2주차 - 프로젝트 생성 및 API 구현

> 경희대학교 컴퓨터공학부 하계 리턴 백엔드(스프링부트) 스터디 2주차 - 트랙장 최현영

실습 코드 저장소 :

```
https://github.com/Martin0o0/return_study
```

0. 구성 요소 설치

1. JDK 17
2. IntelliJ - **Ultimate**
3. Postman

- https://languagestory.tistory.com/11 : Window 10
- https://somjang.tistory.com/entry/JAVA-Windows-10에서-환경변수-설정하기 - JDK 환경변수 설정
- https://webcache.googleusercontent.com/search?q=cache:QjxYJVeDs58J:https://code-lab1.tistory.com/256&cd=1&hl=ko&ct=clnk&gl=kr&client=safari : MacOS
- https://www.jetbrains.com/ko-kr/idea/ : IntelliJ 설치

- https://goddaehee.tistory.com/215 - JetBrains 대학생 인증
- https://www.postman.com - postman

1. REST 통신

1. REST란?
   - 주고받는 자원(Resource)에 이름을 규정하고 URI에 명시해 HTTP 메서드(GET, POST, PUT, DELETE)를 통해 해당 자원의 상태를 주고받는 아키텍처를 의미
2. REST API란?
   - 먼저 API는 Application Programming Interface'의 약자로, 애플리케이션에서 제공하는 인터페이스를 의미한다.
   - API를 통해 서버 또는 프로그램 사이를 연결할 수 있습니다. 즉, REST APT는 REST 아키텍처를 따르는 인터페이스라고 볼 수 있습니다.
   - REST 아키텍처를 구현하는 웹 서비스를 **RESTful**하다'라고 표현함
3. 특징

- 유니폼 인터페이스
  - 유니폼 인터페이스란 '일관된 인터페이스'를 의미한다.
  - REST 서버는 HTTP 표준 전송 규약을 따르기 때문에 어떤 프로그래밍 언어로 만들어졌느냐와 상관없이 플랫폼 및 기술에 종속되지 않고 타 언어, 플랫폼, 기술 등과 호환해 사용할 수 있다는 것을 의미한다.(Polyglot)
- 무상태성
  - REST'는 '무상태성(stateless)'이라는 특징을 가진다.
  - 무상태성이란 서버에 상태 정보를 따로 보관하거나 관리하지 않는다는 의미.
  - 서버는 클라이언트가 보낸 요청에 대해 세션이나 쿠키 정보를 별도 보관하지 않기에 모든 요청을 개별적으로 처리한다.
- 클라이언트-서버 아키텍처
  - REST 서버는 API를 제공하고 클라이언트는 사용자 정보를 관리하는 구조로 분리해 설계합니다. 이 구성은 서로에 대한 의존성을 낮춘다.

4. REST URI 설계 규칙

- **URI는 인터넷상의 리소스 “자원 그 자체”를 식별하는 ‘고유한’ 문자열 시퀀스**
  - khu-return.site 는 도메인이기도 하지만, 서버의 식별자이자 고유한 문자열 시퀀스이다.
- URL : URI에 실제 리소스의 위치를 합친 구체적인 주소를 의미
  - https://khu-return.site/static/profileImg/4054d64f-798e-46c5-8d42-2494232e7d92.JPG

```
https://khu-return.site/static/profileImg/4054d64f-798e-46c5-8d42-2494232e7d92.JPG
```

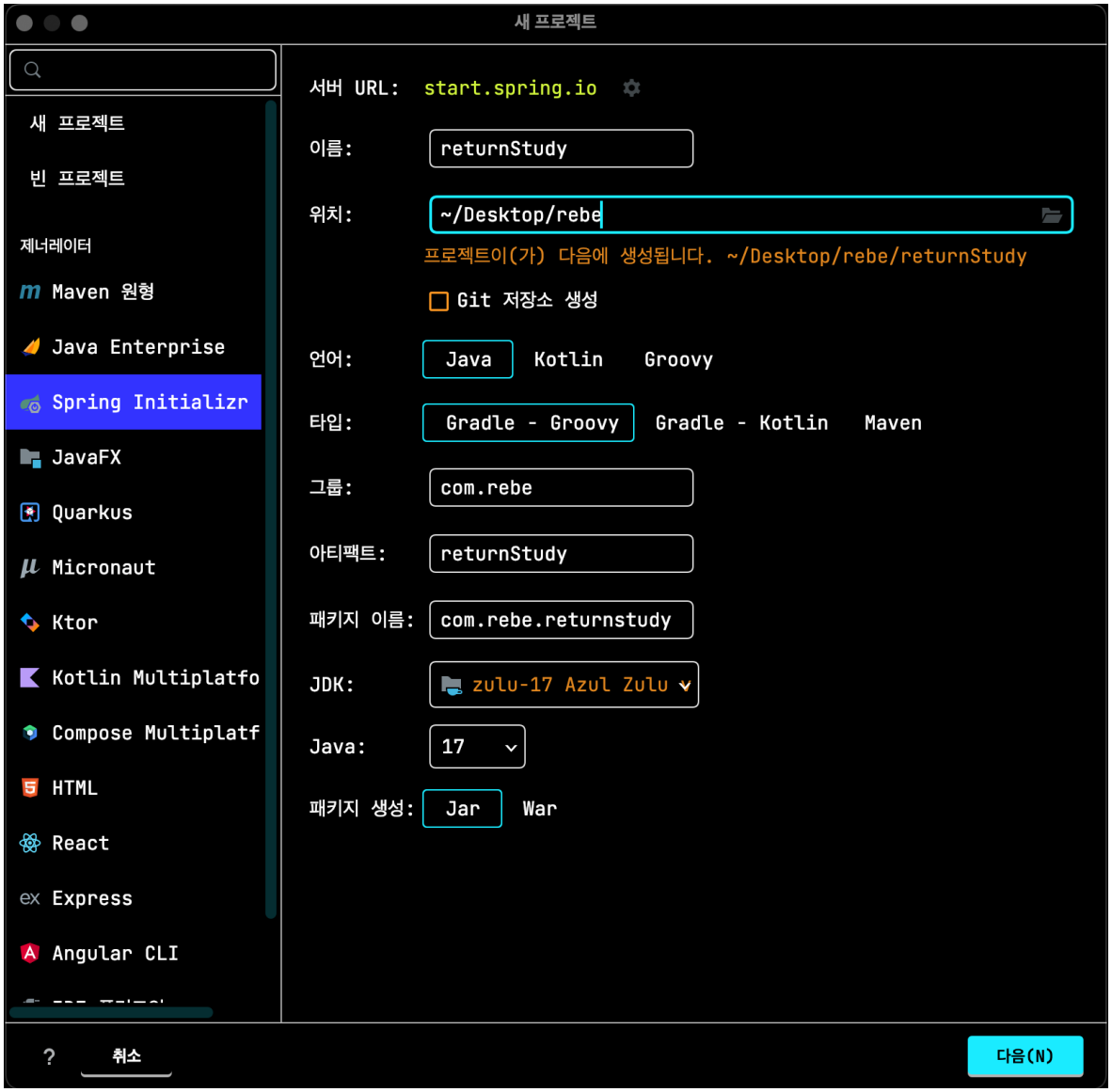
  위와같이 리소스에 접근하는 방식을 명시하는 프로토콜과 리소스의 경로[PATH]를 결합한 것이 URL이다.
- URI의 마지막에는 '/'를 포함하지 않음
  - 옳은 예) http://ocalhost.com/product
  - 잘못된 예) http://localhost.com/product/
- 언더바(_)는 사용하지 않습니다. 대신 하이픈(-)을 이용한다.
  - 하이픈은 리소스의 이름이 길어지면 사용한다

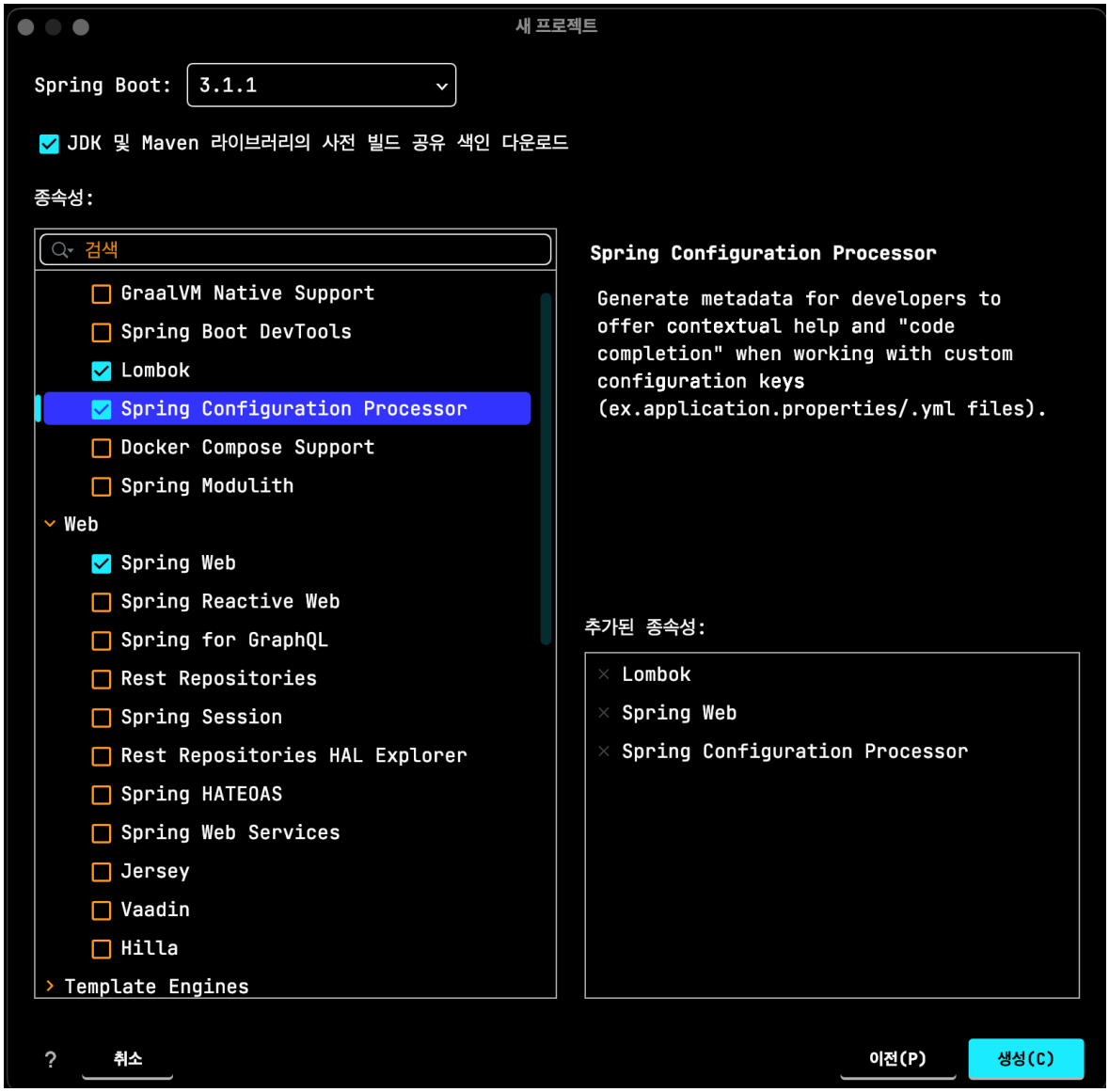
    - 옳은 예) http://localhost.com/provider--company-name
    - 잘못된 예) http://localhost.com/provider_company_name
- URL에는 행위(동사)가 아닌 결과(명사)를 포함한다
    - 옳은 예) http://localhost.com/product/123
    - 잘못된 예) http://ocalhost.com/delete-product/123
    - 행위는 HTTP 메서드로 표현할 수 있어야 함
- URI는 소문자로 작성해야 함
    - URI 리소스 경로에는 대문자 사용을 피하는 것이 좋다.

