1.개요

주차	6주차	날짜	10.26 수
주제	Hash 충돌 문제 해결하기 Spring Boot 빌드, jdbcTemplate으로 web으로 데이터 response하기		
목표	SpringBoot App을 빌드하고 사용할 수 있다.		
실습	gradle build후 java -jar로 spring boot 앱 실행		
Links	java-algorithm Kyeongrok/java-algorithm (github.com)		
과제	1. Spring Boot 인 액션 번역 - Google Docs 안하신 분들 내일까지 2. 처음부터 jdbcTemplate적용까지 복습		
Previous			
Next			

알고리즘 6주차

https://docs.google.com/document/d/1lxTuGc6qMrP15ZlzpvSInqrAxlbPfOSx_2fE3OVRZLc/edit#heading=h.fk50ky6eaqyk

SpringBoot

교재 소스코드

예제 코드

이 책의 예제 코드는 깃허브 저장소에서 관리됩니다. 아래 깃허브 저장소에서 예제 코드를 확인하고 내려받을 수 있으며, 각 장별로 프로젝트가 구분돼 있어 각 장별로 코드를 확인할 수 있습니다.

- 깃허브 저장소: https://github.com/wikibook/springboot

wikibook/springboot: 《스프링 부트 핵심 가이드》 예제 코드 (github.com)

Spring Boot는 개인용 어플리케이션이 아닌 여러 사람(많은 사람)이 쓰는 어플리케이션 개발을 목적으로 만들어진 프레임웍

스프링 프레임워크(Spring Framework)[1]는 자바(Java) 기반의 애플리케이션 프레임워크로 엔터프라이즈급 애플리케이션을 개발하기 위한 다양한 기능을 제공합니다. 스프링은 목적에 따라 다양한 프로젝

```
01 @RestController
02 public class NoDIController {
03
04     private MyService service = new MyServiceImpl();
05
06     @GetMapping("/no-di/hello")
07     public String getHello() {
08         return service.getHello();
09     }
10
11 }
```

여기에서 개발자가 직접 객체를 컨트롤 한다(제어 한다) - new를 직접 한다

1.1.2 의존성 주입(DI)

의존성 주입(DI; Dependency Injection)이란 제어 역전의 방법 중 하나로, 사용할 객체를 직접 생성하지 않고 외부 컨테이너가 생성한 객체를 주입받아 사용하는 방식을 의미합니다.

스프링에서 의존성을 주입받는 방법은 세 가지가 있습니다.

- **생성자**를 통한 의존성 주입
- **필드 객체 선언**을 통한 의존성 주입
- setter **메서드**를 통한 의존성 주입

우리는 생성자를 주로 했습니다. 필드 객체 선언으로도 했습니다. jbdcTemplate(DataSource)

```
01 @RestController
02 public class DIController {
03
04     MyService myService;
05
```

```
06     @Autowired
07     public DIController(MyService myService) {
08         this.myService = myService;
09     }
10
11     @GetMapping("/di/hello")
12     public String getHello() {
13         return myService.getHello();
14     }
15
16 }
```

new가 없습니다.
@Autowired 를 달아 놓으면 Spring 이 DI(Dependency Injection)

@Service – 앞으로 많이 등장할 예정

```
01 @RestController
02 public class FieldInjectionController {
03
04     @Autowired
05     private MyService myService;
06
07 }
```

@Autowird가 필드에 붙어있습니다. — UserDaoTest에서 썼던 방법
@Autowired ApplicationContext할 때 했었습니다.

Spring AoP

핵심기능(비즈로직)과 부가기능(로깅, 모니터링, 트랜잭션, 캐싱...)으로 나누는 디자인 방식
Spring은 비즈니스 로직에 집중할 수 있도록 그래서 생산성을 높일 수 있는 방향으로 진화 하고 있습니다.

TDD가 가능하도록 하는 것
TDD할 때 비즈니스 로직만 테스트 하고 부가기능은 이미 검증된 구현체를 쓰도록 하여
부가기능은 테스트에서 가급적이면 배제할 수 있으면 배재함.

Spring Boot를 썼을때 무엇이 좋은가? 당연한거 아닌가?

```
01 <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
02     <property name="driverClass" value="${db.driver}" />
03     <property name="jdbcUrl" value="${db.url}" />
04     <property name="user" value="${db.username}" />
05     <property name="password" value="${db.password}" />
06 </bean>
07
08 <jdbc:initialize-database data-source="dataSource">
09     <jdbc:script location="classpath:config/schema.sql" />
10     <jdbc:script location="classpath:config/data.sql" />
11 </jdbc:initialize-database>
12
13 <bean class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
id="entityManagerFactory">
14     <property name="persistenceUnitName" value="hsql_pu" />
15     <property name="dataSource" ref="dataSource" />
16 </bean>
17
18 <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
19     <property name="entityManagerFactory" ref="entityManagerFactory" />
20     <property name="dataSource" ref="dataSource" />
21 </bean>
```

Spring Boot을 안쓰면 이런 설정을 몇 바닥씩 해주고 테스트까지 다 해야 합니다.

SpringBoot Starter

1.2.1 의존성 관리

스프링 프레임워크에서는 개발에 필요한 각 모듈의 의존성을 직접 설정해야 했습니다. 또 호환되는 버전을 명시해야 정상 동작합니다. 애플리케이션에서 사용하는 스프링 프레임워크나 라이브러리의 버전을 올리는 상황에서는 연관된 다른 라이브러리의 버전까지도 고려해야 합니다.

하지만 스프링 부트에서는 이 같은 불편함을 해소하기 위해 'spring-boot-starter'라는 의존성을 제공합니다. spring-boot-starter의 의존성은 여러 종류가 있고, 각 라이브러리의 기능과 관련해서 자주

spring-boot-starter-jdbc
spring-boot-starter-test

각 dependency마다 충돌, 버젼 이슈 spring-boot-starter 는 이런 문제들을 대부분 정리 해줍니다.

설정하는데 애로사항
SpringBoot이 나오면서 개발환경 세팅 하는 시간이 아주 많이 줄었습니다. (공기같다)
Git이슈

1.1.3 스프링 부트에 대한 몇 가지 오해

최근 몇 년 동안 Spring boot의 유용한 기능 덕분에 Spring Boot에 대한 많은 이야기가 있었습니다. 이 책을 읽기 전에 들었거나 다른 책을 읽으면서 몇 가지 오해가 생겼을 수도 있는데 이 부분에 대해서 정리하고 넘어가겠습니다.

