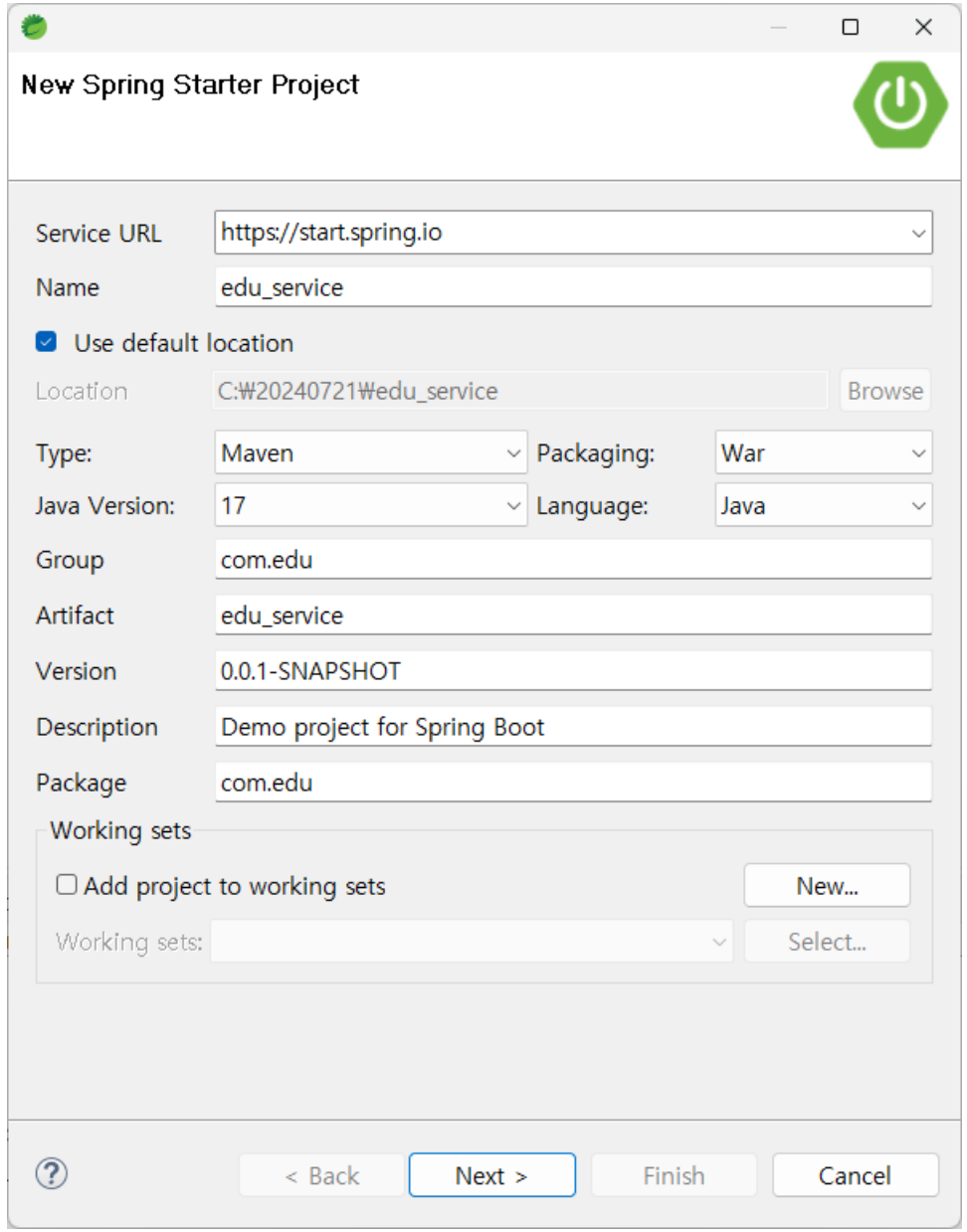
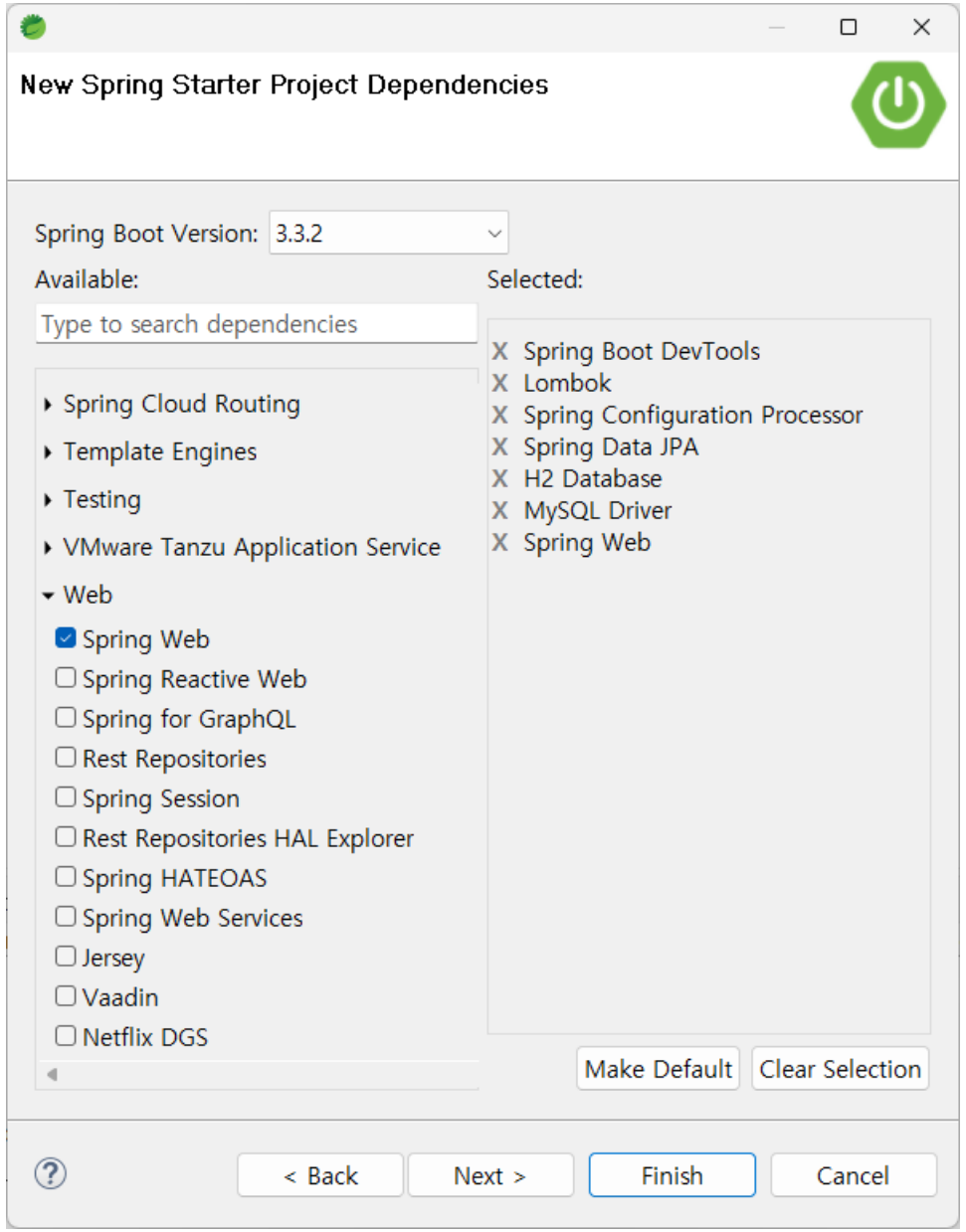
Service/Controller