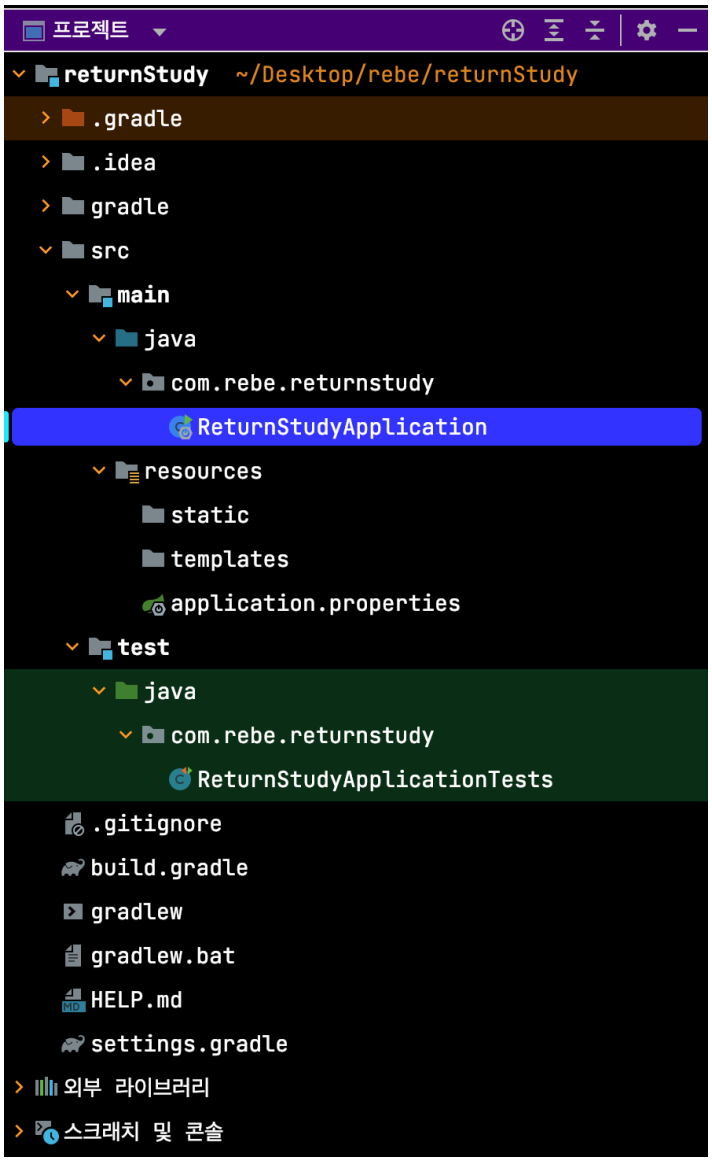
5. Method
    a. GET 메소드: **서버로부터 정보를 요청**할 때 사용하는 Method. 요청한 URL의 정보를 가져와서 클라이언트로 전송.
    b. POST 메소드: 새로 생성할 **데이터를 제출할 때 사용**하는 Method. 클라이언트가 서버에 제출할 데이터를 전송.
    c. PUT 메소드: 기존에 존재하는 **데이터를 업데이트**할 때 사용하는 Method. 클라이언트가 서버에 업데이트할 데이터를 전송.
    d. DELETE 메소드: **서버의 데이터를 삭제**할 때 사용하는 Method. 클라이언트가 삭제할 데이터를 요청.

1. 프로젝트 시작

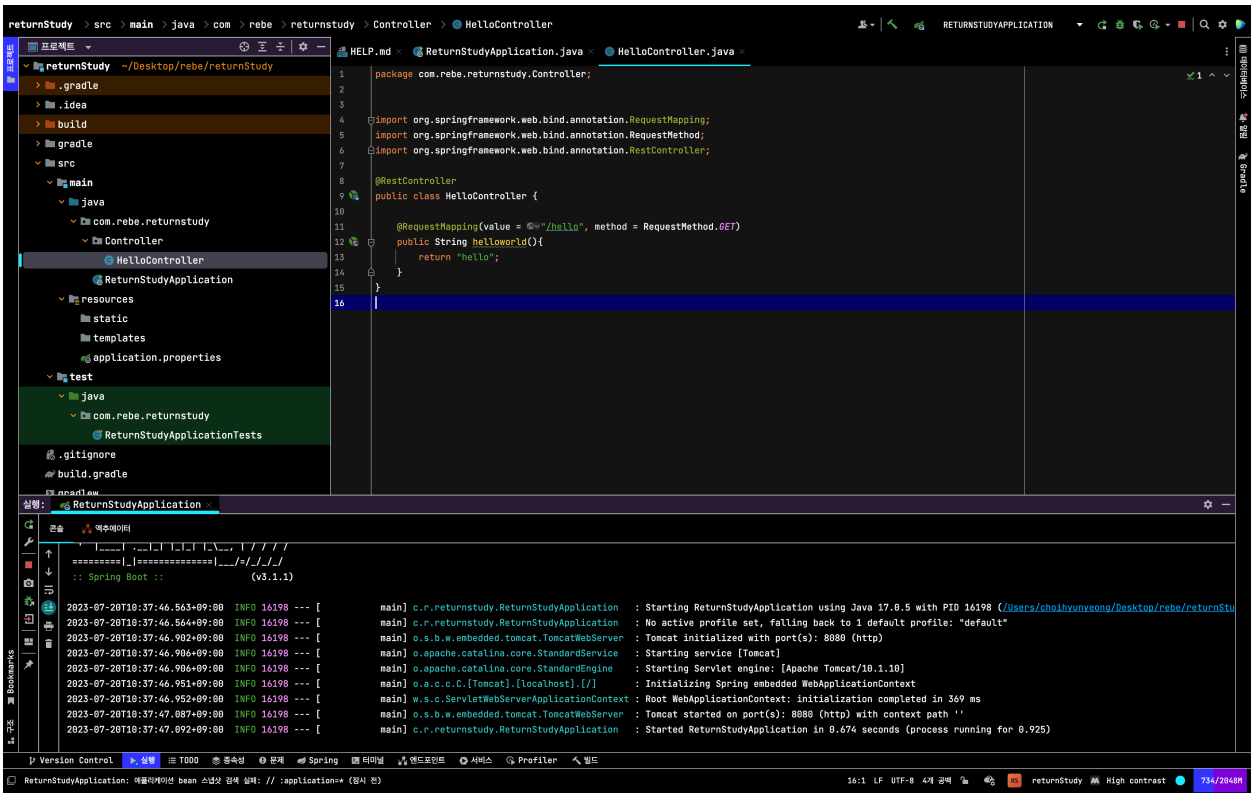
Postman에 `http://localhost:8080/hello` 라는 URL을 입력했을 때 응답값에 "hello"라는 문구를 출력하는 웹 프로그램을 작성해 보자.

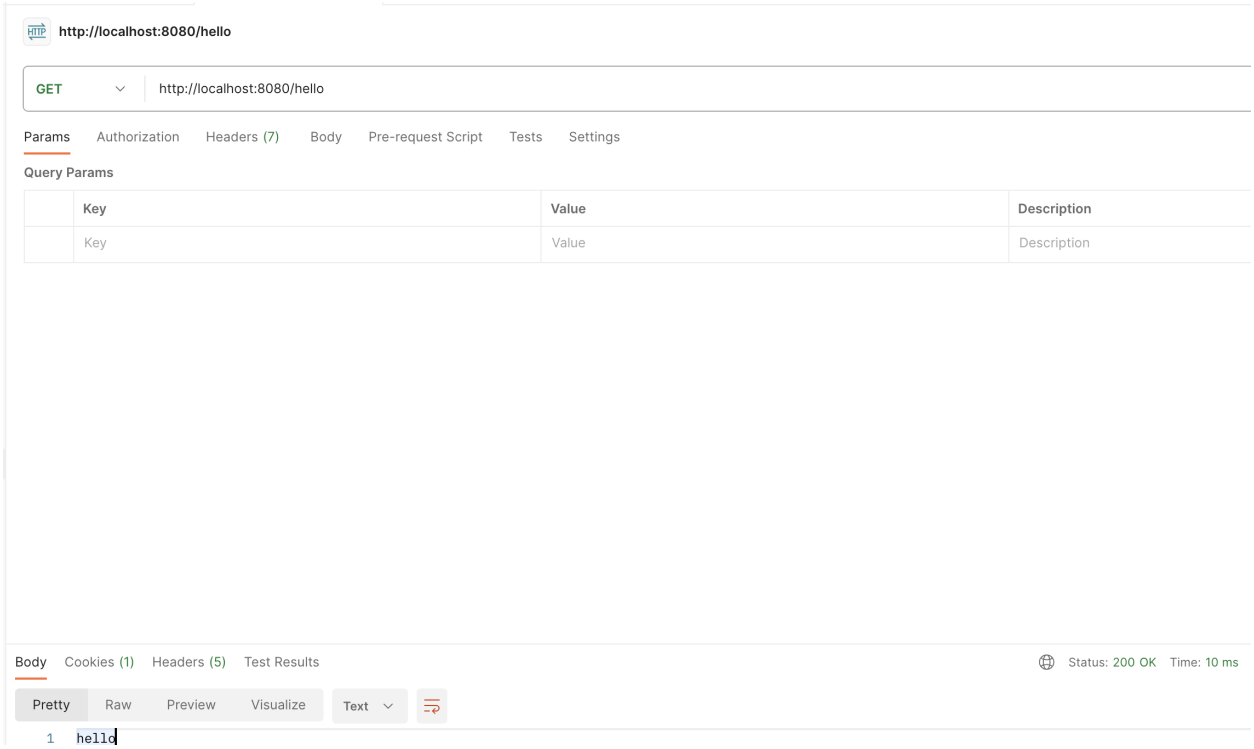

새 프로젝트
새 프로젝트
빈 프로젝트
제너레이터
Maven 원형
Java Enterprise
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatfo
Compose Multiplatf
HTML
React
Express
Angular CLI
서버 URL: start.spring.io
이름: returnStudy
위치: ~/Desktop/rebe
프로젝트이(가) 다음에 생성됩니다. ~/Desktop/rebe/returnStudy
Git 저장소 생성
언어: Java Kotlin Groovy
타입: Gradle - Groovy Gradle - Kotlin Maven
그룹: com.rebe
아티팩트: returnStudy
패키지 이름: com.rebe.returnstudy
JDK: zulu-17 Azul Zulu
Java: 17
패키지 생성: Jar War
?
취소
다음(N)