첫째, Spring Boot는 애플리케이션 서버(WAS)가 아닙니다. 이러한 오해는 기존의 방법처럼 Java 응용 프로그램 서버에 배포하지 않고도 command line을 통해 웹 서버에서 응용 프로그램을 실행할 수 있는 자체 실행(self-execute) JAR 파일로 생성할 수 있다는 사실에서 비롯됩니다. Spring Boot에서는 내부의 Tomcat, Jetty, Undertow와 같은 servlet container를 통해 웹에서 프로그램을 실행할 수 있습니다. 하지만 이것은 servlet container가 제공하는 기능이지, Spring Boot의 자체적인 기능은 아닙니다.

SpringBoot != WAS(Web Application Server – Tomcat, Jetty)

WAS를 포함 하고는 있지만 WAS는 아님

Spring Boot 빌드

Spring Initializr

MSA Microservice Architecture

2.1 서버 간 통신

어떤 포털 사이트를 하나의 서비스 단위로 개발한다고 가정해 보겠습니다. 즉, 블로그, 카페, 메일 등의 기능들을 하나의 애플리케이션에 통합했다는 뜻입니다. 서비스를 이렇게 구성한다면 서버를 업데이트하거나 애플리케이션을 유지보수할 때마다 '사이트 작업 중입니다'라는 팻말을 걸고 작업을 해야 합니다. 그만큼 개발에 보수적인 입장을 취할 수밖에 없고, 서비스 자체의 규모도 커지기 때문에 서비스를 구동하는 데 걸리는 시간도 길어집니다.

이 같은 문제를 해결하기 위해 나온 것이 마이크로서비스 아키텍처(MSA; Microservice Architecture)입니다. 마이크로서비스 아키텍처는 단어 그대로 서비스 규모를 작게 나누어 구성한 아

Monolithic User인증, 쇼핑몰 주문기능, 메세지 전송기능 …
MSA 인증따로 쇼핑몰 주문따로, 배송따로 결제 따로 ,메세지 전송 따로
서버간의 통신은 API로 이루어진다

spring-boot-starter-web = Spring MVC

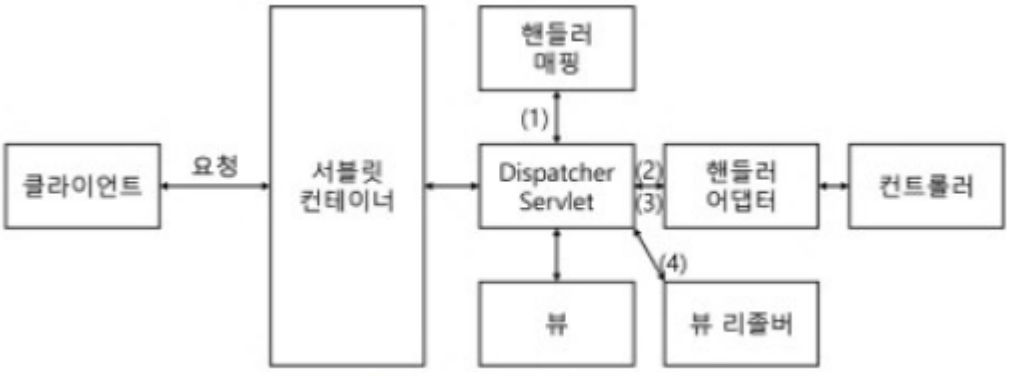


그림 2.2 스프링 부트의 동작 구조

MVC라는 말을 잘 안씁니다.
View 가 많이 Frontend(React, Vue)

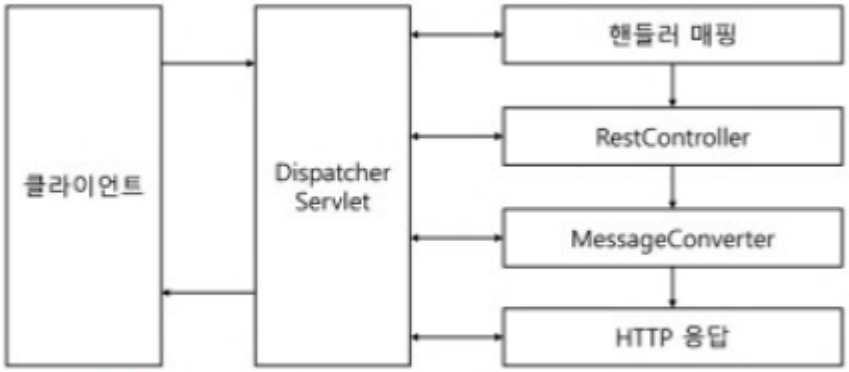


그림 2.4 @RestController를 사용하는 DispatcherServlet의 동작 방식

SpringBoot로는 주로 API를 많이 만들기 때문에 MC구조라고 할 수 있습니다.

2.3 레이어드 아키텍처

레이어드 아키텍처(Layered Architecture)란 애플리케이션의 컴포넌트를 유사 관심사를 기준으로 레이어로 묶어 수평적으로 구성한 구조를 의미합니다. 레이어드 아키텍처는 여러 방면에서 쓰이는 개념이며, 어떻게 설계하느냐에 따라 용어와 계층의 수가 달라집니다.

일반적으로 레이어드 아키텍처라 하면 3계층 또는 4계층 구성을 의미합니다. 이 차이는 인프라(데이터베이스) 레이어의 추가 여부로 결정됩니다. 이번 장에서는 그림 2.5와 같이 3계층으로 이뤄진 레이어드 아키텍처를 살펴봅니다.

이 책은 레이어드 아키텍처 기반으로 쓰여졌습니다.

Hexagonal Architecture
Port Adapter Architecture
DDD 등

영속성?

JPA(Java Persistence API)
영속성 - DB에 저장된다.

데이터 접근 계층

- 상황에 따라 영속(Persistence) 계층이라고도 합니다.
- 데이터베이스에 접근해야 하는 작업을 수행합니다.
- 그림 2.6에서는 DAO라는 컴포넌트를 표현했지만 Spring Data JPA에서는 DAO 역할을 리포지토리가 수행하기 때문에 리포지토리로 대체할 수 있습니다.

REST API

2.5 REST API

REST API는 대중적으로 가장 많이 사용되는 애플리케이션 인터페이스입니다. 이 인터페이스를 통해 클라이언트는 서버에 접근하고 자원을 조작할 수 있습니다. 이번 절에서는 REST의 형식 및 규칙을 알아보겠습니다.

2.5.4 REST의 URI 설계 규칙

REST API를 설계하는 데는 다음과 같은 몇 가지 규칙이 있습니다.

URL 규칙

URI의 마지막에는 '/'를 포함하지 않습니다.