Spring Boot

Service 구현

BoardServiceImpl


New Spring Starter Project
Service URL
https://start.spring.io
Name
edu_service
Use default location
Location
C:₩20240721₩edu_service
Browse
Type:
Maven
Packaging:
War
Java Version:
17
Language:
Java
Group
com.edu
Artifact
edu_service
Version
0.0.1-SNAPSHOT
Description
Demo project for Spring Boot
Package
com.edu
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel


New Spring Starter Project Dependencies
Spring Boot Version:
3.3.2
Available:
Type to search dependencies
Spring Cloud Routing
Template Engines
Testing
VMware Tanzu Application Service
Web
Spring Web
Spring Reactive Web
Spring for GraphQL
Rest Repositories
Spring Session
Rest Repositories HAL Explorer
Spring HATEOAS
Spring Web Services
Jersey
Vaadin
Netflix DGS
Selected:
X Spring Boot DevTools
X Lombok
X Spring Configuration Processor
X Spring Data JPA
X H2 Database
X MySQL Driver
X Spring Web
Make Default
Clear Selection
< Back
Next >
Finish
Cancel

```java
package com.edu.entity;

~ 생략 ~

@Setter
@Getter
@ToString
@Entity
@Table(name = "board3")
@EntityListeners(AuditingEntityListener.class)
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long seq;
    private String title;
    private String writer;
    private String content;

    @CreatedDate
    private LocalDateTime regDate;

    public void updateBoard(BoardDto dto) {
        this.title = dto.getTitle();
        this.content = dto.getContent();
    }


}
```

```java
package com.edu.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.edu.entity.Board;

public interface BoardRepository extends JpaRepository<Board, Long> {

}
```

Model Mapper

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.1.1</version>
</dependency>
```

```java
package com.edu.dto;

import org.modelmapper.ModelMapper;
~ 생략 ~

@Getter
@Setter
@ToString
public class BoardDto {
    private Long seq;
    private String title;
    private String writer;
    private String content;
    private LocalDateTime regDate;

    private static ModelMapper modelMapper = new ModelMapper();

    public Board createBoard() {
        return modelMapper.map(this, Board.class);
    }

    public static BoardDto of(Board board) {
        return modelMapper.map(board, BoardDto.class);
    }

}
```

```java
package com.edu.service;

import java.util.List;

import com.edu.dto.BoardDto;

public interface BoardService {

    Long insertBoard(BoardDto dto);

    Long updateBoard(BoardDto dto);

    void deleteBoard(Long seq);

    BoardDto getBoard(Long seq);

    List<BoardDto> getBoardList();
}
```

```java
package com.edu.service;

import java.util.ArrayList;
import java.util.List;
import jakarta.persistence.EntityNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.edu.dto.BoardDto;
import com.edu.entity.Board;
import com.edu.repository.BoardRepository;

import lombok.RequiredArgsConstructor;

@Service
@Transactional
@RequiredArgsConstructor
public class BoardServiceImpl implements BoardService {

    private final BoardRepository boardRepo;

    public Long insertBoard(BoardDto dto) {
        Board board = dto.createBoard();
        Board savedBoard = boardRepo.save(board);
        return savedBoard.getSeq();
    }
```

```java
    public Long updateBoard(BoardDto dto) {
        Board board = boardRepo.findById(dto.getSeq()).orElseThrow(EntityNotFoundException::new);
        board.updateBoard(dto);
        return board.getSeq();
    }

    public void deleteBoard(Long seq) {
        Board board = boardRepo.findById(seq).orElseThrow(EntityNotFoundException::new);
        boardRepo.delete(board);
    }

    @Transactional(readOnly = true)
    public BoardDto getBoard(Long seq) {
        Board board = boardRepo.findById(seq).orElseThrow(EntityNotFoundException::new);
        return BoardDto.of(board);
    }

    @Transactional(readOnly = true)
    public List<BoardDto> getBoardList() {
        List<Board> boardList = boardRepo.findAll();
        List<BoardDto> boardDtoList = new ArrayList<>();
        for (Board board : boardList) {
            BoardDto boardDto = BoardDto.of(board);
            boardDtoList.add(boardDto);
        }
        return boardDtoList;
    }
}
```

```java
package com.edu.service;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.List;
import java.util.Optional;

import jakarta.persistence.EntityNotFoundException;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.edu.dto.BoardDto;
import com.edu.entity.Board;
import com.edu.repository.BoardRepository;
import com.edu.service.BoardService;

@SpringBootTest
public class BoardServiceTest {

    @Autowired
    private BoardService boardService;

    @Autowired
    private BoardRepository boardRepo;
```

```java
public BoardDto saveBoardDto() {
    BoardDto BoardDto = new BoardDto();
    BoardDto.setTitle("MVC");
    BoardDto.setWriter("홍길동");
    BoardDto.setContent("Model, View, Controller 디자인 패턴이다");
    return BoardDto;
}

@Test
public void insertBoardTest() {
    BoardDto boardDto = this.saveBoardDto();
    long seq = boardService.insertBoard(boardDto);

    Board board = boardRepo.findById(seq).orElseThrow(EntityNotFoundException::new);

    assertThat(board.getWriter()).isEqualTo(boardDto.getWriter());
    assertThat(board.getTitle()).isEqualTo(boardDto.getTitle());
    assertThat(board.getContent()).isEqualTo(boardDto.getContent());
}
```

```java
@Test
public void updateBoardTest() {
    BoardDto boardDto = new BoardDto();
    boardDto.setSeq(1L);
    boardDto.setTitle("Spring");
    boardDto.setContent("Backend Java 프레임워크이다");

    long seq = boardService.updateBoard(boardDto);

    Board board = boardRepo.findById(seq).orElseThrow(EntityNotFoundException::new);

    assertThat(board.getTitle()).isEqualTo(boardDto.getTitle());
    assertThat(board.getContent()).isEqualTo(boardDto.getContent());
}


@Test
public void getBoardTest() {
    long seq = 1L;
    BoardDto BoardDto = boardService.getBoard(seq);
    System.out.println(BoardDto);
}
```

```
    @Test
    public void getBoardListTest() {
        BoardDto boardDto = this.saveBoardDto();
        boardService.insertBoard(boardDto);

        List<BoardDto> boardList = boardService.getBoardList();

        boardList.forEach(System.out::println);
    }

    @Test
    public void deleteBoardTest() {
        long seq = 1L;
        boardService.deleteBoard(seq);

