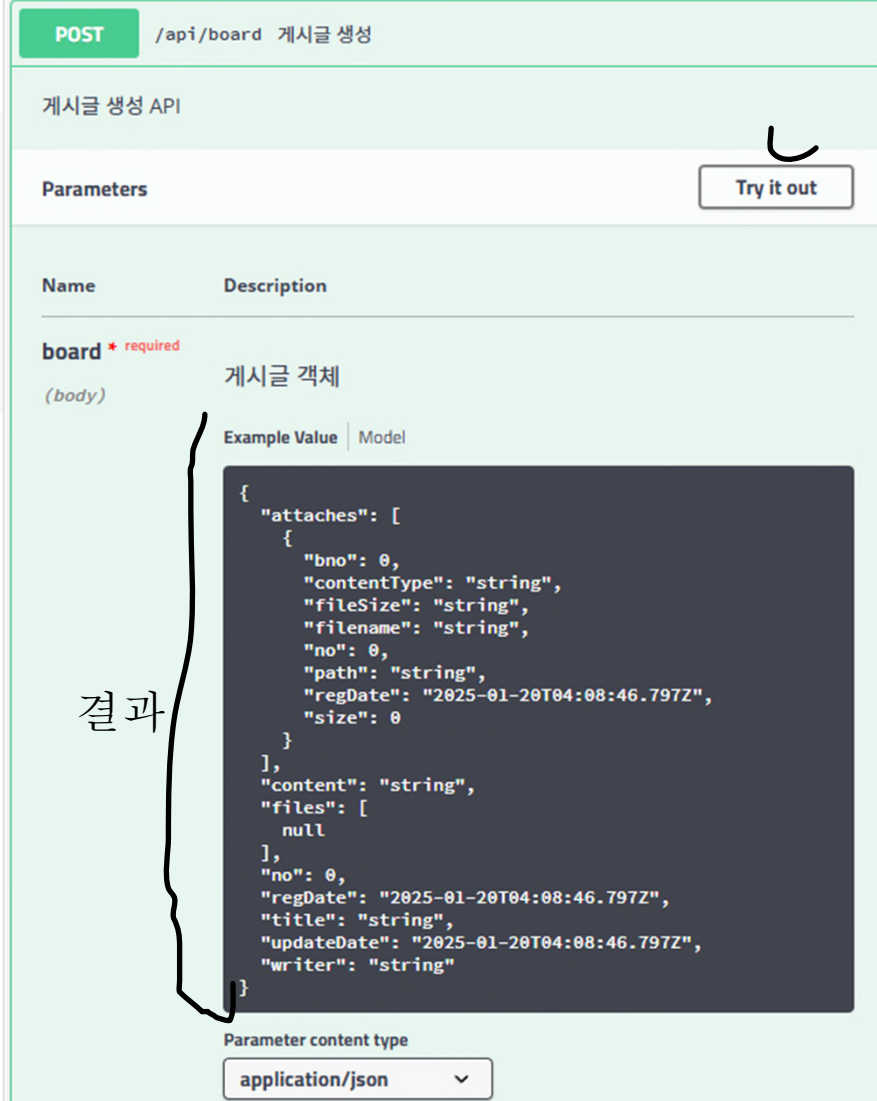
Try it out ✓

api사용자 입자엿서는  
백엔드가 어느 언어로 쓰여있는지 몰라.  
언어의 독립형태를 위해서  
저런식으로 표현.

# Swagger-ui를 활용한 문서 자동화

## controller.BoardController.java

```
@ApiOperation(value = "게시글 생성", notes = "게시글 생성 API")
@PostMapping("")
public ResponseEntity<BoardDTO> create(
    @ApiParam(value = "게시글 객체", required = true)
    @RequestBody BoardDTO board) {
    return ResponseEntity.ok(service.create(board));
}
```