- 옳은 예) http://localhost.com/product
- 잘못된 예) http://localhost.com/product/

언더바(_)는 사용하지 않습니다. 대신 하이픈(-)을 이용합니다.

- 하이픈은 리소스의 이름이 길어지면 사용합니다.
- 옳은 예) http://localhost.com/provider-company-name
- 잘못된 예) http://localhost.com/provider_company_name

URL에는 행위(동사)가 아닌 결과(명사)를 포함합니다.

- 옳은 예) http://localhost.com/product/123
- 잘못된 예) http://localhost.com/delete-product/123
- 행위는 HTTP 메서드로 표현할 수 있어야 합니다.

URI는 소문자로 작성해야 합니다.

- URI 리소스 경로에는 대문자 사용을 피하는 것이 좋습니다.
- 일부 웹 서버의 운영체제는 리소스 경로 부분의 대소문자를 다른 문자로 인식하기 때문입니다. 이러한 이유로 RFC 3986은 URI 문법 형식을 정의하고 있는데, 호스트의 구성요소를 제외하고 URI의 대소문자를 구분해서 정의하고 있습니다.

파일의 확장자는 URI에 포함하지 않습니다.

- HTTP에서 제공하는 `Accept` 헤더를 사용하는 것이 좋습니다.

권장되는 표현방식이 아닌 표현방식

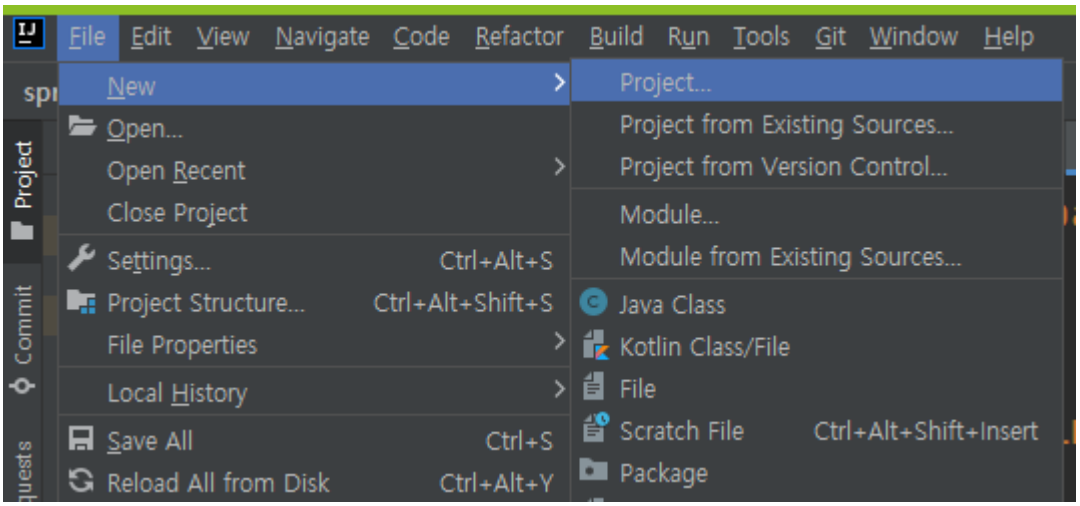
/product/save x
/user/get x
/vendor_company x

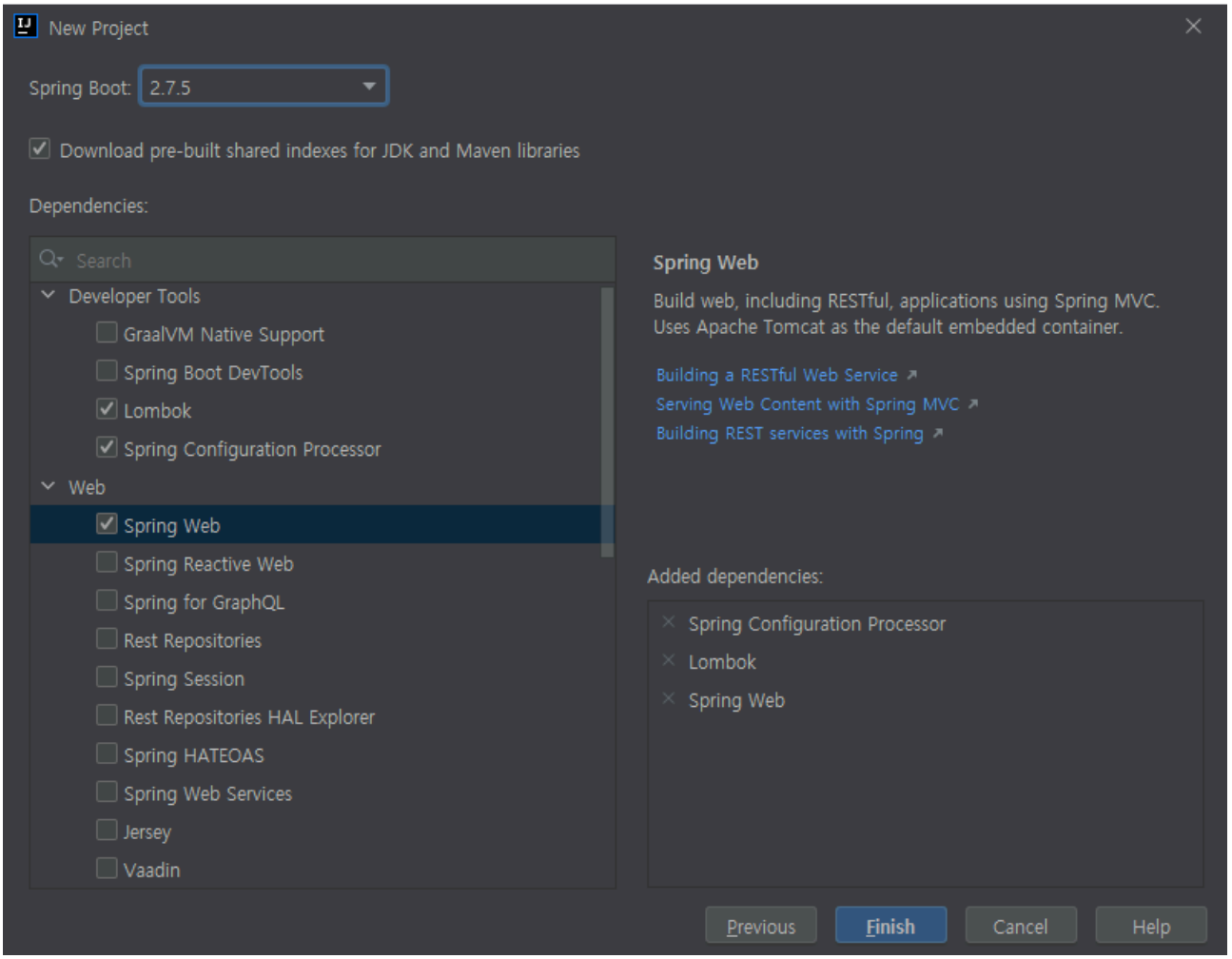
/vendorCompany/save x → 총체적난국

이 책에서 사용한 개발 환경

- Windows 10 64-bit
- 인텔리제이(IntelliJ) Ultimate
- JDK 11(1.8 버전 이상)
- 스프링 부트 2.5.6~2.5.8
- MariaDB 10.6.5
- 메이븐(Maven)