새 프로젝트
Spring Boot: 3.1.1
JDK 및 Maven 라이브러리의 사전 빌드 공유 색인 다운로드
종속성:
검색
GraalVM Native Support
Spring Boot DevTools
Lombok
Spring Configuration Processor
Docker Compose Support
Spring Modulith
Web
Spring Web
Spring Reactive Web
Spring for GraphQL
Rest Repositories
Spring Session
Rest Repositories HAL Explorer
Spring HATEOAS
Spring Web Services
Jersey
Vaadin
Hilla
Template Engines
Spring Configuration Processor
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).
추가된 종속성:
Lombok
Spring Web
Spring Configuration Processor
?
취소
이전(P)
생성(C)


프로젝트
returnStudy ~/Desktop/rebe/returnStudy
.gradle
.idea
gradle
src
main
java
com.rebe.returnstudy
ReturnStudyApplication
resources
static
templates
application.properties
test
java
com.rebe.returnstudy
ReturnStudyApplicationTests
.gitignore
build.gradle
gradlew
gradlew.bat
HELP.md
settings.gradle
외부 라이브러리
스크래치 및 콘솔


returnStudy > src > main > java > com > rebe > returnstudy
프로젝트
returnStudy ~/Desktop/rebe/returnStudy
.gradle
.idea
gradle
src
main
java
com.rebe.returns
ReturnStudyApp
resources
test
.gitignore
build.gradle
gradlew
gradlew.bat
HELP.md
settings.gradle
외부 라이브러리
스크래치 및 콘솔
HELP.md
ReturnStudyApplication.java
rgs) {
udyApplication.class, args);
새로 만들기
잘라내기 ⌘X
복사 ⌘C
경로/참조 복사...
붙여넣기 ⌘V
사용 위치 찾기 ⌥F7
파일에서 찾기... ⇧⌘F
파일에서 바꾸기... ⇧⌘R
분석
리팩터링
북마크
코드 서식 다시 지정 ⌥⌘L
import 문 최적화 ^⌥O
삭제... ⌦
모듈 'returnStudy.main' 빌드
다시 빌드(E) 'com.rebe.returnstudy' ⇧⌘F9
다음에서 열기
로컬 기록
IDE 수정
디스크에서 다시 로드
다음과 비교... ⌘D
디렉터리를 다음으로 표시
다이어그램
Java 파일을 Kotlin 파일로 변환 ⌥⇧⌘K
Java 클래스
Kotlin 클래스/파일
파일
스크래치 파일 ⇧⌘N
패키지
package-info.java
module-info.java
HTML 파일
Kotlin 스크립트
Kotlin 워크시트
스타일시트
JavaScript 파일
TypeScript 파일
package.json 파일
OpenAPI 사양
Swing UI 디자이너
리소스 번들
EditorConfig 파일
XML 구성 파일
HTTP 요청
다이어그램
경로에 데이터 소스
Version Control
TODO
Profiler
새 디렉터리 또는 패키지 생성
Bookmarks
구조





- 정상적으로 출력된다. ⇒ 현 컴퓨터(localhost)가 웹 서버가 되어 8080 포트에서 실행되어, `http://localhost:8080/hello` 라는 URL을 통해 서버에 요청이 발생하면 "hello" 라는 문구를 반환받는다.
- **HelloController**

```
@RestController
public class HelloController {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String helloworld(){
        return "hello";
    }
}
```

`http://localhost:8080/hello` 와 같은 브라우저의 요청을 처리하기 위해서는 컨트롤러(Controller)가 필요하다. 컨트롤러는 서버에 전달된 클라이언트의 요청(URL과 전달된 파라미터 등)을 처리하는 자바 클래스이다.

- 앞서 컴포넌트 시리즈 중에서 RestController 어노테이션을 기억하자.

> Component 시리즈 : @Component 어노테이션을 통칭하기 위해 사용한 표현이다.
>
> 1. @RestController : Rest API(JSON) 통신할 때 사용하는 컨트롤러 어노테이션
> 2. @Service : 서비스 레이어에서 사용하는 어노테이션
> 3. @Repository : 리파지토리에서 사용하는 어노테이션
> 4. @Configuration : 여러개 빈을 가진 클래스를 등록하기 위한 어노테이션, 대개 외부에서 가져온 라이브러리들의 클래스를 빈으로 등록할 때 사용하는 어노테이션이다.

- @RestController Annotation을 intelliJ에서 Ctrl 을 눌러서 이동해보면 아래와 같은 소스코드가 보여진다. @Controller Annotation에는 @Component Annotation이 있는 것을 확인할 수 있다. @Component Annotation으로 인하여 Spring은 해당 Controller를 Bean으로 등록했던 것이다.

```
A convenience annotation that is itself annotated with @Controller and @ResponseBody.

Types that carry this annotation are treated as controllers where @RequestMapping
methods assume @ResponseBody semantics by default.

NOTE: @RestController is processed if an appropriate HandlerMapping-HandlerAdapter
pair is configured such as the RequestMappingHandlerMapping-
RequestMappingHandlerAdapter pair which are the default in the MVC Java config and
the MVC namespace.

시작 시간: 4.0
작성자:   Rossen Stoyanchev, Sam Brannen
```

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

    /**
     * The value may indicate a suggestion for a logical component name, to be turned
     * into a Spring bean in case of an autodetected component.
     * 반환:     the suggested component name, if any (or empty String otherwise)
     * 시작 시간: 4.0.1
     */
    @AliasFor(annotation = Controller.class)
    String value() default "";

}
```

- 레이어드 아키텍처에서 살펴봤듯, 클라이언트의 요청을 수용하는 컨트롤러 레이어를 구현하기 위해 @RestController라는 어노테이션을 사용한다. 이 어노테이션을 통해 해당 클래스는 컨트롤러를 담당하라고 명시할 수 있고, 빈으로 등록할 수 있다.

2. 디렉토리 구조

src/main/java 디렉터리

src/main/java 디렉터리의 com.mysite.sbb 패키지는 자바 파일을 작성하는 공간이다. 자바 파일로는 HelloController와 같은 스프링부트의 컨트롤러, 폼과 DTO, 데이터 베이스 처리를 위한 엔티티, 서비스 파일등이 있다.

Application.java 파일

모든 프로그램에는 시작을 담당하는 파일이 있다. 스프링부트 애플리케이션에도 시작을 담당하는 파일이 있는데 그 파일이 바로 `<프로젝트명>` + Application.java 파일이다.