        Optional<Board> board = boardRepo.findById(seq);

        assertThat(board.isPresent()).isFalse();
    }
}
```

Controller 구현

BoardController

REST API

- REST API (Representational State Transfer Application Programming Interface)
  웹 서비스를 구현하기 위한 아키텍처 스타일이다. 이 아키텍처 스타일은 클라이언트와 서버 간의 상호작용을 구조화하는 규칙과 제약을 정의한다.
- RESTful API
  "RESTful"이라는 용어는 해당 API가 REST 아키텍처 스타일을 따르고 있다는 것을 강조하는 것으로 RESTful API는 REST의 제약과 원칙을 충실히 따르는 API이다.

REST API 원칙

Stateless

- 각 요청은 독립적이며, 서버는 클라이언트의 이전 요청에 대한 정보를 저장하지 않아야 한다.

Client-Server Architecture

- 클라이언트와 서버는 서로 독립적으로 설계되어야 한다.

Uniform Interface

- 일관된 방식으로 리소스에 접근할 수 있어야 한다.

REST API 원칙

리소스(Resource)

- REST API에서 모든 데이터는 리소스로 취급되며 각 리소스는 고유한 URL로 식별된다. 예를 들어, 사용자는 /users 리소스에 의해 나타낼 수 있다.

HTTP 메서드

- REST API는 HTTP 프로토콜을 사용하며, 주요 HTTP 메서드에 따라 동작이 결정된다.

HTTP 메서드	동작
GET	리소스를 조회한다
POST	새로운 리소스를 생성한다
PUT	기존 리소스를 업데이트한다
DELETE	리소스를 삭제한다
PATCH	리소스의 일부를 업데이트한다

REST API의 장점

유연성	• 클라이언트와 서버의 독립적인 개발이 가능하다. 클라이언트는 서버의 구현 방식에 의존하지 않는다.
확장성	• REST API는 확장성이 뛰어나며, 다양한 클라이언트(웹, 모바일, IoT 등)에서 사용할 수 있다.
단순성	• HTTP 프로토콜을 기반으로 하여 간단하고 직관적인 설계를 제공한다.
캐시 처리	• HTTP 캐싱 메커니즘을 통해 성능을 최적화할 수 있다.

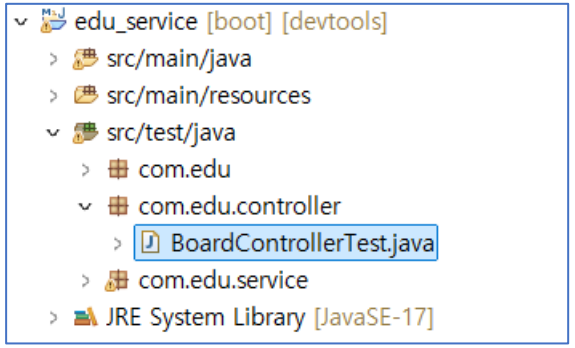
com.edu.controller.BoardController

```
package com.edu.controller;

import java.util.List;

import org.springframework.web.bind.annotation.DeleteMapping;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.edu.dto.BoardDto;
import com.edu.service.BoardService;
import lombok.RequiredArgsConstructor;
```

com.edu.controller.BoardController

```java
@RequestMapping("/board")
@RestController
@RequiredArgsConstructor
public class BoardController {

    private final BoardService boardService;

    @PostMapping("/insert")
        public void insertBoard(@RequestBody BoardDto dto) {
        boardService.insertBoard(dto);
    }
    @PutMapping("/update")
        public void updateBoard(@RequestBody BoardDto dto) {
        boardService.updateBoard(dto);
    }
    @DeleteMapping("/delete/{seq}")
        public void deleteBoard(@PathVariable Long seq) {
        boardService.deleteBoard(seq);
    }
    @GetMapping("/get/{seq}")
        public BoardDto getBoard(@PathVariable Long seq) {
        return boardService.getBoard(seq);
    }
    @GetMapping("/list")
        public List<BoardDto> getBoardList() {
        return boardService.getBoardList();
    }
}
```

```java
package com.edu.controller;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import com.edu.dto.BoardDto;
import com.fasterxml.jackson.databind.ObjectMapper;

@SpringBootTest
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;
```



src/test/java/com.edu.controller.BoardControllerTest

```
public BoardDto saveBoardDto() {
    BoardDto BoardDto = new BoardDto();
    BoardDto.setTitle("Spring Boot");
    BoardDto.setWriter("오정임");
    BoardDto.setContent("Spring을 더 쉽게 이용하기 위한 도구이다.");
    return BoardDto;
}

@Test
@DisplayName("게시글 등록 테스트")
public void insertBoardTest() throws Exception {
    BoardDto boardDto = this.saveBoardDto();
    String content = objectMapper.writeValueAsString(boardDto);
    mockMvc.perform(post("/board/insert")
                                        .contentType(MediaType.APPLICATION_JSON)
                                        .content(content))
                                        .andDo(print())
                                        .andExpect(status().isOk());
}
```

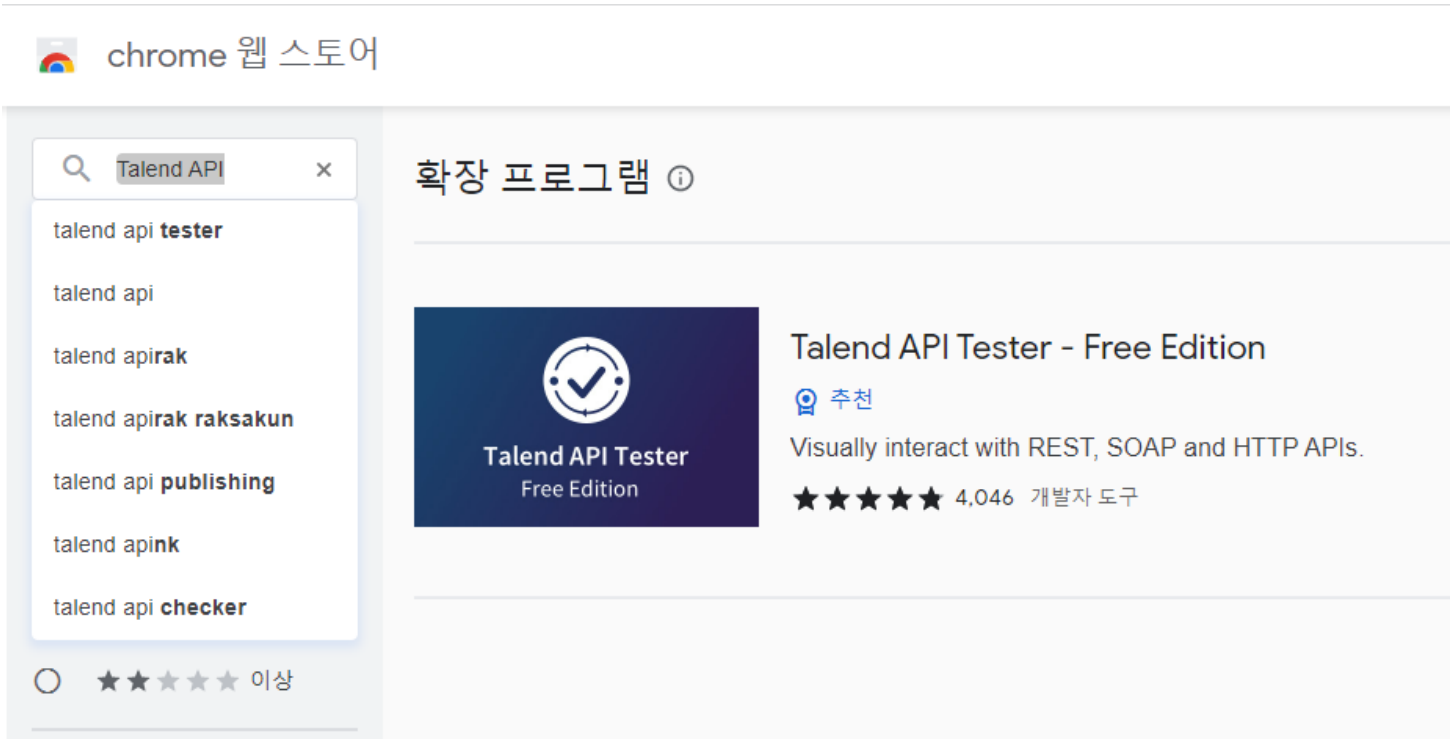
src/test/java/com.edu.controller.BoardControllerTest