SpringBoot빌드 http://start.spring.io/ or IntelliJ Ultimit





Community Edition에서 프로젝트 빌드

Spring Initializr

spring initializr
Project
Gradle Project
Maven Project
Language
Java
Kotlin
Groovy
Spring Boot
3.0.0 (SNAPSHOT)
3.0.0 (RC1)
2.7.6 (SNAPSHOT)
2.7.5
2.6.14 (SNAPSHOT)
2.6.13
Project Metadata
Group
com.springboot
Artifact
hello
Name
hello
Description
Demo project for Spring Boot
Package name
com.springboot.hello
Packaging
Jar
War
Java
19
17
11
8
Dependencies
ADD ... CTRL + B
Lombok
DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.
Spring Configuration Processor
DEVELOPER TOOLS
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...

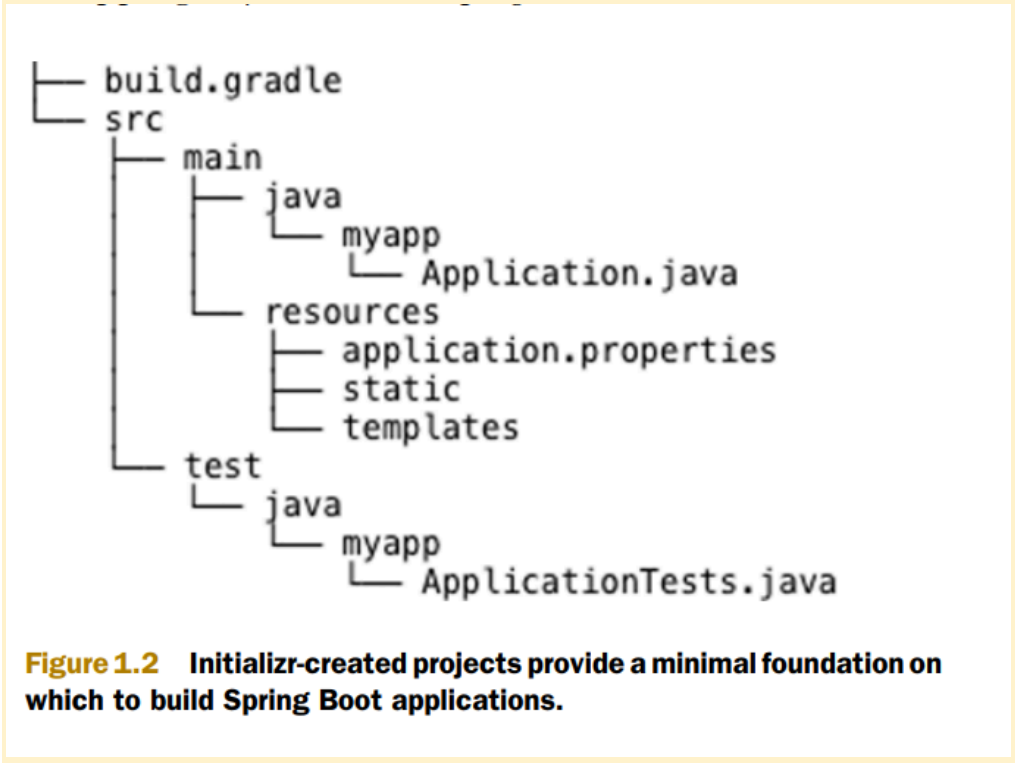
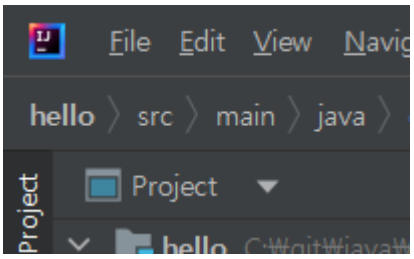


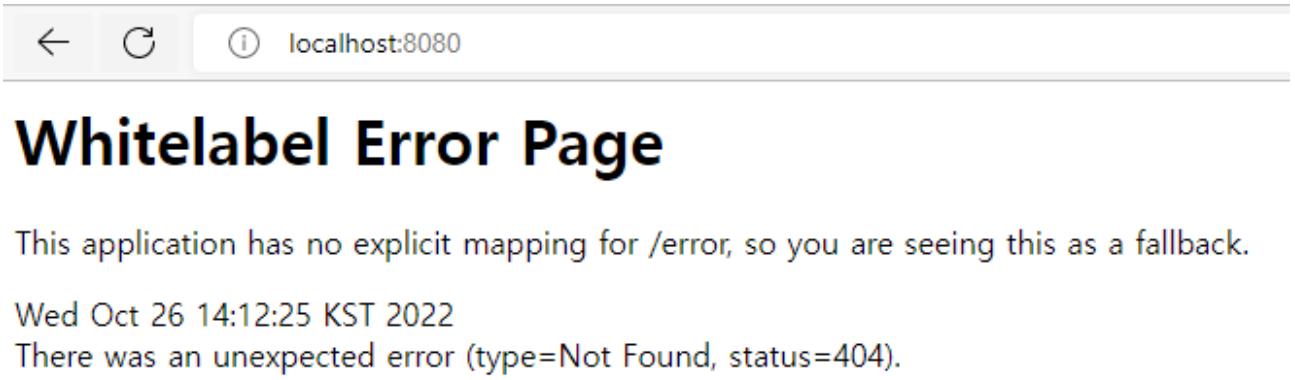
Figure 1.2 **Initializr-created projects provide a minimal foundation on which to build Spring Boot applications.**

위와 같은 구조로 빌드가 되어 있습니다.