```
package com.return.;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ReturnSiteApplication {
    public static void main(String[] args) {
        SpringApplication.run(ReturnSiteApplication.class, args);
    }
}
```

src/main/resources 디렉터리

src/main/resources 디렉터리는 자바 파일을 제외한 HTML, CSS, Javascript, 환경파일 등을 작성하고 정적 리소스를 보관하는 공간이다.

templates 디렉터리

src/main/resources 디렉터리의 하위 디렉터리인 templates 디렉터리에는 템플릿 파일을 저장한다. 템플릿 파일은 HTML 파일 형태로 자바 객체와 연동되는 파일이다. 대표적으로 JSP, 머스태치, 타임리프 등이 존재,

static 디렉터리

static 디렉터리는 프로젝트의 스타일시트(.css), 자바스크립트(.js) 그리고 이미지 파일(.jpg, .png) 등을 저장하는 공간이다.

application.properties 파일

application.properties 파일은 프로젝트의 환경을 설정한다. 환경변수, 데이터베이스 등의 설정을 이 파일에 저장한다.

src/test/java 디렉터리

src/test/java 디렉터리는 프로젝트에서 작성한 파일을 테스트하기 위한 테스트 코드를 작성하는 공간이다. JUnit과 스프링부트의 테스팅 도구를 사용하여 서버를 실행하지 않은 상태에서 src/main/java 디렉터리에 작성한 코드를 테스트할 수 있다.

build.gradle 파일

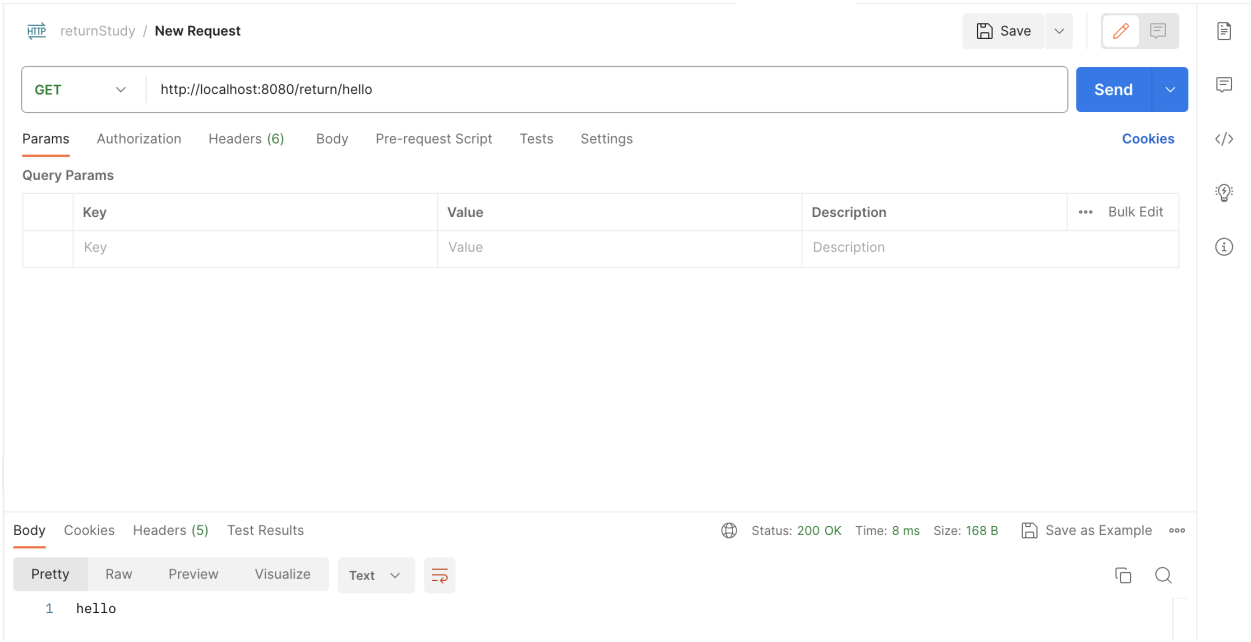
- 빌드 관리 도구 : JVM 혹은 WAS가 프로젝트를 인식하고 실행할 수 있게 내가 작성한 소스코드와 프로젝트에 사용된 파일들(.gradle, .xml, .jar, .properties, .yml 등)을 빌드하는 도구이다.
- 그레이들(Gradle)이 사용하는 환경 파일이다. 그레이들은 그루비(Groovy)를 기반으로 한 빌드 도구로 build.gradle 파일에는 프로젝트를 위해 필요한 플러그인과 라이브러리 등을 기술한다.
- gradle의 경우 별도의 빌드 스크립트를 통하여 사용할 어플리케이션 버전, 라이브러리등의 항목을 설정 할 수 있다.
  - **라이브러리 관리** : 설정된 서버를 통하여 라이브러리를 다운로드 받아 모두 동일한 의존성을 가진 환경을 수정할 수 있다.
  - **프로젝트 관리 : 모든 프로젝트가 일관된 디렉토리 구조를 가지고 빌드 프로세스를 유지하도록 도와준다.**
  - 동적인 빌드(Querry DSL과 같은 동적 쿼리)는 Groovy 스크립트로 플러그인을 호출하거나 직접 코드를 짜면 된다.
  - 프로젝트 환경변수 및 라이브러리 설정 주입 시 프로젝트의 조건(로직을 구현)을 체크할 수 있어서 프로젝트별로 주입되는 설정을 다르게 할 수 있다.

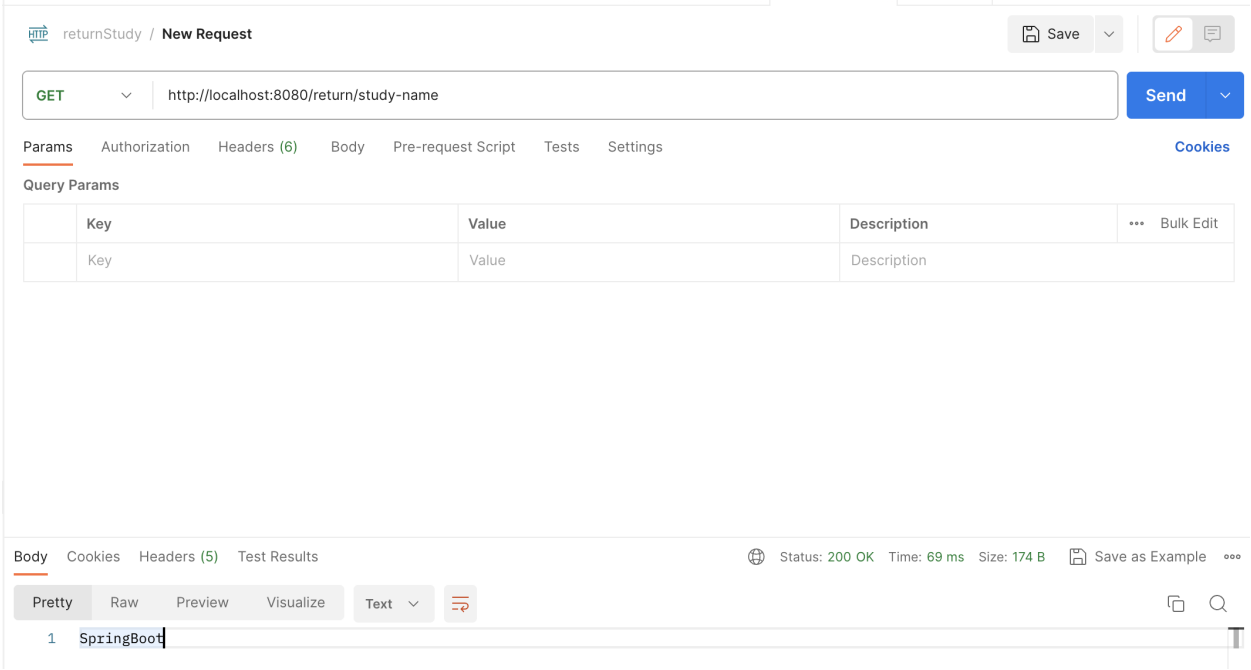
3. @ReqeustMapping

- HTTP의 모든 요청을 받는 어노테이션
  - **메서드의 옵션을 별도로 주지 않으면 모든 메서드를 수용한다.**
- 어노테이션에 특정 경로를 설정하여 클래스에 지정하면 특정 URI(경로)에 해당하는 공통된 요청을 수용한다.
- 메서드 단위에서는 더이상 RequestMapping을 사용하지 않음
- 어노테이션에 특정 경로와 메서드에 지정하면 특정 URL에 해당하는 요청만 받아 해당 메서드를 호출한다.

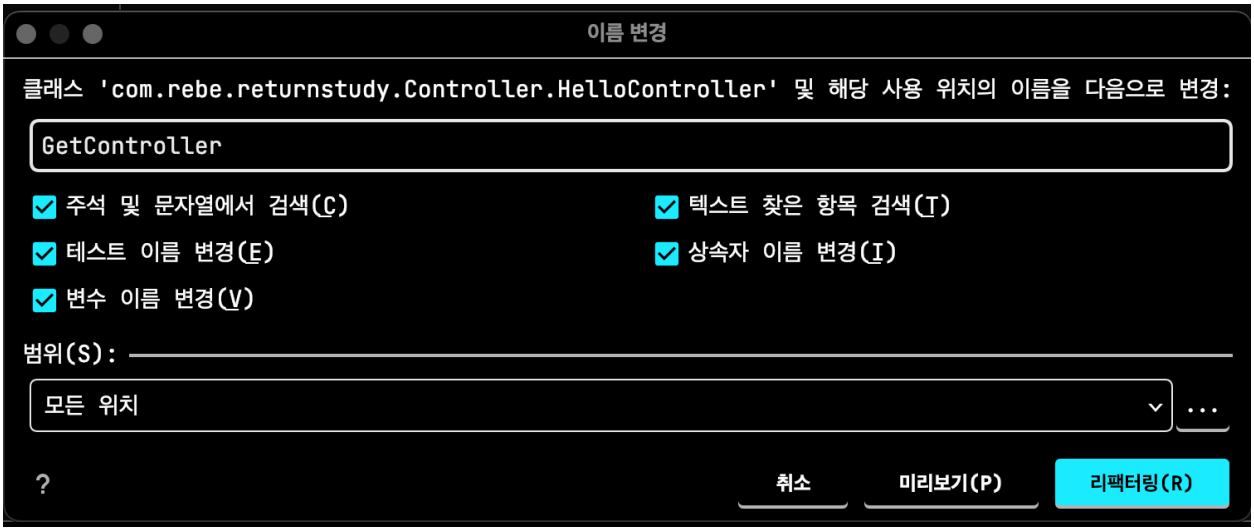
```
package com.rebe.returnstudy.Controller;


import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/return")
public class HelloController {

    //http://localhost:8080/return/hello
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String helloworld() {
        return "hello";
    }
```

```
    // http://localhost:8080/return/study-name
     @RequestMapping(value = "/study-name", method = RequestMethod.GET)
    public String getStudyName() {
        return "SpringBoot";
    }
}
```





4. @GetMapping - Only GET Method



우선, HelloController ⇒ GetController로 이름을 바꿔주자.

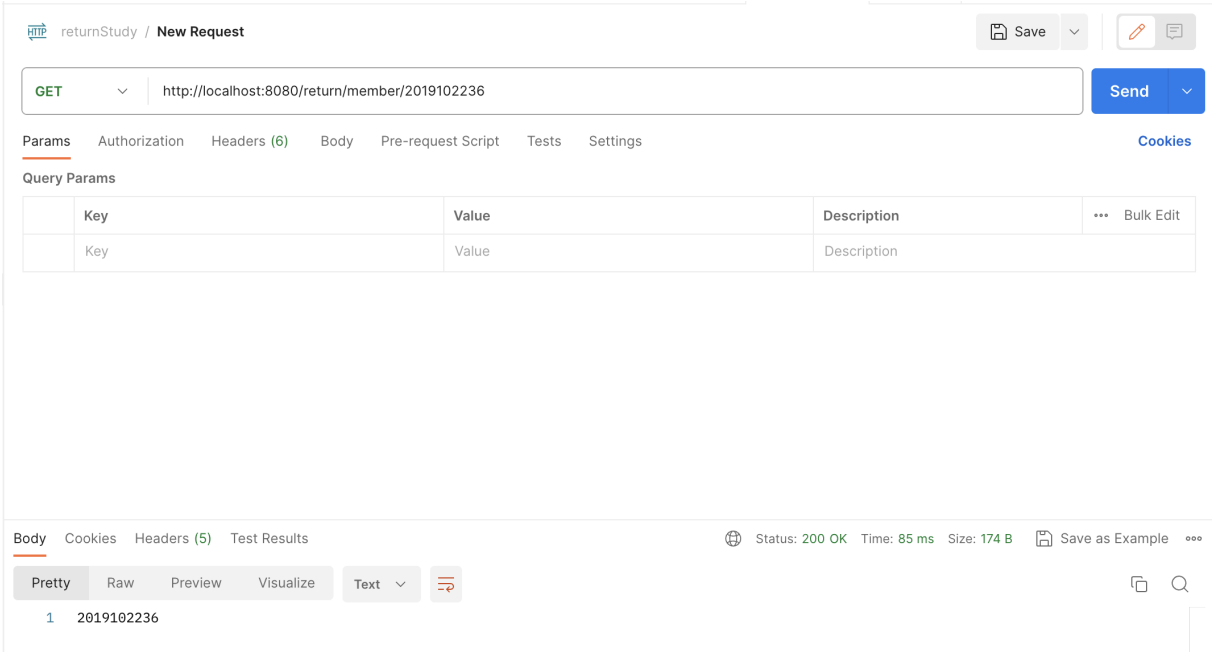
- GET API는 WAS에서 값을 가져올 때 사용하는 API이다.
- @GetMapping 어노테이션은 메서드를 지정할 필요없이, Get 메서드만 요청받기 위한 어노테이션이다.
- 상황에 따라, 특정 리소스를 조회하기 위해 식별자[멤버의 PK]를 포함하여 요청을 보냄

1. 매개 변수가 없는 GET 메서드 구현