POST /api/board 게시글 생성

게시글 생성 API

Try it out

Parameters

Name	Description
board * required (body)	게시글 객체

Example Value | Model

```
{
  "attaches": [
    {
      "bno": 0,
      "contentType": "string",
      "fileSize": "string",
      "filename": "string",
      "no": 0,
      "path": "string",
      "regDate": "2025-01-20T04:08:46.797Z",
      "size": 0
    }
  ],
  "content": "string",
  "files": [
    null
  ],
  "no": 0,
  "regDate": "2025-01-20T04:08:46.797Z",
  "title": "string",
  "updateDate": "2025-01-20T04:08:46.797Z",
  "writer": "string"
}
```

Parameter content type  
application/json

결과



## Swagger-ui를 활용한 문서 자동화

배열로 지정. api 응답

### ✓ API 엔드포인트 상세 설명

#### ○ @ApiResponses(value = {}) :

- @ApiResponse의 배열
  - code 속성: 응답 코드 ✓
  - message 속성: 값의 의미설명 ✓
  - response 속성: 응답 객체 class 정보 ✓ ——— 실제로 받는 객체

```
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
```

### controller.BoardController.java

```
@ApiOperation(value = "게시글 목록", notes = "게시글 목록을 얻는 API")
```

```
@ApiResponses(value = {
```

```
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
```

```
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
```

```
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
```

```
})
```

```
@GetMapping("")
```

```
public ResponseEntity<List<BoardDTO>> getList() {
```

```
    return ResponseEntity.ok(service.getList());
```

```
}
```

리스트 or 배열 형태여도  
단일 타입으로 지정

Responses Response content type \*/\*

Code	Description
200	성공적으로 요청이 처리되었습니다.

Example Value | Model

```
{
  "attaches": [
    {
      "bno": 0,
      "contentType": "string",
      "fileSize": "string",
      "filename": "string",
      "no": 0,
      "path": "string",
      "regDate": "2025-01-20T03:55:06.154Z",
      "size": 0
    }
  ],
  "content": "string",
  "files": [
    null
  ],
  "no": 0,
  "regDate": "2025-01-20T03:55:06.154Z",
  "title": "string",
  "updateDate": "2025-01-20T03:55:06.154Z",
  "writer": "string"
}
```

해당 객체가 응답으로 왔음

400	잘못된 요청입니다.
401	Unauthorized
403	Forbidden
404	Not Found
500	서버에서 오류가 발생했습니다.

### controller.BoardController.java

```
@ApiOperation(value = "상세정보 얻기", notes = "게시글 상세 정보를 얻는 API")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
@GetMapping("/{no}")
public ResponseEntity<BoardDTO> get(
    @ApiParam(value = "게시글 ID", required = true, example = "1")
    @PathVariable Long no) {
    return ResponseEntity.ok(service.get(no));
}
```

Responses Response content type \*/\*

Code	Description
200	OK

Example Value | Model

```

{
  "attaches": [
    {
      "bno": 0,
      "contentType": "string",
      "fileSize": "string",
      "filename": "string",
      "no": 0,
      "path": "string",
      "regDate": "2025-01-20T04:08:56.492Z",
      "size": 0
    }
  ],
  "content": "string",
  "files": [
    null
  ],
  "no": 0,
  "regDate": "2025-01-20T04:08:56.492Z",
  "title": "string",
  "updateDate": "2025-01-20T04:08:56.492Z",
  "writer": "string"
}
    
```

401	Unauthorized
403	Forbidden
404	Not Found

### controller.BoardController.java(전체 코드)

```
@RestController
@RequestMapping("/api/board")
@RequiredArgsConstructor
@Log4j2
@Api(tags = "게시글 관리") ✓
public class BoardController {
    private final BoardService service;

    @ApiOperation(value = "게시글 목록", notes = "게시글 목록을 얻는 API")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
        @ApiResponse(code = 400, message = "잘못된 요청입니다."),
        @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
    })
    @GetMapping("")
    public ResponseEntity<List<BoardDTO>> getList() {
        return ResponseEntity.ok(service.getList());
    }
}
```

### controller.BoardController.java(전체 코드)

```
@ApiOperation(value = "상세정보 얻기", notes = "게시글 상세 정보를 얻는 API")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
@GetMapping("/{no}")
public ResponseEntity<BoardDTO> get(
    @ApiParam(value = "게시글 ID", required = true, example = "1")
    @PathVariable Long no) {
    return ResponseEntity.ok(service.get(no));
}
```