실행 하기

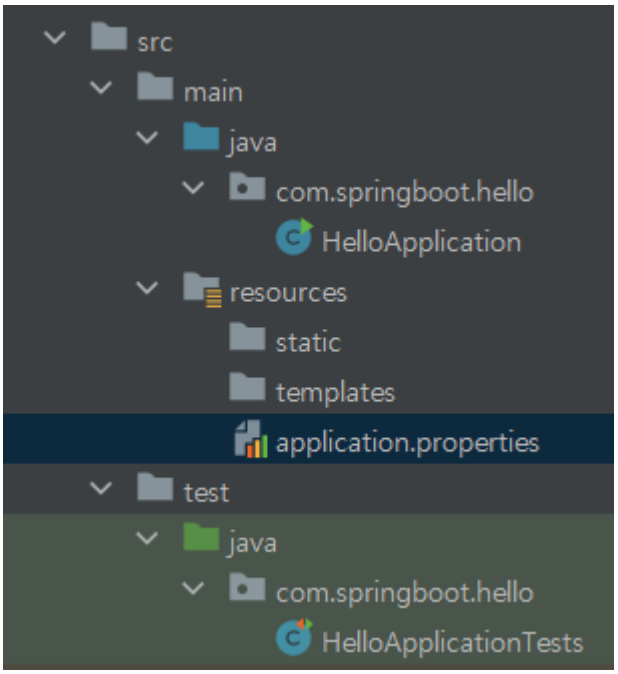
```
Tomcat started on port(s): 8080 (http) with context path ''
Started HelloApplication in 2.97 seconds (JVM running for 3.518)
```





웹브라우저에서 localhost:8080/ 로 접속 해보세요.

Spring Boot프로젝트 기본 구조



Maven이란?

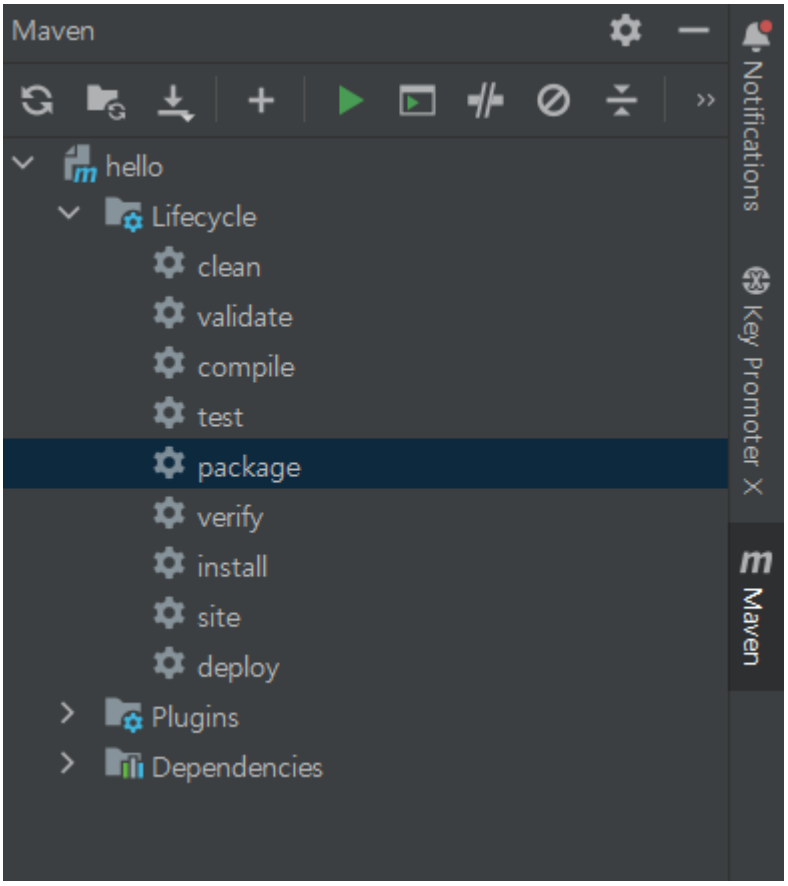
Gradle과 비슷한 도구 Gradle보다 먼저 나왔음
라이브러리(Dependency) 관리 + 빌드 관리 도구

```
<groupId>com.springboot</groupId>
<artifactId>hello</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>hello</name>
<description>hello</description>
```

4.2.2 메이븐

아파치 메이븐은 자바 기반의 프로젝트를 빌드하고 관리하는 데 사용하는 도구입니다. 초창기 자바 프로젝트의 대표적 관리 도구였던 Ant를 대체하기 위해 개발됐습니다. 메이븐의 가장 큰 특징은 `pom.xml` 파일에 필요한 라이브러리를 추가하면 해당 라이브러리에 필요한 라이브러리까지 함께 내려받아 관리한다는 점입니다. 메이븐의 대표 기능은 다음과 같습니다.

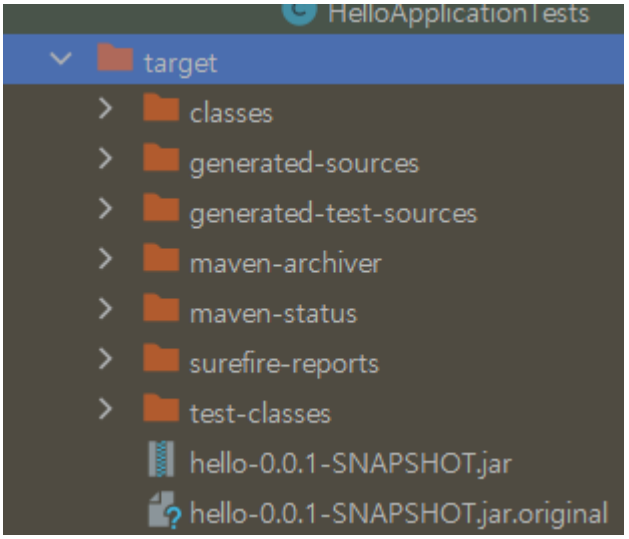
- 프로젝트 관리: 프로젝트 버전과 아티팩트를 관리합니다.
- 빌드 및 패키징: 의존성을 관리하고 설정된 패키지 형식으로 빌드를 수행합니다.
- 테스트: 빌드를 수행하기 전에 단위 테스트를 통해 작성된 애플리케이션 코드의 정상 동작 여부를 확인합니다.
- 배포: 빌드가 완료된 패키지를 원격 저장소에 배포합니다.



오른쪽 Maven - package

package를 하면

target/ 아래에 .jar파일이 만들어 집니다.



.jar파일은 우리가 개발한 소스코드가 실행 가능한 형태로 패키징 된 파일 입니다.

이 파일이 있으면 우리가 개발한 백엔드 어플리케이션을 jvm이 설치된 환경에서 실행 할 수 있습니다.