```
    // http://localhost:8080/return/study-name
     @GettMapping(value = "/study-name")
    public String getStudyName() {
        return "SpringBoot";
    }
```

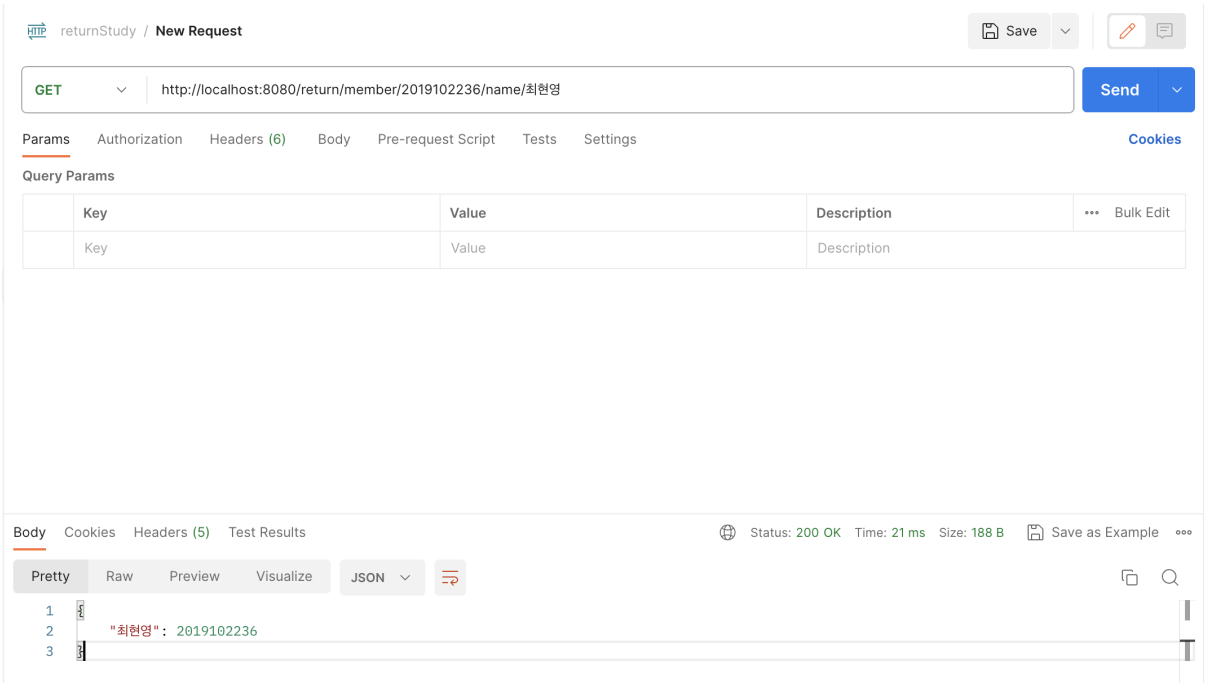
2. 매개변수를 받는 GET 메서드 구현
    - 대부분의 **메서드(함수)는 매개변수**를 포함한다.
    - @PathVariable 어노테이션을 활용
    - 중괄호를 통해 경로의 **어떤 위치의 값(변수명)을** 매개변수로 받을지 지정해야 함
    - URL의 경로 중 어떤 변수명의 정확한 값을 가져오기 위해 value라는 요소를 통해 특정할 수 있고, 입력 파라미터의 변수명으로 매핑할 수 있다.

```
//http://localhost:8080/return/member/2019102236
    @GetMapping(value = "/member/{studentId}")
    public Integer getStudentId(@PathVariable(value = "studentId") Integer studentId){
        System.out.println(studentId);
        return studentId;
    }
```



- 2개 이상의 매개변수도 가능하다.

```
//http://localhost:8080/return/member/2019102236/name/최현영
    @GetMapping(value = "/member/{studentId}/name/{name}")
    public HashMap<String, Integer> getStudentIdAndName(@PathVariable(value = "studentId") Integer studentId, @PathVariable(value = "na
        System.out.println(studentId);
        System.out.println(name);
        HashMap<String, Integer> response = new HashMap<>();
        response.put(name, studentId);
        return response;
    }
```

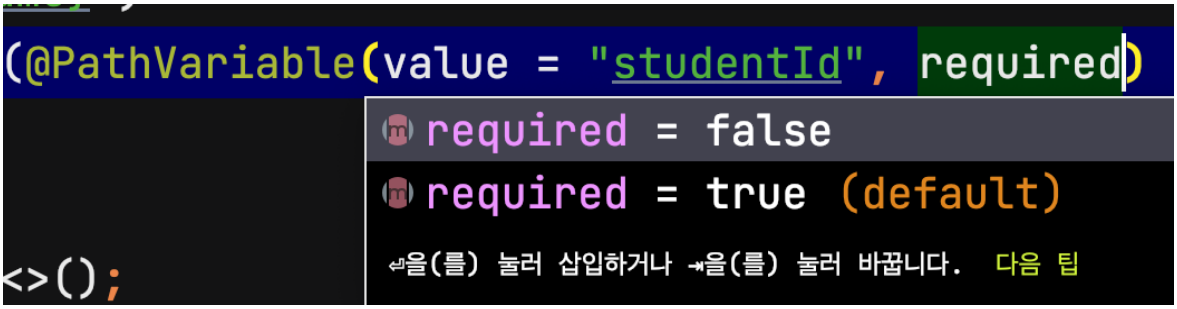


Body Cookies Headers (5) Test Results Status: 200 OK Time: 21 ms Size: 188 B Save as Example

Key	Value
Content-Type	application/json
Transfer-Encoding	chunked
Date	Sun, 23 Jul 2023 16:04:34 GMT
Keep-Alive	timeout=60
Connection	keep-alive

HashMap을 통해 JSON으로 반환됨

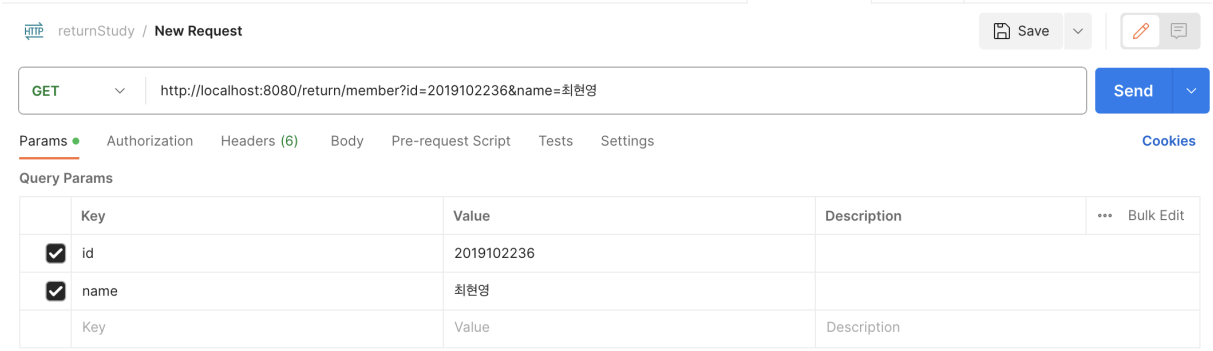
- 값들을 필수 또는 선택적으로 받을 수 있는 옵션을 지원한다.

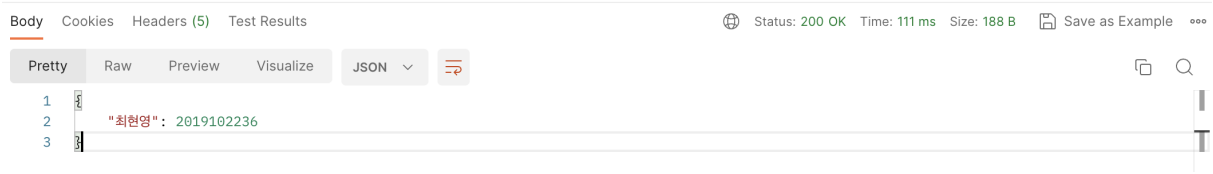


기본값으로는 반드시 해당 변수가 포함되어 요청되어야 하는데, false를 둠으로써 선택적으로 포함하여 요청할 수 있다.

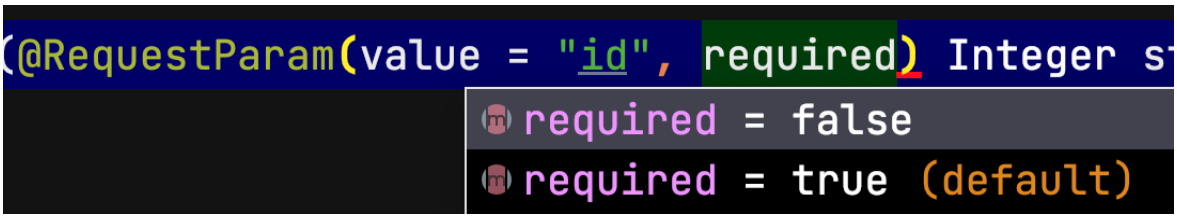
3. 쿼리 형식을 받는 GET 메서드 구현
   - 쿼리 형식 : URI에서 ?를 기준으로 우측에 {key}={value} 형태로 구성된 요청
   - @RequestParam 어노테이션을 통해 쿼리값을 매개변수와 매핑한다.
     - 쿼리 파라미터 중 어떤 key의 정확한 값을 가져오기 위해 value라는 요소를 통해 특정할 수 있고, 입력 파라미터의 변수명으로 매핑할 수 있다.