```
@Test
@DisplayName("게시글 수정 테스트")
public void updateBoardTest() throws Exception{
    BoardDto boardDto = new BoardDto();
    boardDto.setSeq(1L);
    boardDto.setTitle("Spring");
    boardDto.setContent("MVC 기반의 웹 프레임워크이다");

    mockMvc.perform(put("/board/update")
                                    .contentType(MediaType.APPLICATION_JSON)
                                    .content(objectMapper.writeValueAsString(boardDto)))
                                    .andExpect(status().isOk())
                                    .andDo(print());
}

@Test
@DisplayName("게시글 삭제 테스트")
public void deleteBoardTest() throws Exception {
mockMvc.perform(delete("/board/delete/3")
                                      .andExpect(status().isOk())
                                      .andDo(print());
                            }
```

src/test/java/com.edu.controller.BoardControllerTest

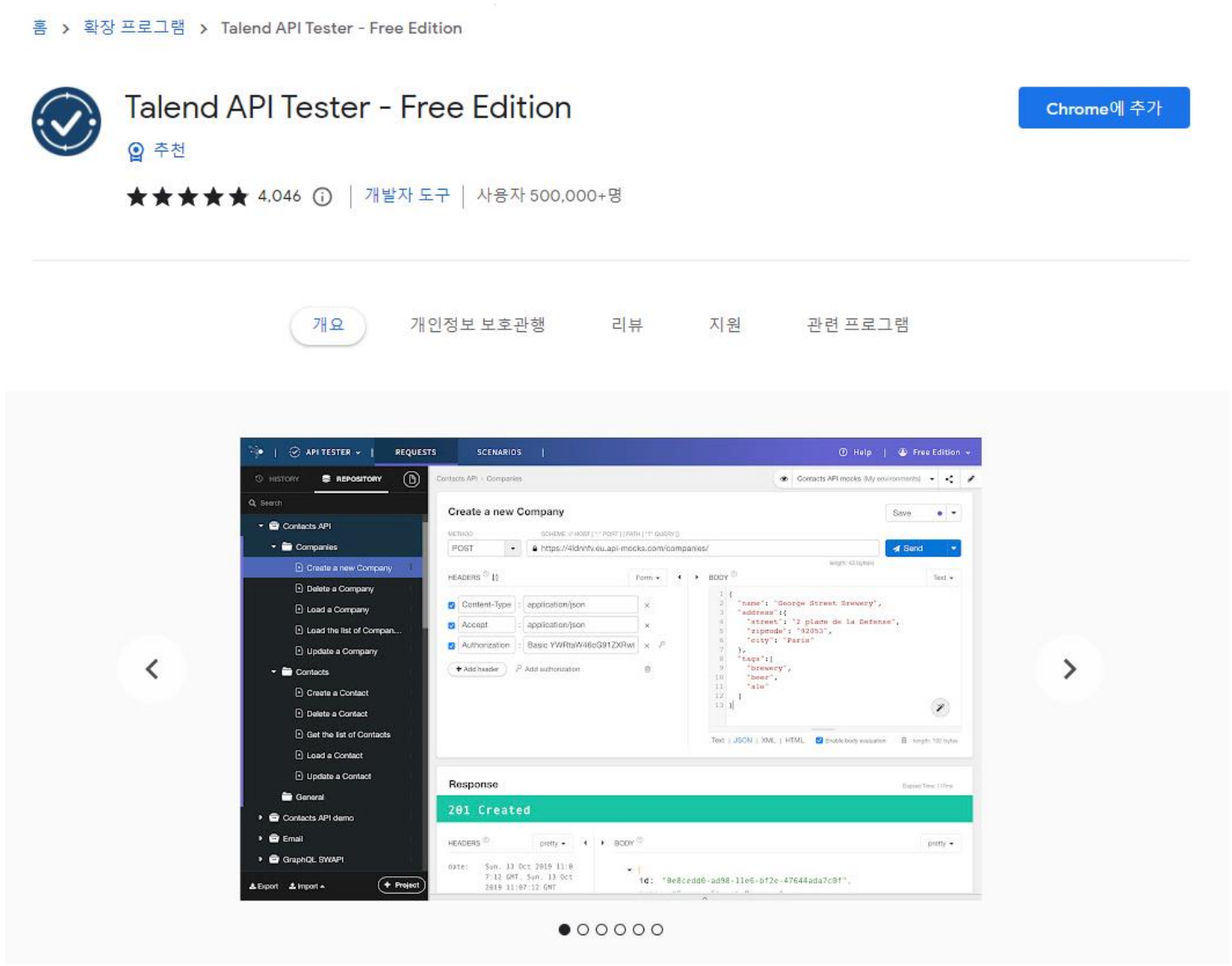
```
    @Test
    @DisplayName("특정 게시글 조회 테스트")
    public void getBoardTest() throws Exception{
    mockMvc.perform(get("/board/get/1"))
           .andExpect(status().isOk())
           .andDo(print());

    }

    @Test
    @DisplayName("모든 게시글 조회 테스트")
    public void getBoardListTest() throws Exception{
    mockMvc.perform(get("/board/list"))
           .andExpect(status().isOk())
           .andDo(print());