.jar을 만들려면 Maven:package를 실행 해야 합니다. package를 하기 위해 아래 Life Cycle을 거칩니다.

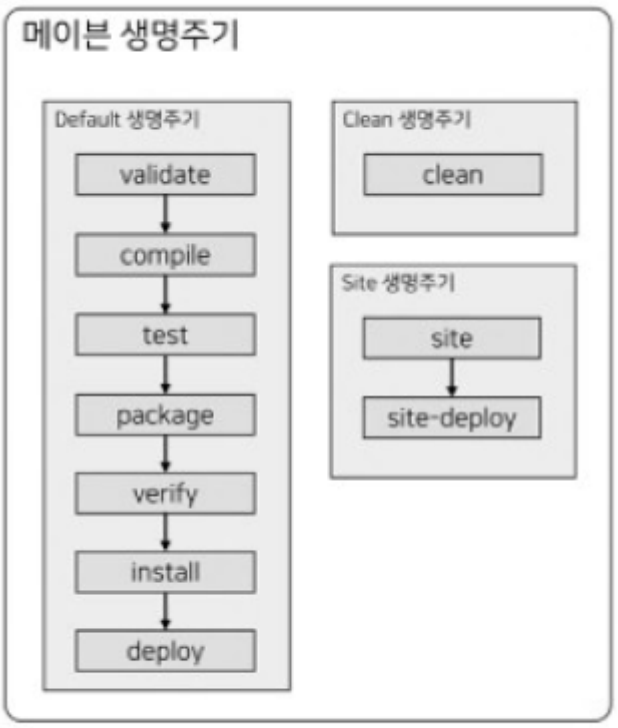


그림 4.12 메이븐 주요 생명주기

package전에 test가 있습니다.
우리가 만든 테스트 코드는 이 단계에서 모든 테스트 코드가 실행 됩니다.

이 단계에서 side effect나 미처 생각하지 못했던 버그들을 발견 할 수 있습니다.

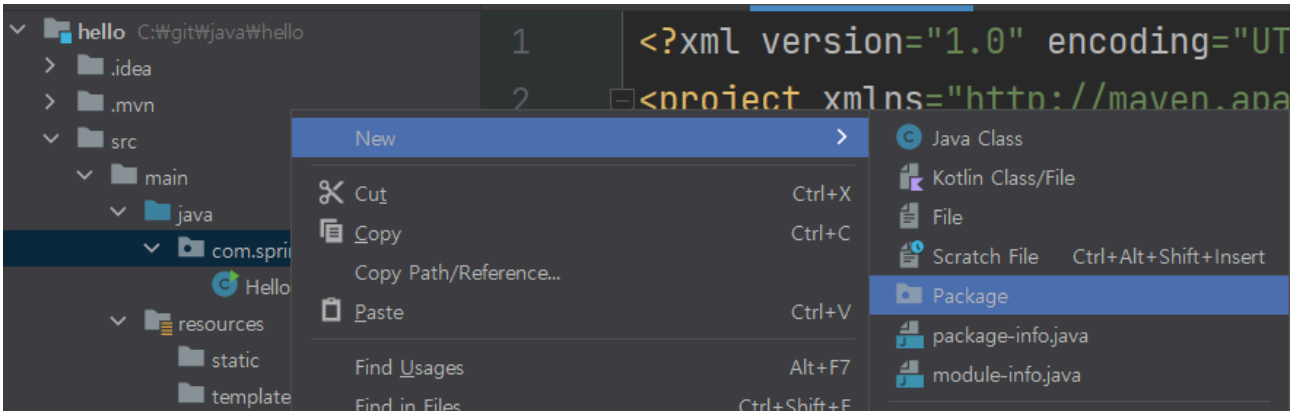
gradle도 유사한 Life Cycle을 거칩니다.

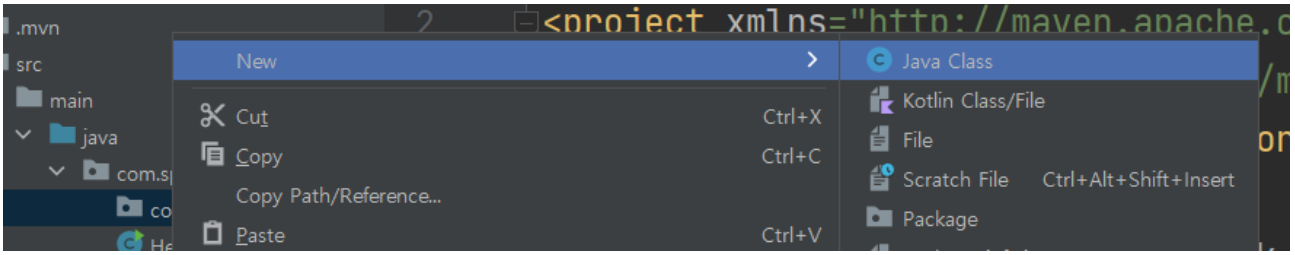
.jar로 실행
Terminal켜고
$ cd target
$ java -jar hello-0.0.1-SNAPSHOT.jar

지금까지 한줄 요약

SpringBoot빌드해서 .jar을 만든 후 실행시켜보세요.

SpringBoot Controller로 Hello출력하기





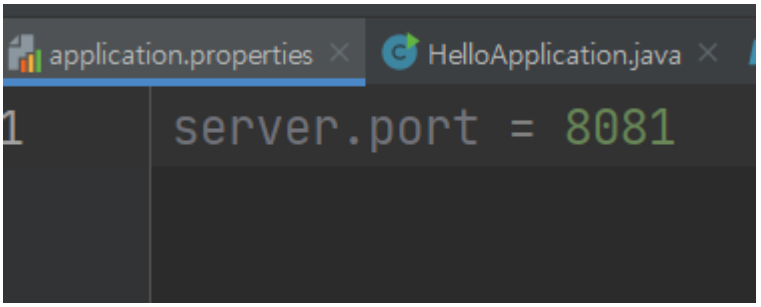
@RestController, @RequestMapping(“path”)

```
@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String hello(){
        return "Hello World";
    }

}
```

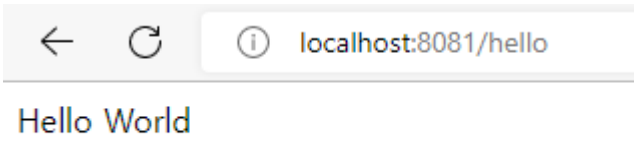
@RestController - dispatcherServlet이 매핑해줄 Controller를 등록

@RequestMapping - controller가 할당 된 후 어떤 method를 실행할지 연결해주는 어노테이션 입니다.