```
//http://localhost:8080/return/member?id=2019102236&name=최현영
    @GetMapping(value = "/member")
    public HashMap<String, Integer> getStudentInfo(@RequestParam(value = "id") Integer studentId, @RequestParam(value = "name") String
        System.out.println(studentId);
        System.out.println(name);
        HashMap<String, Integer> response = new HashMap<>();
        response.put(name, studentId);
        return response;
    }
```



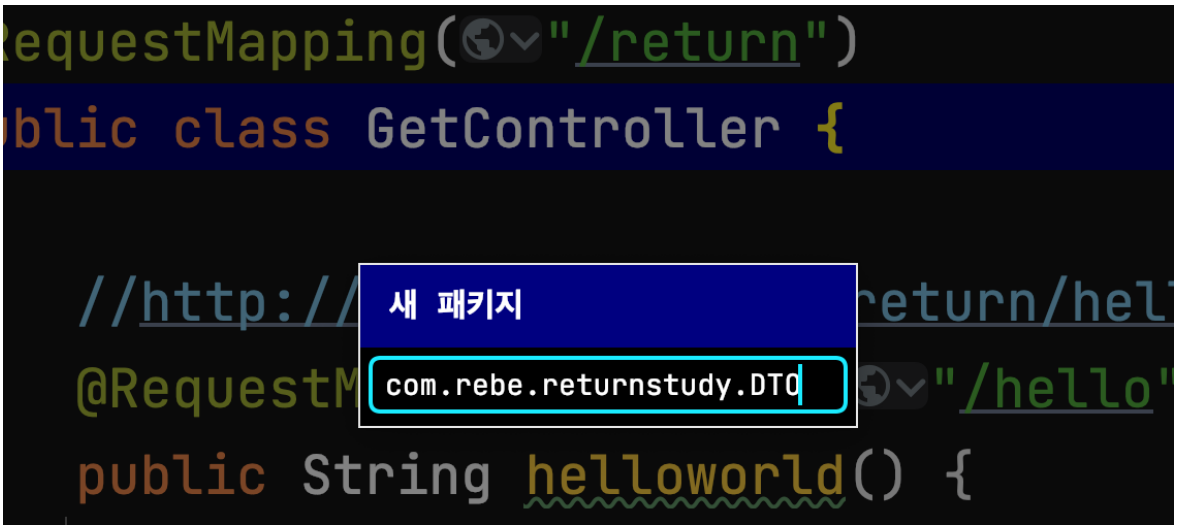


- 쿼리값들을 필수 혹은 선택으로 받을 수 있는 옵션이 존재한다.



해당 옵션을 false로 둔다면, key=value를 선택적으로 포함하여 요청할 수 있다.

4. DTO 객체를 활용한 GET 메서드 구현
   - DTO?
     - 다른 레이어간 데이터 교환에 활용되는 객체
     - 데이터 전송을 위해 사용되는 데이터 컨테이너라고 생각하자.
   - DTO 클래스에서 선언된 필드는 컨트롤러의 메서드에서 쿼리 파리미터(?key1=value1&key1=value2...)의 키와 매핑된다.
   - 정리해보자면, 키값이 많이 존재한다면 DTO 객체를 활용하여 코드의 가독성을 높일 수 있다.

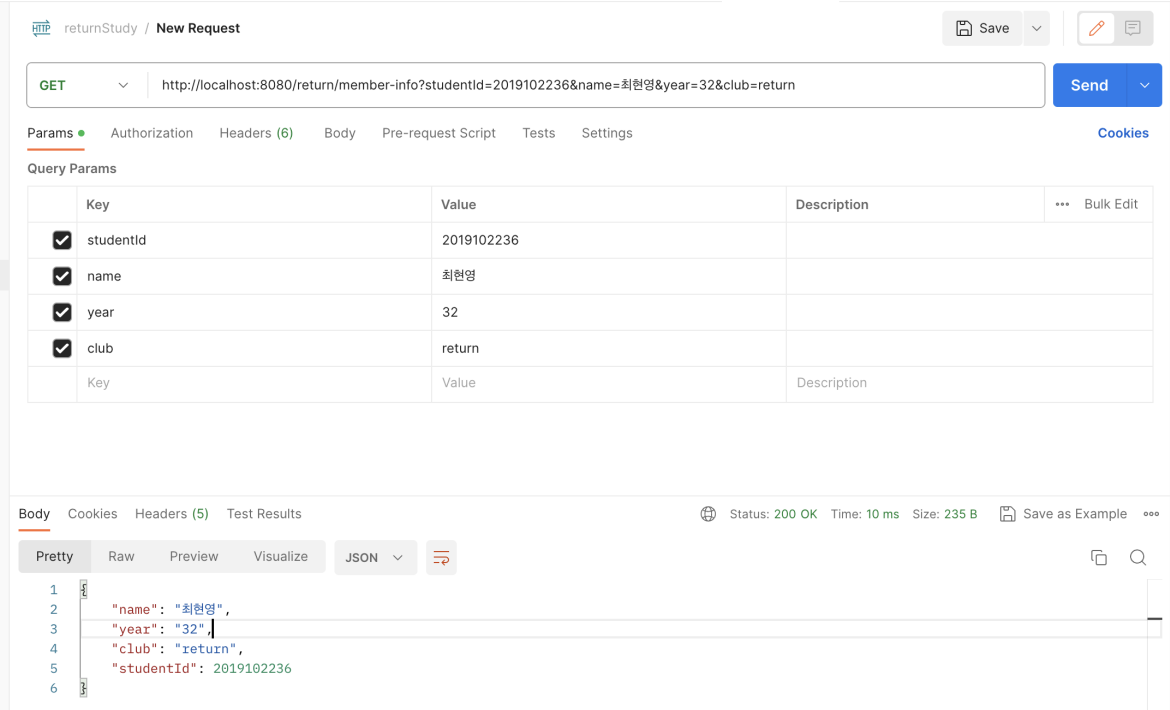


DTO 패키지를 형성해 주자.

```
package com.rebe.returnstudy.DTO;



public class MemberDto {

    Integer studentId;
    String name;
    String year;
    String club;

    public MemberDto(){

    }

    public MemberDto(Integer studentId, String name, String year, String club) {
        this.studentId = studentId;
        this.name = name;
        this.year = year;
        this.club = club;
    }

    public MemberDto(Integer studentId) {
        this.studentId = studentId;
    }

    public Integer getStudentId() {
        return studentId;
    }

    public void setStudentId(Integer studentId) {
        this.studentId = studentId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }

    public String getClub() {
        return club;
```

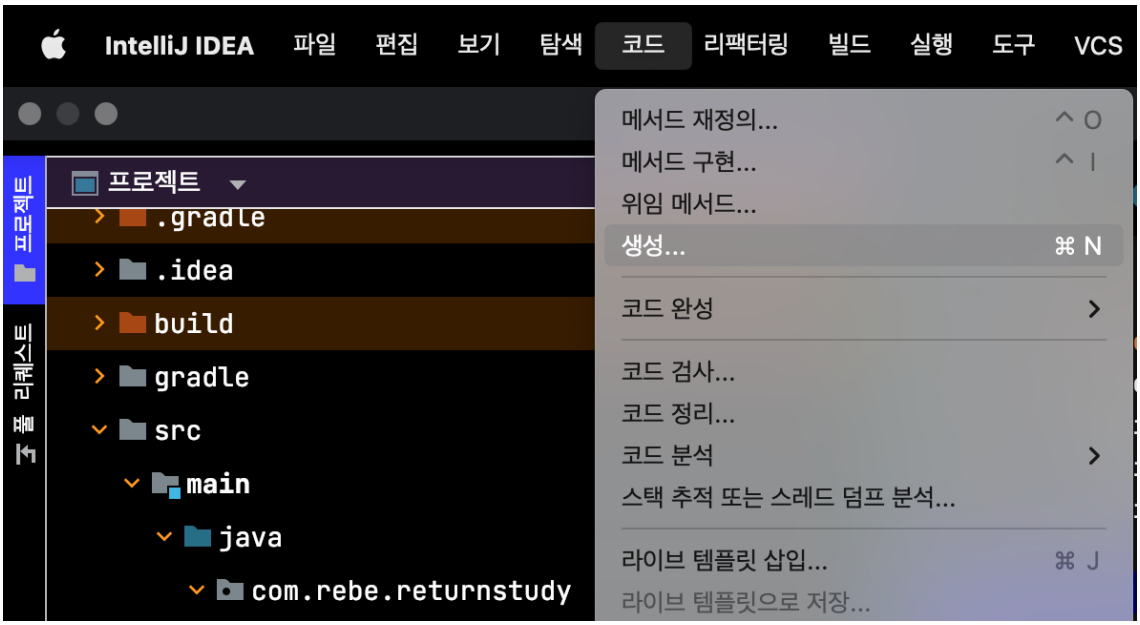
```
    }

    public void setClub(String club) {
        this.club = club;
    }

    @Override
    public String toString() {
        return "MemberDto{" +
                "studentId=" + studentId +
                ", name='" + name + '\'' +
                ", year='" + year + '\'' +
                ", club='" + club + '\'' +
                '}';
    }