    }
}
```

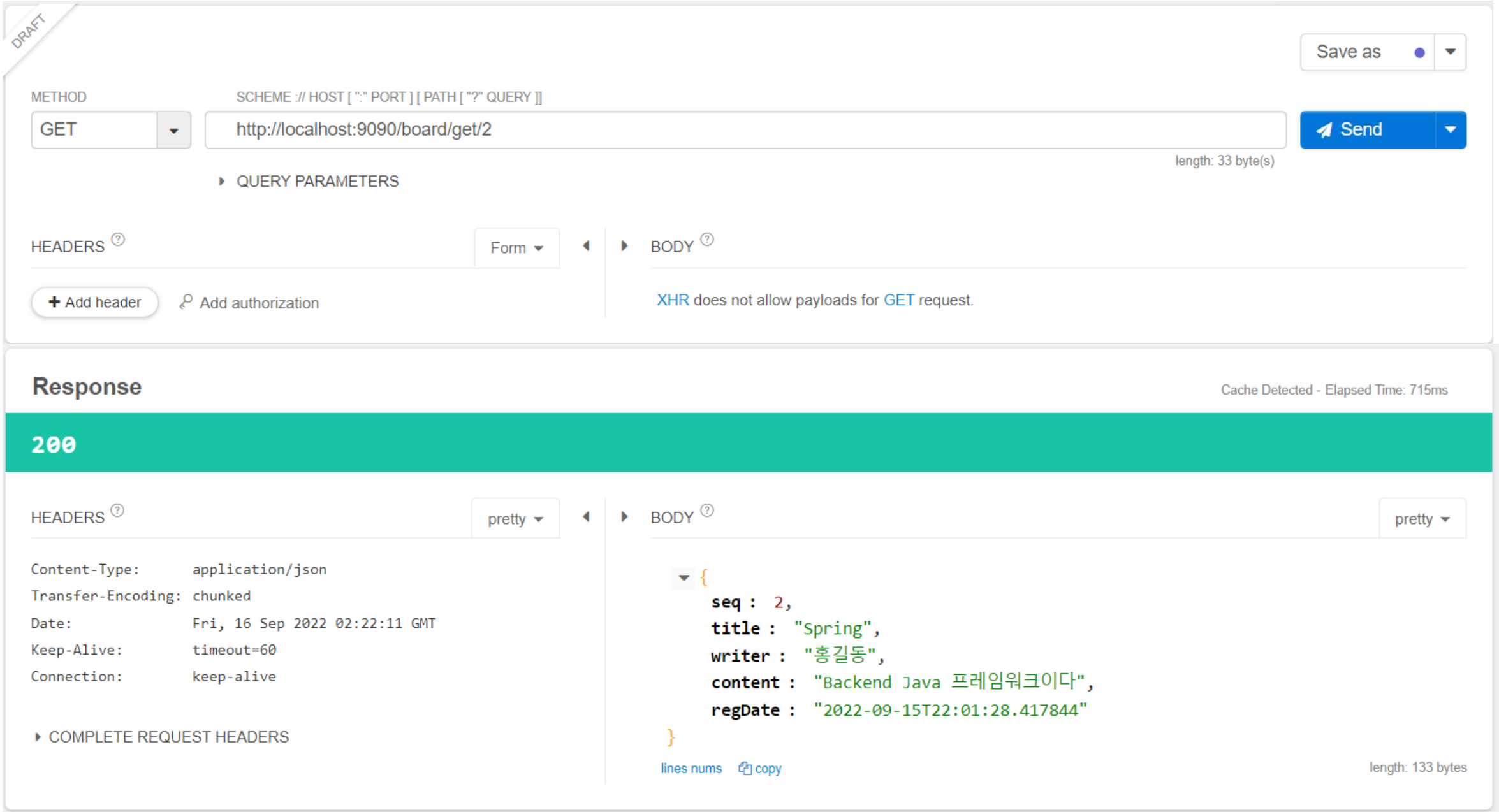
Talend API Tester

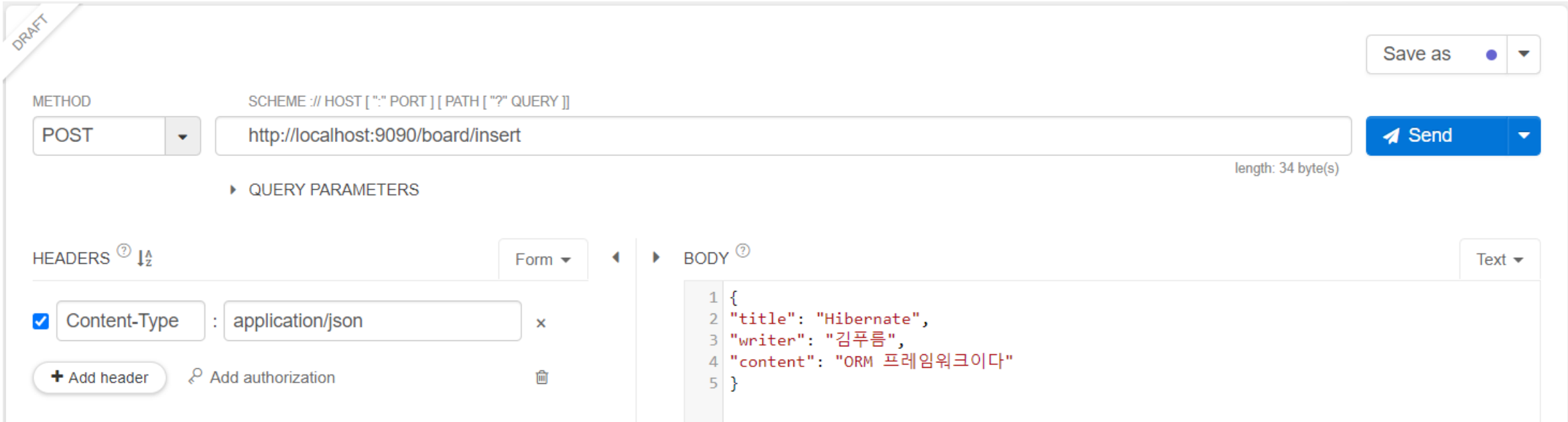


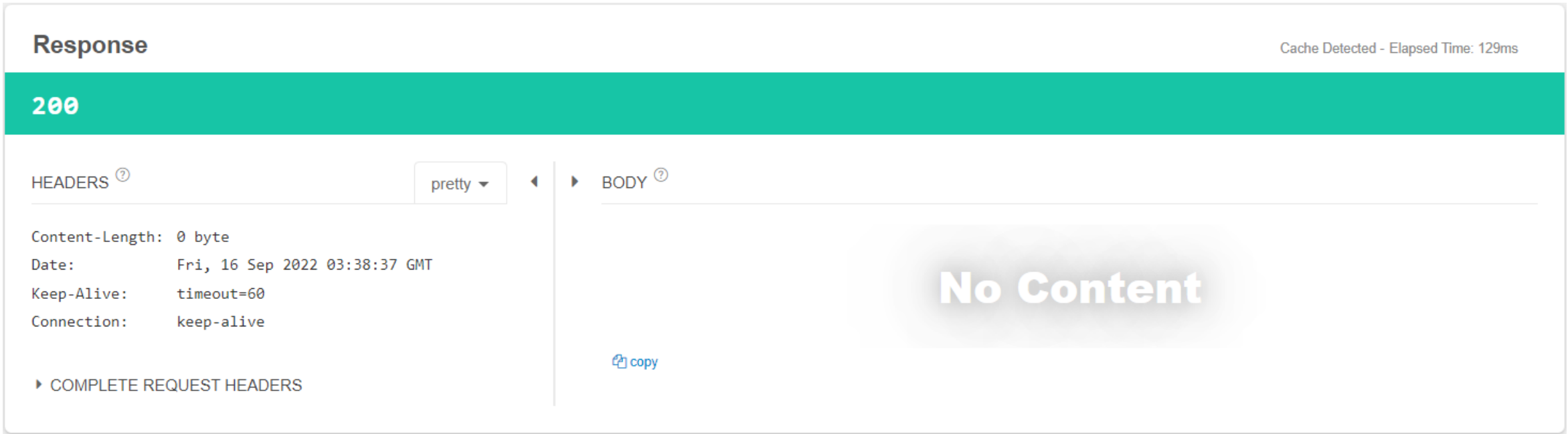
- https://chrome.google.com/webstore/category/extensions?hl=ko
- Talend API 검색 → [Talend API Tester – Free Edition] 선택

Talend API Tester




DRAFT
Save as
METHOD
SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]
GET
http://localhost:9090/board/get/2
Send
length: 33 byte(s)
QUERY PARAMETERS
HEADERS
Form
BODY
Add header
Add authorization
XHR does not allow payloads for GET request.
Response
Cache Detected - Elapsed Time: 715ms
200
HEADERS
pretty
BODY
pretty
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 16 Sep 2022 02:22:11 GMT
Keep-Alive: timeout=60
Connection: keep-alive
COMPLETE REQUEST HEADERS
{
seq : 2,
title : "Spring",
writer : "홍길동",
content : "Backend Java 프레임워크이다",
regDate : "2022-09-15T22:01:28.417844"
}
lines nums
copy
length: 133 bytes


DRAFT
Save as
METHOD
SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]
POST
http://localhost:9090/board/insert
Send
length: 34 byte(s)
QUERY PARAMETERS
HEADERS
Form
BODY
Text
Content-Type : application/json
Add header
Add authorization
{
"title": "Hibernate",
"writer": "김푸름",
"content": "ORM 프레임워크이다"
}


Response
Cache Detected - Elapsed Time: 129ms
200
HEADERS
pretty
BODY
Content-Length: 0 byte
Date: Fri, 16 Sep 2022 03:38:37 GMT
Keep-Alive: timeout=60
Connection: keep-alive
COMPLETE REQUEST HEADERS
No Content
copy

DRAFT

Save as

METHOD

PUT

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:9090/board/update

Send

length: 34 byte(s)

QUERY PARAMETERS

HEADERS

Form

Content-Type : application/json

Add header

Add authorization

BODY

Text