포트 변경 방법



실행 완료



실습

1. SpringBoot 빌드
2. package한 후 .jar실행
3. HelloController추가 후 실행
4. port바꿔보고 실행

17분 3:10분까지

Talend API Tester

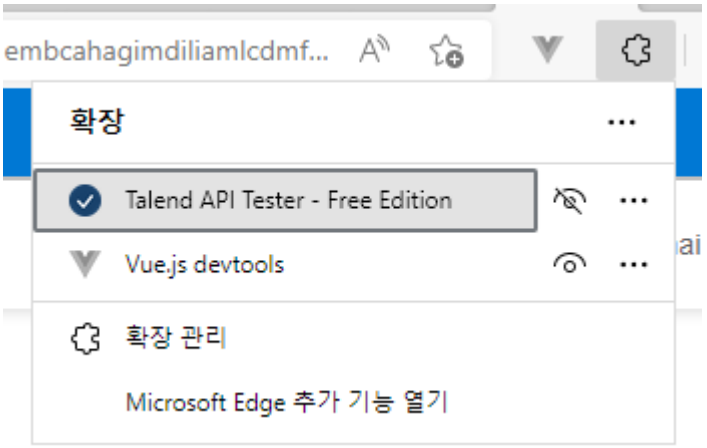
or PostMan(유료)

API를 만들고 테스트 할 수 있는 Tool.
특히 Webbrowser에서는 Post테스트가 안되기 때문에 API Tester 툴을 씁니다.

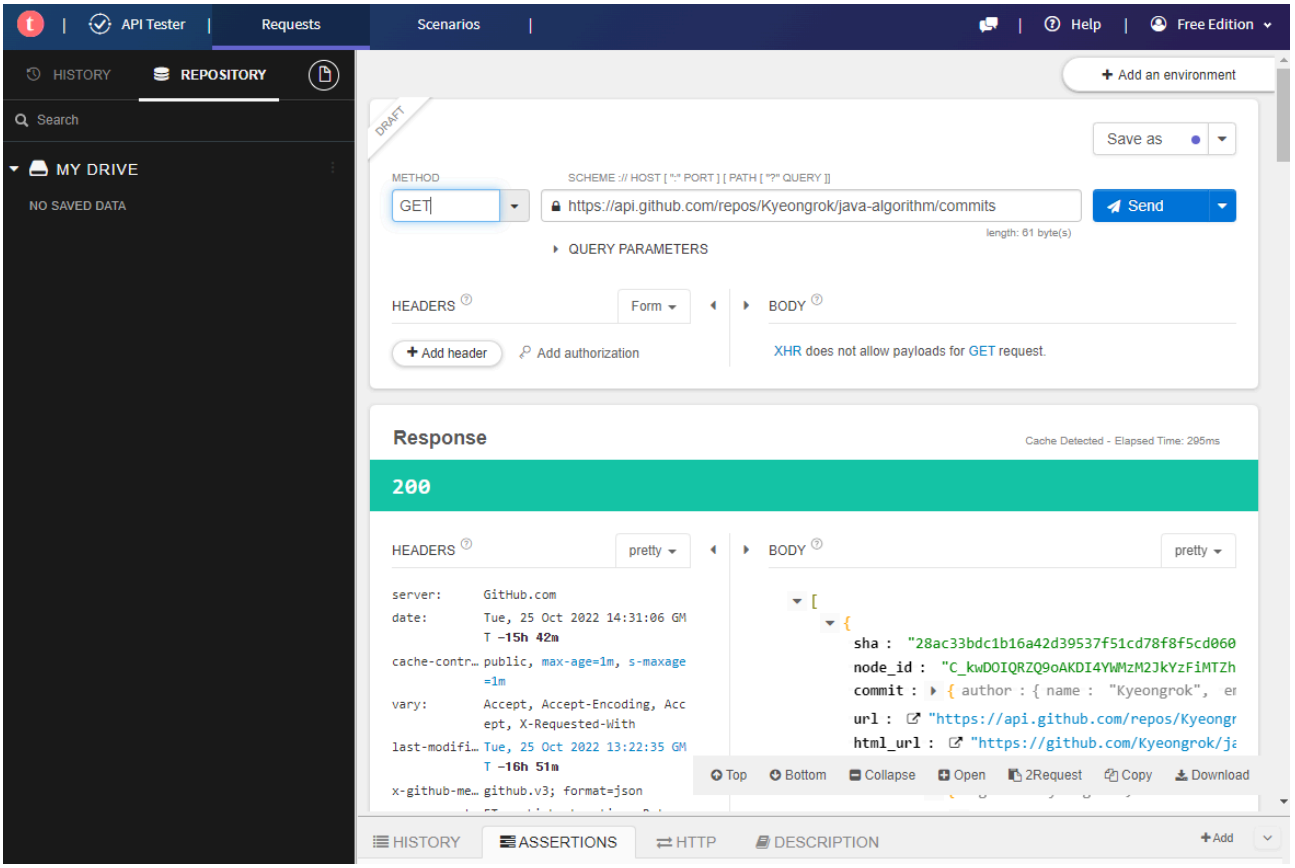
https://chrome.google.com › talend-api-tester-free-ed

Talend API Tester - Free Edition

Aug 22, 2022 — **Talend API Tester** makes it easy to invoke, discover and test HTTP and REST APIs. **Talend API Tester** - Free Edition's main functions include: 1.



실행한 화면



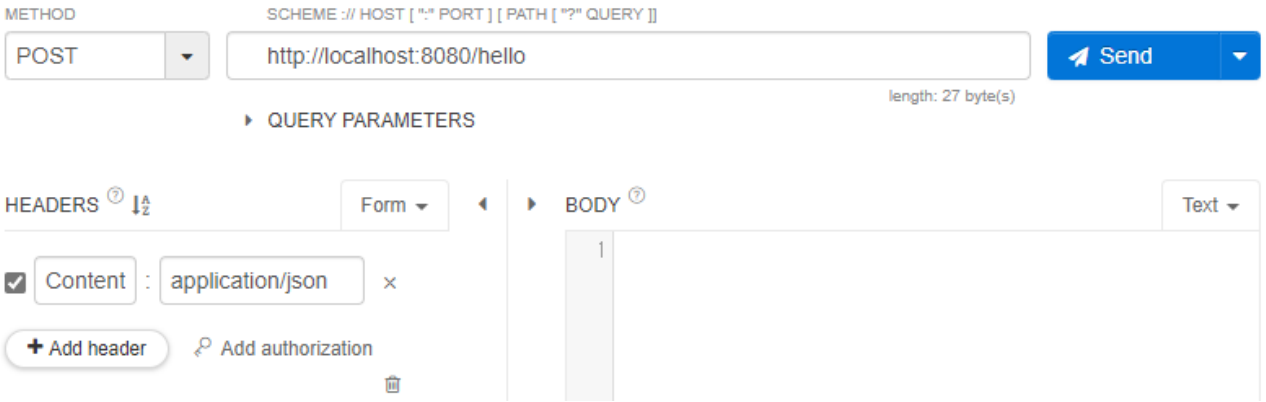
HTTP 메서드

Get, Post, Put, Delete

Talend API Tester는 HTTP 통신을 테스트하는 프로그램입니다. GET, POST, PUT, DELETE 등의 다양한 HTTP 메서드를 설정하고 쿼리(query)와 파라미터(parameter)를 담아 요청을 보낼 수 있습니다. 크롬 브라우저에서 Talend API Tester를 실행하면 그림 4.20과 같은 화면이 표시됩니다.