}
```

```
//http://localhost:8080/return/member-info?id=2019102236&name=최현영?year=32?club=return
@GetMapping(value = "/member-info")
public MemberDto getStudentFromDTO(MemberDto memberDto){
      return memberDto;
}
```

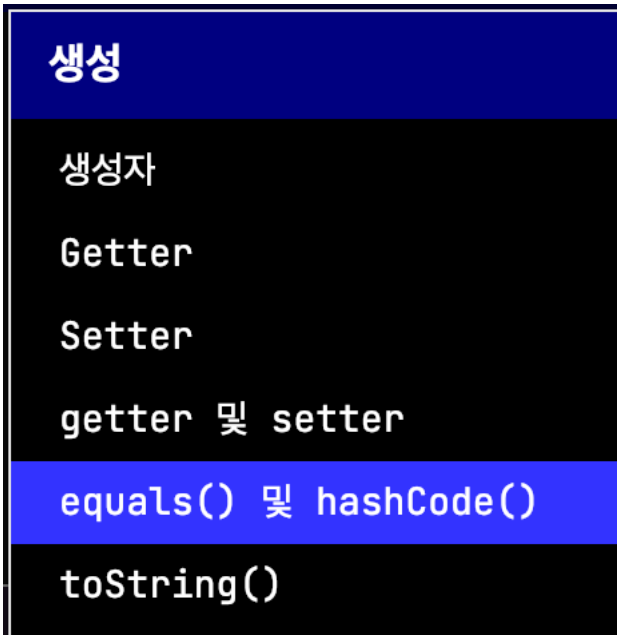


MemberDTO 객체를 메서드의 반환값으로 해주었다. 그 결과 JSON 형식으로 반환되는 것을 살펴볼 수 있다. 이처럼 JSON으로 반환하기 위해 DTO 객체를 두어 레이어 간의 값을 전달하기 위한 용도뿐만 아니라 JSON 형식으로 클라이언트에게 반환하기 위해 사용하기도 한다.

- 인텔리제이의 자동생성 기능[개꿀기능]



현재 클래스에서 코드→ 생성에 들어간다.

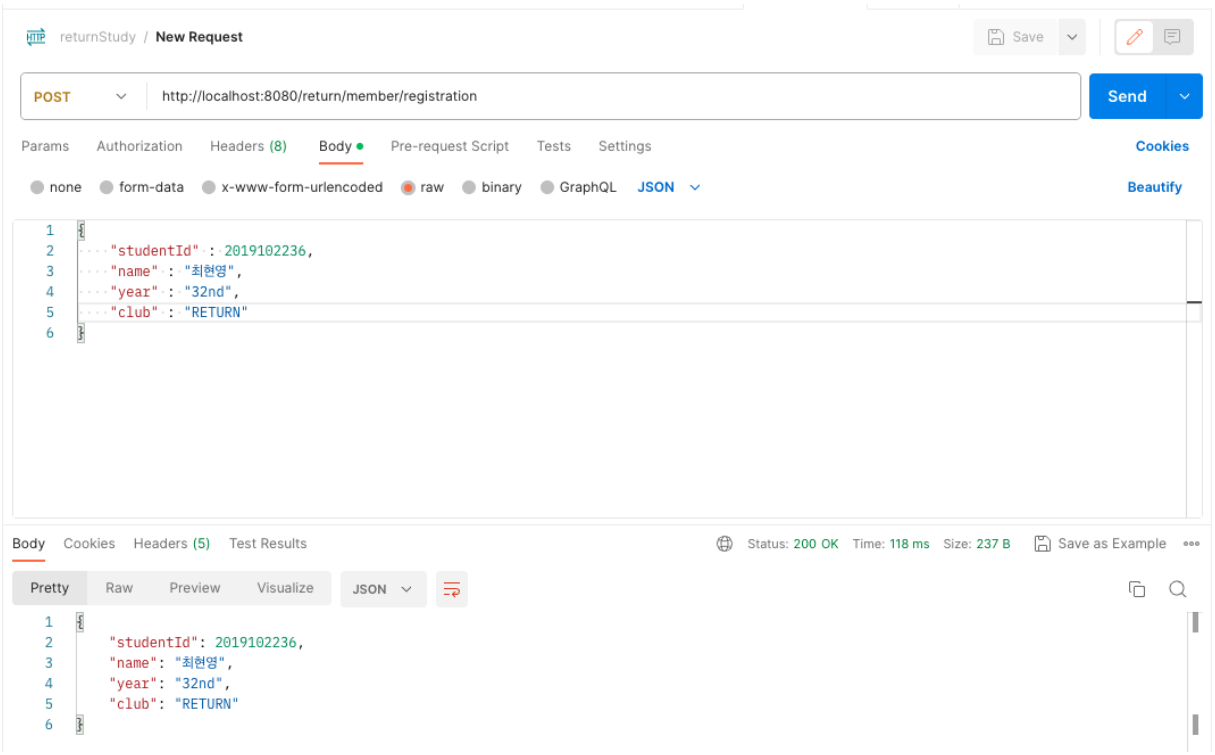


생성자와 getter/setter를 비롯한 Object 클래스의 equals, hashcode, toString을 자동으로 오버라이딩 해준다.

5. @PostMapping - Only POST Method

- 데이터베이스 저장소에 리소스를 저장할 때 사용되는 API
- 저장하고자 하는 리소스는 HTTP Body에 담아 제출한다.
- JSON 형식의 바디를 통한 POST 메서드 구현
  - @RequestBody 어노테이션을 활용
    - HTTP의 Body 내용을 지정된 객체(DTO)에 매핑하는 역할을 담당


```
package com.rebe.returnstudy.Controller;


import com.rebe.returnstudy.DTO.MemberDto;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/return")
public class PostController {

    @PostMapping("/member/registration")
    public MemberDto registerMember(@RequestBody MemberDto memberDto){
        System.out.println(memberDto.toString());
        return memberDto;

    }

}
```

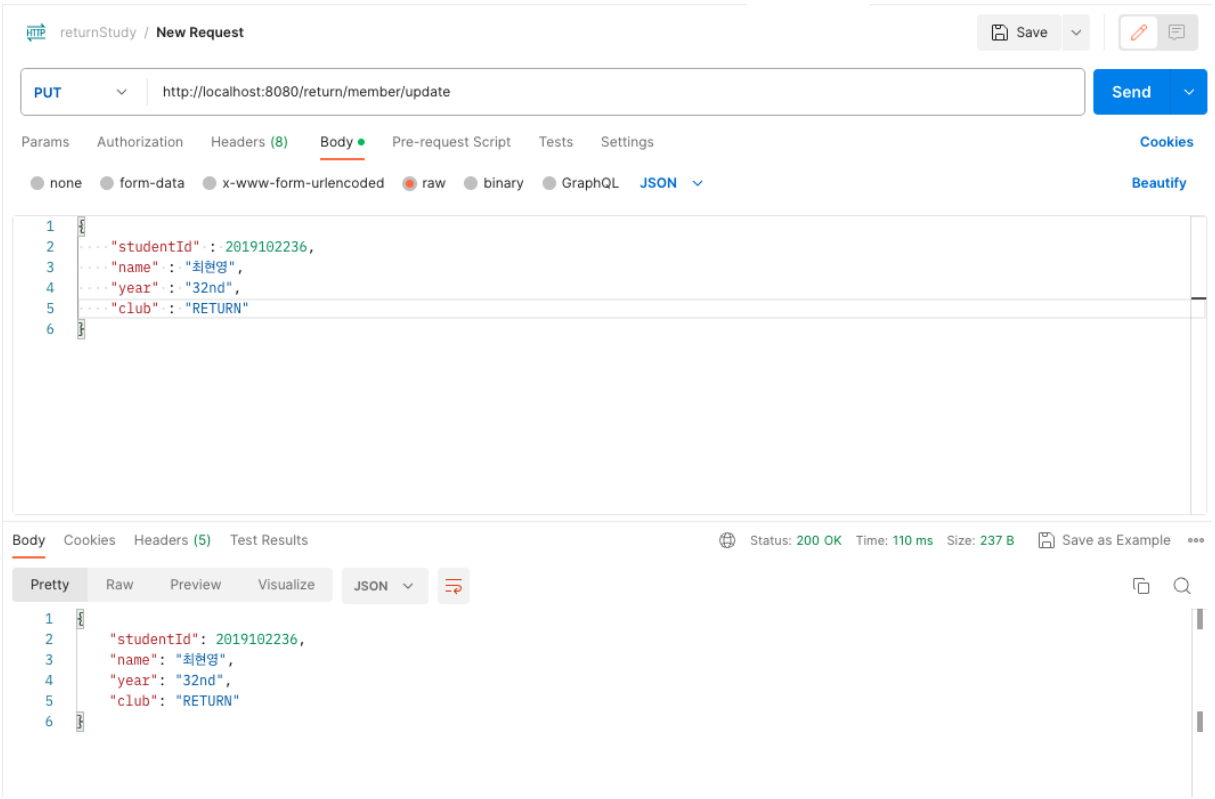



```
MemberDto{studentId=2019102236, name='최현영', year='32nd', club='RETURN'}
```

6. @PutMapping - Only PUT Method

- 리소스의 값을 업데이트하는데 사용됨.
- POST와 마찬가지로 HTTP Body를 통해 서버에 요청함
- 대개, 리소스를 식별할 수 있는 값[멤버의 PK 등]을 받아 해당 리소스를 조회한 뒤, 존재한다면 요청받은 내용으로 갱신을 수행함
- @RequestBody를 활용한 PUT 메서드 구현

```
package com.rebe.returnstudy.Controller;

import com.rebe.returnstudy.DTO.MemberDto;
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;

@RestController
@RequestMapping("/return")
public class PutController {

    @PutMapping("/member/update")
    public MemberDto registerMember(@RequestBody MemberDto memberDto){
        System.out.println(memberDto.toString());
        return memberDto;

    }

}
```

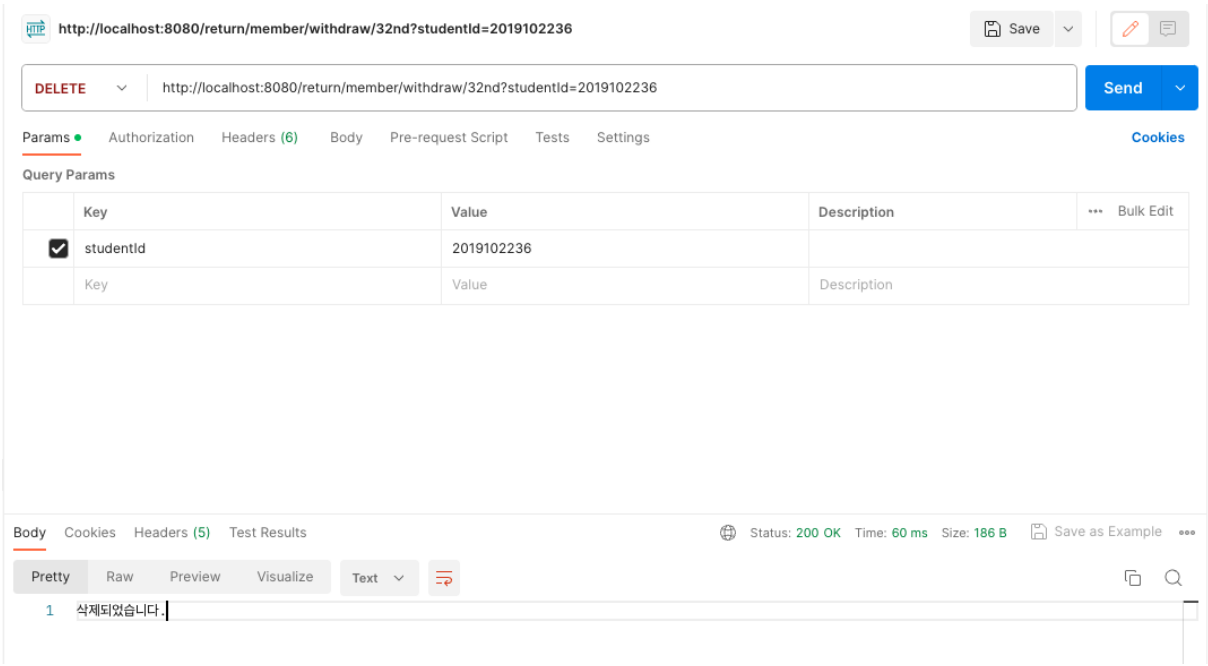


7. @DeleteMapping - Only DELETE Method

- 서버의 리소스를 삭제할 때 사용
- 리소스를 식별할 수 있는 값[멤버의 PK 등]을 받아 해당 리소스를 조회한 뒤, 존재한다면 삭제하는 역할을 수행

1. @PathVariable와 @RequestParam을 활용한 메서드 구현

```
package com.rebe.returnstudy.Controller;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/return")
public class DeleteController {

    //http://localhost:8080/return/member/withdraw/32nd?studentId=2019102236
    @DeleteMapping("/member/withdraw/{year}")
    public String dropOutMember(@PathVariable(value = "year") String year, @RequestParam(value = "studentId") Long studentId){
        return "삭제되었습니다.";