```
{
  "seq": 37,
  "title": "JAVA",
  "content": "가장 많이 사용되는 프로그램언어이다"
}
```

Response

Cache Detected - Elapsed Time: 87ms

200

HEADERS

pretty

```
Content-Length:  0 byte
Date:            Fri, 16 Sep 2022 03:45:45 GMT -1s
Keep-Alive:      timeout=60
Connection:      keep-alive
```

COMPLETE REQUEST HEADERS

BODY

No Content

copy

METHOD

GET

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:9090/board/list

Send

length: 32 byte(s)

QUERY PARAMETERS

+ Add query parameter

HEADERS

Form

BODY

+ Add header

Add authorization

XHR does not allow payloads for GET request.

Response

Cache Detected - Elapsed Time: 425ms

200

HEADERS

pretty

Content-Type:	application/json
Transfer-Encoding:	chunked
Date:	Sun, 02 Oct 2022 02:55:26 GMT
Keep-Alive:	timeout=60
Connection:	keep-alive

COMPLETE REQUEST HEADERS

BODY

pretty


```
[
  {
    seq : 1,
    title : "MVC",
    writer : "홍길동",
    content : "Model, View, Controller 디자인 패턴이다",
    regDate : "2022-10-02T11:21:48.619911"
  },
  {
    seq : 2,
    title : "JPA",
    writer : "이순신",
    content : "자바진영의 ORM 표준이다",
    regDate : "2022-10-02T11:21:48.853285"
```


Top Bottom Collapse Open 2Request Copy Download

DRAFT

Save as

METHOD

DELETE

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:9090/board/delete/3

Send

length: 36 byte(s)

Send request (Alt + Enter)

QUERY PARAMETERS

+ Add query parameter

HEADERS

Form

BODY

XHR does not allow payloads for DELETE request.

+ Add header

Add authorization

Response

Cache Detected - Elapsed Time: 39ms

200

HEADERS

pretty

BODY

```
Content-Length:     0 byte
Date:               Sun, 02 Oct 2022 02:57:58 GMT
Keep-Alive:         timeout=60
Connection:         keep-alive
```

No Content

copy

COMPLETE REQUEST HEADERS

Swagger

Spring Boot

Swagger 란

- Swagger는 API 설계를 문서화하고, 테스트할 수 있는 오픈 소스 프레임워크이다.
- Swagger UI를 통해 웹 브라우저에서 API를 시각화하고 테스트할 수 있다.
- OpenAPI Specification(OAS)에 따라 API를 표준화된 방식으로 기술한다.

Swagger 장점

API 문서화	REST API의 구조와 동작을 명확하게 설명하는 문서를 자동으로 생성
개발자 간의 소통 개선	Swagger는 REST API의 명세서를 시각적으로 제공하여, 팀 내 개발자들 간의 소통을 개선
자동화된 테스트	REST API의 엔드포인트를 테스트할 수 있는 기능을 제공
API 클라이언트 생성	다양한 프로그래밍 언어와 플랫폼에 맞춘 REST API 클라이언트를 자동으로 생성할 수 있는 도구 제공
유지보수 용이성	REST API 코드와 동기화되어 있어, API가 변경될 때마다 문서도 자동으로 업데이트됨.
표준화된 접근 방식	OpenAPI Specification(OAS)을 따르며, 이는 API 설계와 문서화에 대한 표준을 제공

Swagger (pom.xml)

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

application.properties