그냥 RequestMapping은 Post, Get, Delete, Put이 다 됩니다.

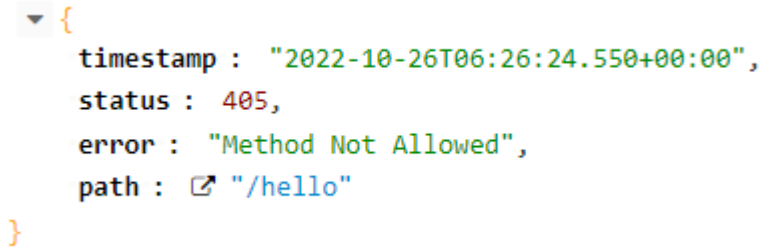
```
@RestController
public class HelloController {
    @RequestMapping("/hello")
    public String hello() {
        return "Hello World";
    }
}
```



특정 Method만 허용

```
@RestController
```

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

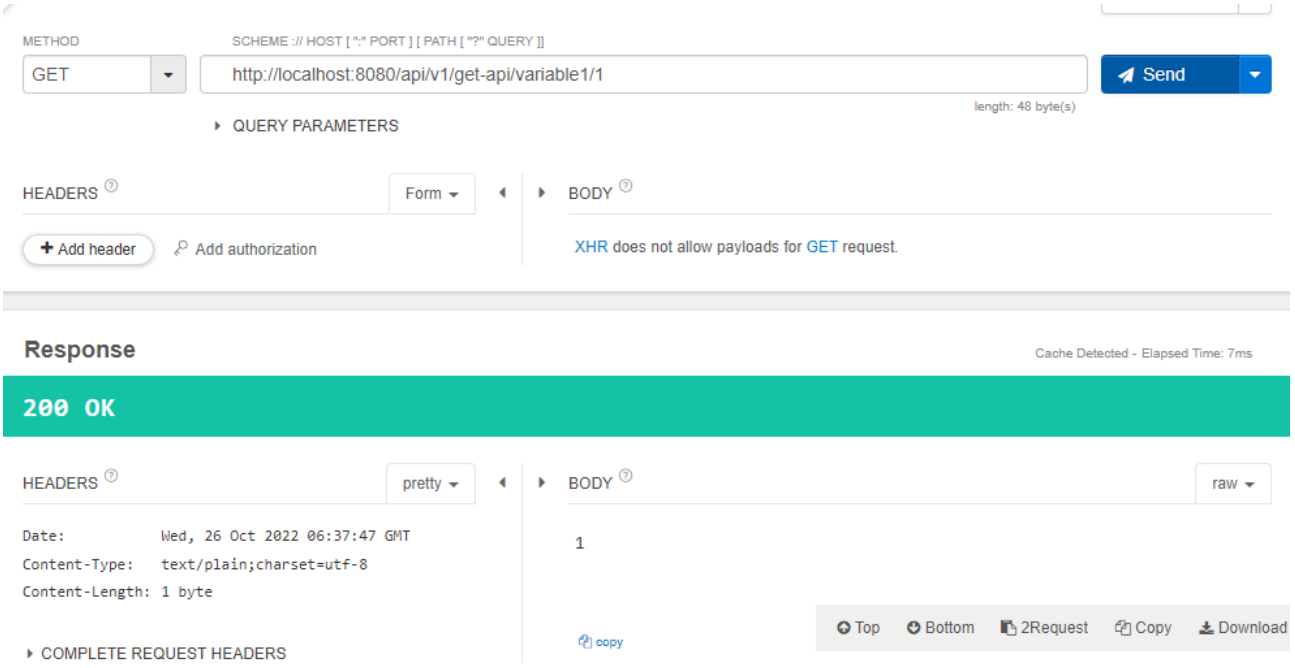


5.2.3 @PathVariable을 활용한 GET 메서드 구현

실무 환경에서는 매개변수를 받지 않는 메서드가 거의 쓰이지 않습니다. 웹 통신의 기본 목적은 데이터를 주고받는 것이기 때문에 대부분 매개변수를 받는 메서드를 작성하게 됩니다. 매개변수를 받을 때 자주 쓰이는 방법 중 하나는 URL 자체에 값을 담아 요청하는 것입니다. 예제 5.4는 URL에 값을 담아 전달되는 요청을 처리하는 방법을 보여줍니다.

userDao.findById(id) → 특정 user

```
@GetMapping(value =
"/variable1/{variable}")
public String getVariable1(@PathVariable
String variable) {
    return variable;
}
```



http://localhost:8080/api/v1/get-api/variable1/1

주소의 경로를 통해 variable을 넘기고 싶을 때 씁니다.
ex)
http://localhost:8080/api/v1/get-api/variable1/<variable>

3:50까지 실습

아기사자반 Method와 Class(50이론 50실습)

반복되는(3번 이상) 로직을 계속 태워야 할 경우 method()로 만드는 것을 고려합니다.

언제 Return을 쓰게 될까?

-

메소드 설계 – hash()를 설계 한다면 무엇을 받고 무엇을 리턴 하는가?
함수에서 클래스로 넘어가는 의사결정은 언제 하게 될까?
Stack에서 stack size, HashTable에서 size
클래스로 넘어갈 때 값 전달 - 디폴트 생성자, 생성자 오버로딩

앞으로 할 내용	SpringBoot로 게시판 API(CRUD) 만들기
6주차	Spring Boot API만들기 Spring Boot in Action (lagout.org)” 아기사자반 Topic : Method와 Return, Return Type 최대인원 17분
7주차	Docker, Spring Boot App Docker 빌드하기, AWS EC2에 배포하기 .jar파일 서버에서 빌드 후 바로 실행
8주차	