Spring Boot

Spring Boot

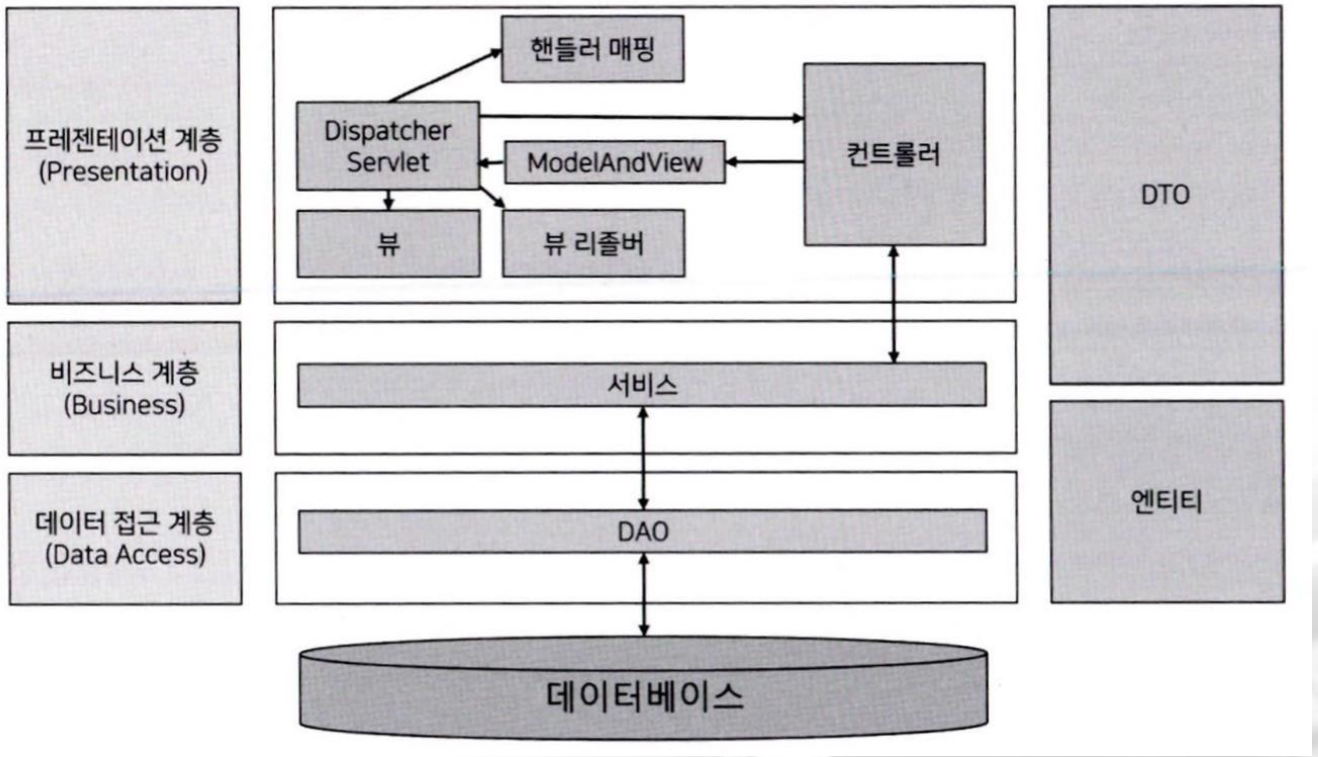
- Spring Boot
  - ✓ Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration - 스프링 부트는 단독 실행되는 상용화 가능한 수준의 스프링 기반 애플리케이션을 쉽게 만들어낼 수 있으며 최소한의 설정으로 스프링 플랫폼과 3rd Party 라이브러리들을 사용할 수 있도록 함

Spring Boot

- Spring Boot
  - ✓ 장점
    - 자동 설정 - 환경 설정을 최소화
    - WAS를 내장해서 독립 실행이 가능한 스프링 애플리케이션 개발이 가능
    - Spring Boot Starter 의존성을 제공하여 Maven/Gradle 의 설정을 간소화 하고 자동화된 설정 제공
    - XML 설정 없이 자바 수준의 설정 방식 제공
    - JAR을 사용하여 자바 옵션 만으로 배포 가능
    - 애플리케이션의 모니터링과 관리를 위한 Spring Actuator 제공
  - ✓ 단점
    - 설정을 자동화하지 않고 별도로 설정하면 버전을 올릴 때 기존 스프링 프레임워크를 사용하는 것과 동일한 불편함
    - 특정 설정을 별도로 하거나 설정 자체를 변경하고 싶다면 내부의 설정 코드를 살펴봐야 하는 불편함

Spring Boot

- Spring Boot
  - ✓ Layerd Architecture



Spring Boot

- Spring Boot
  - ✓ Layerd Architecture
    - 프레젠테이션 계층
      - 유저 인터페이스(UI: User Interface) 계층
      - 클라이언트 와 의 접점
      - 클라이언트로부터 데이터와 함께 요청을 받고 처리 결과를 응답으로 전달하는 역할
    - 비즈니스 계층
      - 서비스(Service) 계층
      - 핵심 비즈니스 로직을 구현하는 영역
      - 트랜잭션 처리나 유효성 검사 등의 작업도 수행
    - 데이터 접근 계층
      - 영속(Persistence) 계층
      - 데이터베이스에 접근해야 하는 작업을 수행
      - DAO라는 컴포넌트로 표현했지만 Spring Data JPA에서는 DAO 역할을 리포지토리가 수행하기 때문에 리포지토리로 대체할 수 있음