    }

}
```



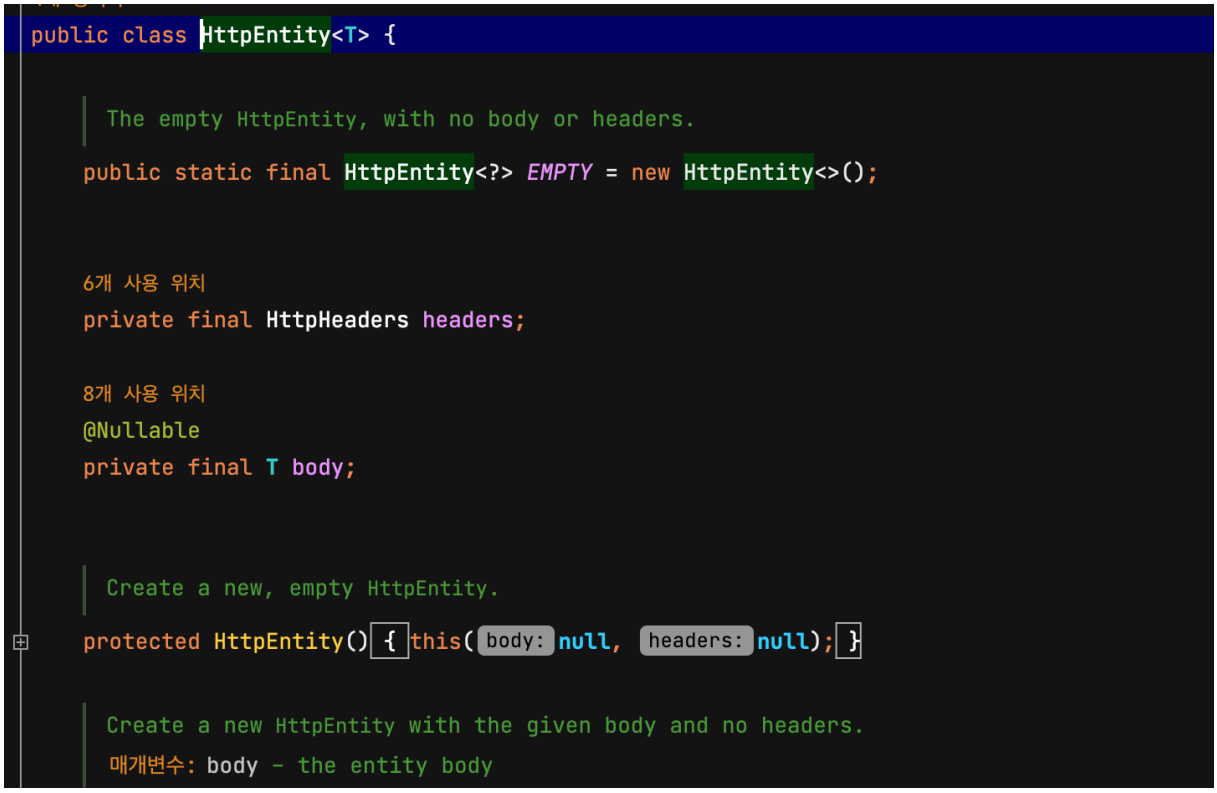
8. ReponseEntity를 활용한 커스텀 Response

- 지금까지는 DTO를 단지 JSON으로 직렬화한 뒤[알아서 해줌], 200 OK[default] status code로만 응답하는 과정을 살펴보았다.
- 1주차에서 우리는 REST 통신일 때의 동작 과정을 살펴보았다.

    - 동작 순서
        1. DispatcherServlet으로 요청(서브릿 컨테이너 내에서 HttpServletRequest 인스턴스를 생성)이 들어오면,
        2. Dispatcherservlet은 핸들러 매핑(Handler Mapping)을 통해 요청 URI에 매핑된 핸들러(컨트롤러)를 탐색한다.
            a. 핸들러 매핑은 **요청 정보(URI, Method)**를 기준으로 어떤 컨트롤러(빈)의 메서드를 사용할지 선정하는 인터페이스이다.
        3. 그리고 핸들러 어댑터(RequestMappingHandlerAdpter)에게 컨트롤러를 호출하도록 요청하고, 어댑터는 컨트롤러를 호출
        4. 핸들러 어댑터에 컨트롤러의 응답이 돌아오면 ResponseEntity 타입로 응답을 가공해 디스패처 서블릿에게 반환
        5. 디스 패처 서블릿은 다시, JSON으로 가공하기 위해 Messageconverter를 호출하여 ResponseEntity 형식의 데이터를 JSON으로 변환하는데, 스프링 부트의 자동 설정 내역을 보면 HttpMessageConverter 인터페이스를 사용한다.
        6. 클라이언트에게 최종적으로 JSON으로 직렬화되어 반환된다.

- 5번 항목에 ResponseEnitty 형식으로 데이터를 직렬화 한다고 하였는데, 대부분의 경우, 헤더의 HTTP status 코드 및 가공된 데이터를 Body에 담아 클라이언트에게 전달한다.

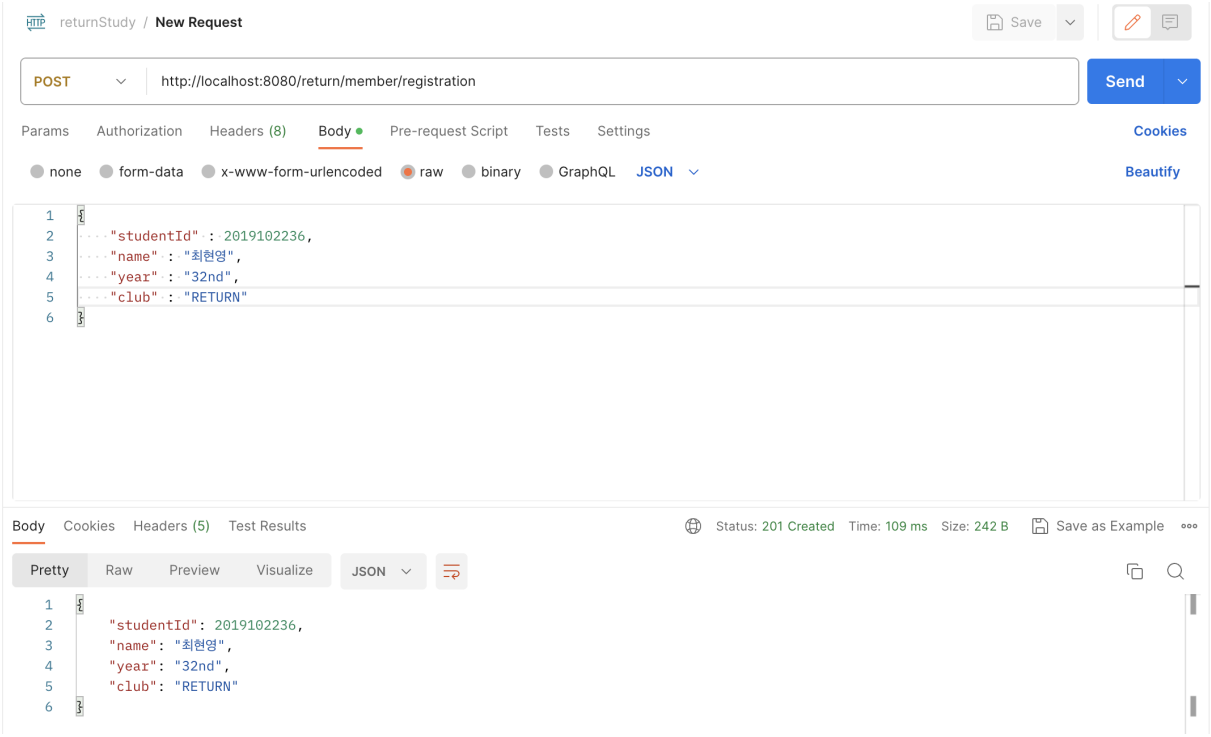
- 따라서, 이제부터 ResponseEntity 객체를 모든 컨트롤러의 메서드 반환타입으로 선언하여 활용할 것이다.
- 스프링 프레임워크에는 HttpEntity라는 클래스가 존재한다.



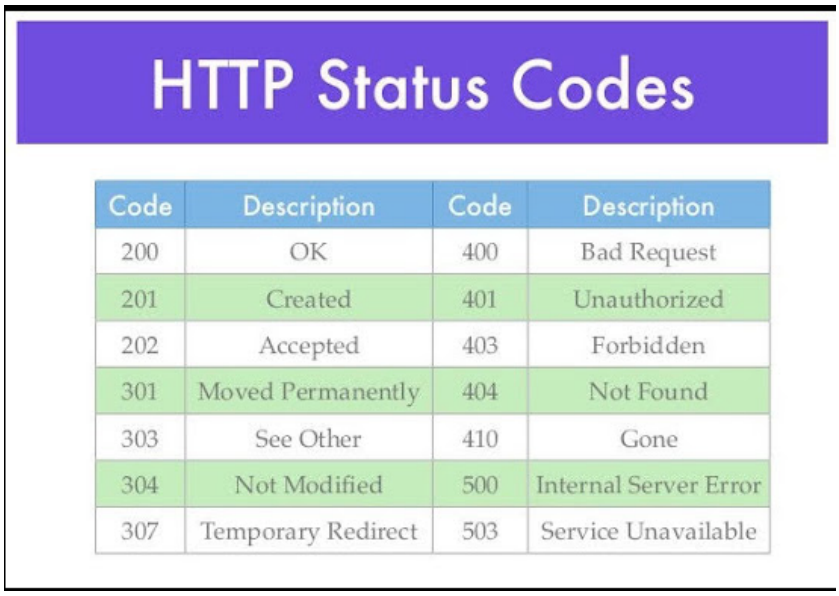
보는바와 같이 Header 와 Body로 구성된 HTTP 요청과 응답을 구성하는 역할을 담당한다.

- ResponseEntity는 HttpEntity를 상속받아 위의 구현한 클래스이다. 서버에 들어온 요청에 대해 응답 데이터를 개발자가 임의로 구성하여 전달할 수 있게 해준다.
- HttpStatus 열거형을 활용하여 응답 코드 변경을 해줄 수 있다.
- Body 부분에 DTO 객체를 담아줌으로써 JSON으로 클라이언트에게 전달된다.

```
package com.rebe.returnstudy.Controller;


import com.rebe.returnstudy.DTO.MemberDto;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/return")
public class PostController {

    @PostMapping("/member/registration")
    public ResponseEntity<MemberDto> registerMember(@RequestBody MemberDto memberDto){
        System.out.println(memberDto.toString());
        return ResponseEntity.status(HttpStatus.CREATED).body(memberDto);
    }
}
```



- Http Status Code 참고



https://youngjinmo.github.io/2020/01/http-codes/