### controller.BoardController.java(전체 코드)

```
@ApiOperation(value = "게시글 생성", notes = "게시글 생성 API")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
@PostMapping("")
public ResponseEntity<BoardDTO> create(
    @ApiParam(value = "게시글 객체", required = true) ↴
    @RequestBody BoardDTO board) {
    return ResponseEntity.ok(service.create(board));
}
```



### controller.BoardController.java(전체 코드)

```
@ApiOperation(value = "게시글 수정", notes = "게시글 수정 API")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다.", response = BoardDTO.class),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
@PutMapping("/{no}")
public ResponseEntity<BoardDTO> update(
    @ApiParam(value = "게시글 ID", required = true, example = "1")
    @PathVariable Long no,
    @ApiParam(value = "게시글 객체", required = true)
    @RequestBody BoardDTO board) {
    return ResponseEntity.ok(service.update(board));
}
```

## Swagger-ui를 활용한 문서 자동화

### controller.BoardController.java(전체 코드)

```

@ApiOperation(value = "게시글 삭제", notes = "게시글 삭제 API")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "성공적으로 요청이 처리되었습니다."),
    @ApiResponse(code = 400, message = "잘못된 요청입니다."),
    @ApiResponse(code = 500, message = "서버에서 오류가 발생했습니다.")
})
@DeleteMapping("/{no}")
public ResponseEntity<BoardDTO> delete(
    @ApiParam(value = "게시글 ID", required = true, example = "1")
    @PathVariable
    Long no) {
    return ResponseEntity.ok(service.delete(no));
}
}

```

livetemplate이라고  
자동완성 기능을 커스텀할 수 있다.

반복되는 swAGGER 어노테이션  
내용을  
자동완성 되게끔 하자.

설정 > 에디터 > 라이브 템플릿 > swagger 그룹 추가  
> 거기에 livetemplate 추가 >  
약어 등록 'api-operation' 설명: 'swagger@apiOperation 추가'

템플릿 텍스트 > 등록할 코드 내용 넣기. 정의 > java 체크

>>> 약어를 치면 자동등록 내용이 뜬다.

api-param도 라이브 템플릿을 추가하여 편하게 작업해보자

## Swagger-ui를 활용한 문서 자동화

### ✓ DTO 모델에 대한 문서화

- `@ApiModelProperty(description = "게시글 DTO")` ✓
  - DTO 클래스에 지정
- `@ApiModelProperty(value = "게시글 ID", example = "1")` V
  - DTO 필드에 지정
  - 숫자 형인 경우 반드시 example에 숫자값 지정

이것도 라이브 템플릿으로

## Swagger-ui를 활용한 문서 자동화

### dto.BoardDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@ApiModel(description = "게시글 DTO")
public class BoardDTO {
    @ApiModelProperty(value = "업로드 파일 목록")
    List<MultipartFile> files = new ArrayList<>(); // 실제 업로드된 파일(Multipart) 목록
    @ApiModelProperty(value = "게시글 ID", example = "1")
    private Long no;
    @ApiModelProperty(value = "제목")
    private String title;
    @ApiModelProperty(value = "글 본문")
    private String content;
    @ApiModelProperty(value = "작성자")
    private String writer;
    @ApiModelProperty(value = "등록일")
    private Date regDate;
    @ApiModelProperty(value = "수정일")
    private Date updateDate;
    // 첨부 파일
    @ApiModelProperty(value = "첨부파일 목록")
    private List<BoardAttachmentVO> attaches;
```

우리가 만든  
rest API  
에서 사용하는  
객체들에 대한  
설명  
들이 나옴

#### Models

##### BoardAttachmentVO >

##### BoardDTO ▾ {

description:	게시글 DTO
attaches	> [...]
content	string 글 본문
files	> [...]
no	integer(\$int64) example: 1 게시글 ID
regDate	string(\$date-time) 등록일
title	string 제목
updateDate	string(\$date-time) 수정일
writer	string 작성자