```
#Swagger 설정
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
```

Swagger 문서화

HTTP 요청 메소드 어노테이션

- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping
- @RequestMapping

API 관련 Swagger 어노테이션

- @Operation
- @ApiResponse
- @Tag

Swagger 어노테이션

@Operation

- API 메소드에 대한 설명
- summary: API 메소드에 대한 간략한 설명.
- description: API 메소드에 대한 자세한 설명.

@ApiResponse

- API 메소드에서 반환하는 응답에 대한 설명
- responseCode: HTTP 상태 코드를 지정한다 (예: 200, 404).
- description: 해당 응답 코드에 대한 설명.
- content: 응답의 미디어 타입과 스키마를 정의한다.

@Tag

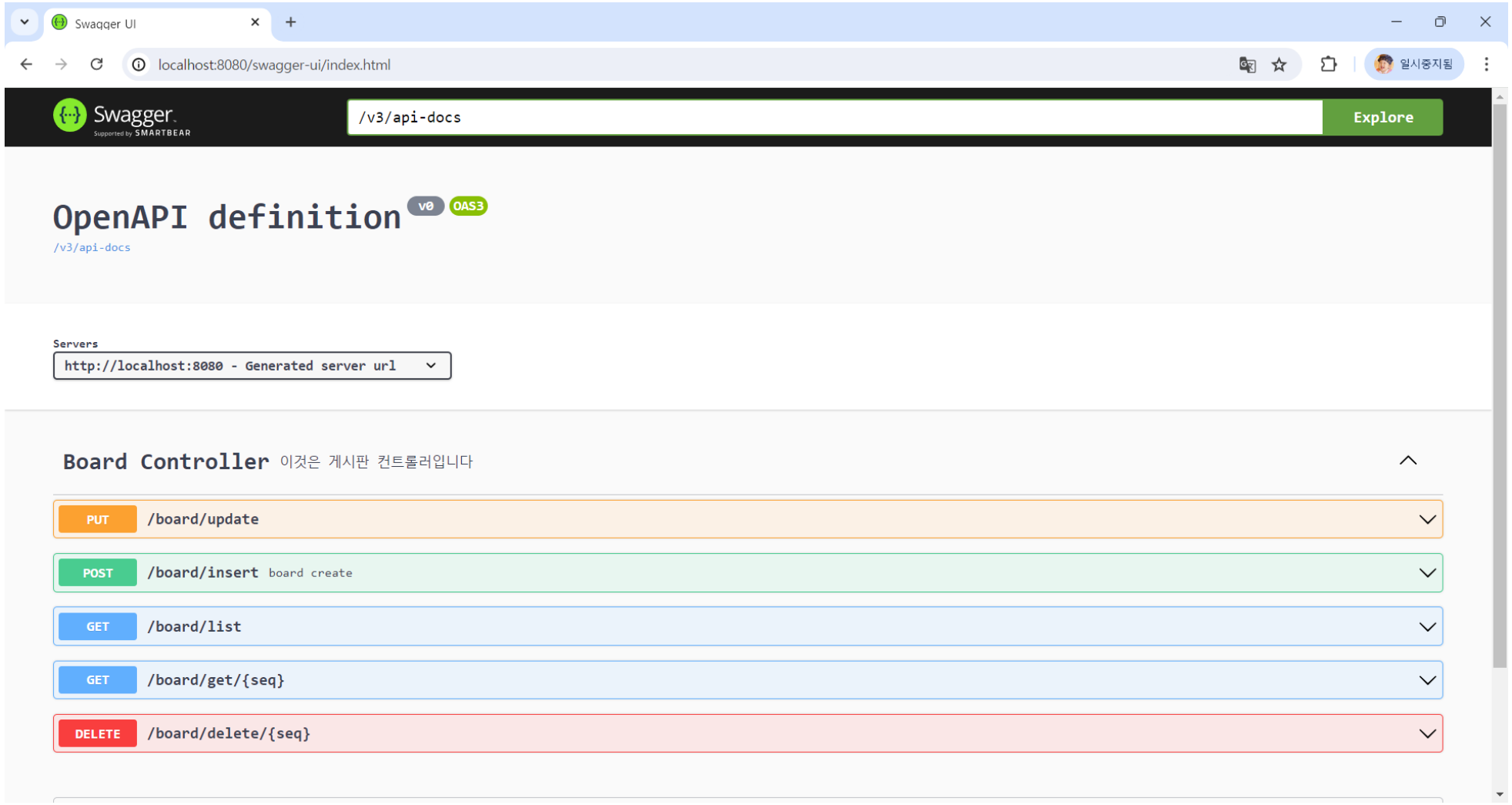
- API 엔드포인트를 논리적으로 그룹화하는데 사용
- name: 태그의 이름. Swagger UI에 표시될 그룹 이름을 정의
- description: 태그에 대한 설명.

@Schema

- 모델 클래스 또는 필드에 대한 메타 데이터 지정
- implementation : 해당 필드 또는 엔티티의 실제 타입 지정
- description : 필드나 클래스에 대한 설명

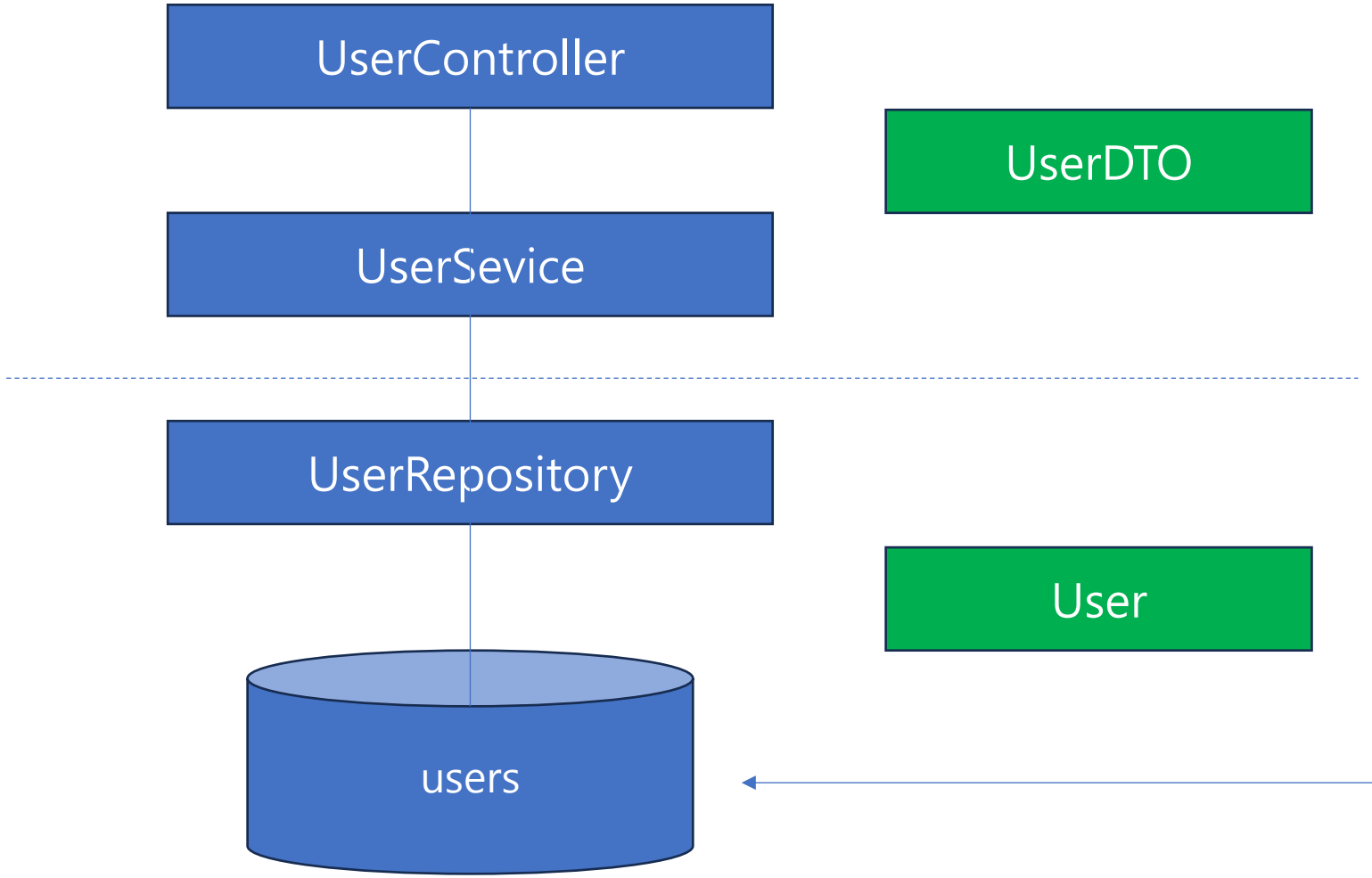
```java
@RequestMapping("/board")
@RestController
@RequiredArgsConstructor
@Tag(name = "Board Controller", description = "이것은 게시판 컨트롤러입니다")
public class BoardController {

	private final BoardService boardService;

	@PostMapping(value = "/insert")
	@Operation(summary = "board create", description = "게시판에 새글을 등록합니다.")
	public void insertBoard(@RequestBody BoardDto dto) {
		boardService.insertBoard(dto);
	}

	@PutMapping(value = "/update")
	public void updateBoard(@RequestBody BoardDto dto) {
		boardService.updateBoard(dto);
	}

	@DeleteMapping(value = "/delete/{seq}")
	public void deleteBoard(@PathVariable Long seq) {
		boardService.deleteBoard(seq);
	}

	@GetMapping(value = "/get/{seq}")
	public BoardDto getBoard(@PathVariable Long seq) {
		return boardService.getBoard(seq);
	}

	@GetMapping(value = "/list")
	public List<BoardDto> getBoardList() {
		return boardService.getBoardList();
	}

}
```

http://localhost:8080/swagger-ui.html



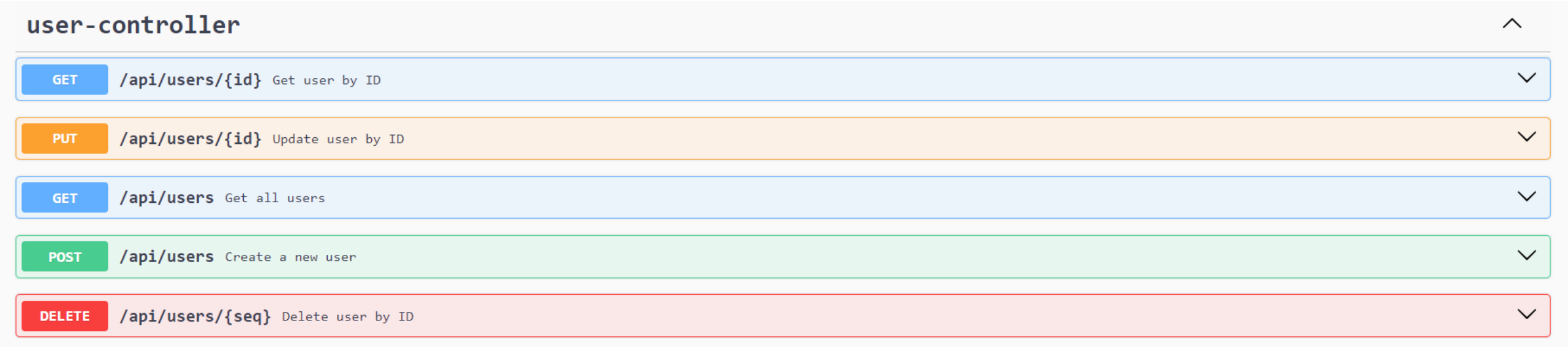
User RESTful API 구현

https://github.com/purum01/springboot : edu_service




```
UserRepositoryTest
@Test
testInsertUsers(){
    실습 데이터 생성
}
```

http://localhost:8080/swagger-ui.html



```java
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @Operation(summary = "Get user by ID", description = "Fetch a user by their ID")
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "Found the user", content = @Content(schema = @Schema(implementation = UserDTO.class))),
        @ApiResponse(responseCode = "400", description = "Invalid ID supplied"),
        @ApiResponse(responseCode = "404", description = "User not found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<UserDTO> getUserById(@PathVariable String id) {
        Optional<UserDTO> user = userService.getUserById(id);
        return user.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(404).body(null));
    }

    @Operation(summary = "Get all users", description = "Fetch all users")
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "Fetched all users", content = @Content(schema = @Schema(implementation = UserDTO.class)))
    })
    @GetMapping
    public ResponseEntity<List<UserDTO>> getAllUsers() {
        List<UserDTO> users = userService.getAllUsers();
        return ResponseEntity.ok(users);
    }

    @Operation(summary = "Create a new user", description = "Add a new user to the system")
    @ApiResponses({
        @ApiResponse(
            responseCode = "201",
            description = "User created successfully",
            content = @Content(schema = @Schema(implementation = UserDTO.class)),
            headers = @Header(name = "Location", description = "The URI of the newly created user", schema = @Schema(type = "string"))),
        @ApiResponse(responseCode = "400", description = "Invalid input"),
        @ApiResponse(responseCode = "500", description = "Internal server error")
    })
```

```java
@PostMapping
    public ResponseEntity<UserDTO> createUser(@RequestBody String name) {
        UserDTO createdUser = userService.createUser(name);
        return ResponseEntity.status(201).header("Location", "/api/users/" + createdUser.getId()).body(createdUser);
    }

    @Operation(summary = "Update user by ID", description = "Update a user's name by their ID")
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "User updated successfully", content = @Content(schema = @Schema(implementation = UserDTO.class))),
        @ApiResponse(responseCode = "400", description = "Invalid ID or input supplied"),
        @ApiResponse(responseCode = "404", description = "User not found")
    })
    @PutMapping("/{id}")
    public ResponseEntity<UserDTO> updateUserById(@PathVariable String id, @RequestBody String name) {
        UserDTO updatedUser = userService.updateUser(id, name);
        return ResponseEntity.ok(updatedUser);
    }

    @Operation(summary = "Delete user by ID", description = "Delete a user by their ID")
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "User deleted successfully"),
        @ApiResponse(responseCode = "400", description = "Invalid ID supplied"),
        @ApiResponse(responseCode = "404", description = "User not found")
    })
    @DeleteMapping("/{seq}")
    public ResponseEntity<Void> deleteUserById(@PathVariable String seq) {
        boolean deleted = userService.deleteUser(seq);
        if (deleted) {
            return ResponseEntity.ok().build();
        } else {
            return ResponseEntity.status(404).build();
        }
    }
}
```