Spring Boot

- Spring Boot Starter
  - ✓ 의존 관계를 개발자가 간편하게 설정할 수 있도록 도와주는 특정 목적을 달성하기 위한 의존성 그룹
  - ✓ 스프링과 JPA가 필요하다면 pom.xml 파일이나 build.gradle 파일에 spring-boot-starter-data-jpa 만 추가하면 됨
  - ✓ 명명 규칙
    - spring-boot-starter-*: *에 해당 stater 이름을 명시
    - 스프링에서 웹 관련 프로젝트를 진행하는 경우: spring-boot-starter-web

Spring Boot

- Spring Boot Starter
  - ✓ 내부의 의존성 확인 방법
    - https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter

Table 13.1. Spring Boot application starters

Name	Description	Pom
spring-boot-starter	Core starter, including auto-configuration support, logging and YAML	Pom
spring-boot-starter-activemq	Starter for JMS messaging using Apache ActiveMQ	Pom
spring-boot-starter-amqp	Starter for using Spring AMQP and Rabbit MQ	Pom
spring-boot-starter-aop	Starter for aspect-oriented programming with Spring AOP and AspectJ	Pom
spring-boot-starter-artemis	Starter for JMS messaging using Apache Artemis	Pom
spring-boot-starter-batch	Starter for using Spring Batch	Pom
spring-boot-starter-cache	Starter for using Spring Framework's caching support	Pom
spring-boot-starter-cloud-connectors	Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku	Pom
spring-boot-starter-data-cassandra	Starter for using Cassandra distributed database and Spring Data Cassandra	Pom
spring-boot-starter-data-cassandra-reactive	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive	Pom
spring-boot-starter-data-couchbase	Starter for using Couchbase document-oriented database and Spring Data Couchbase	Pom

Spring Boot

- Spring Boot Starter
  - ✓ 내부의 의존성 확인 방법
    - pom.xml

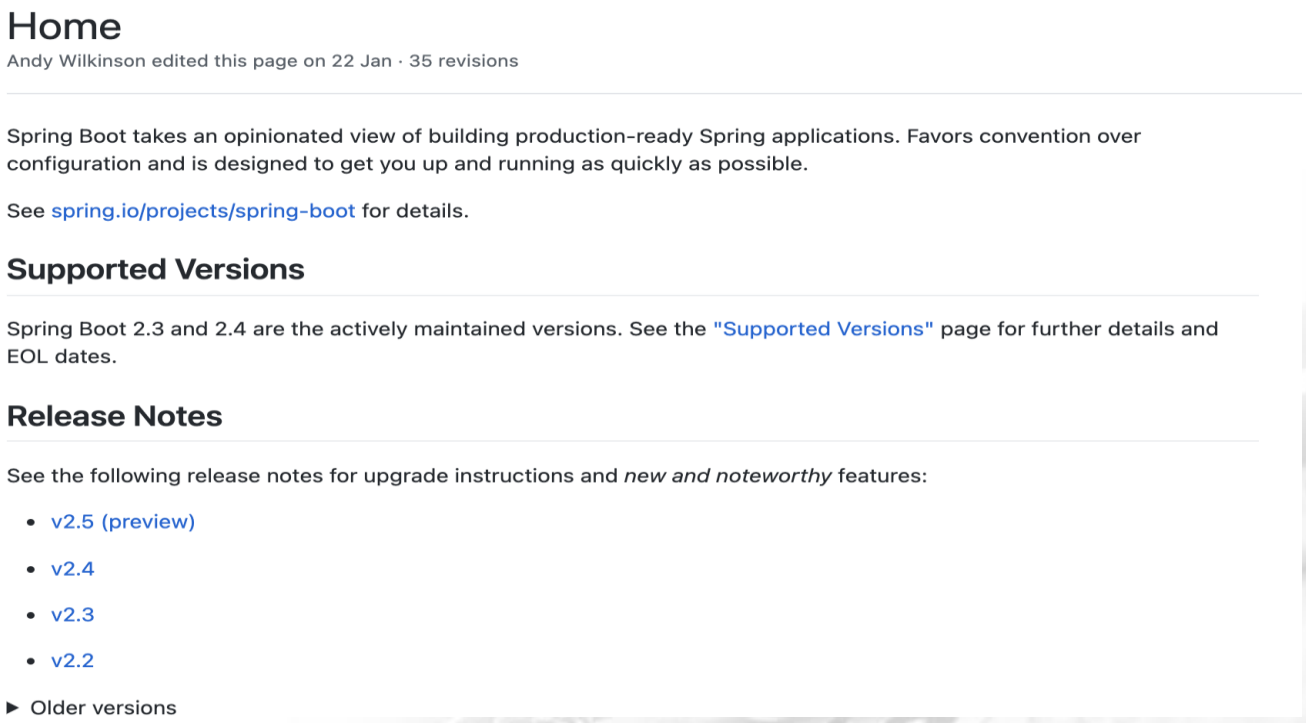
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot</artifactId> ❶
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-autoconfigure</artifactId> ❷
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-logging</artifactId> ❸
    </dependency>
    <dependency>
        <groupId>javax.annotation</groupId>
        <artifactId>javax.annotation-api</artifactId> ❹
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId> ❺
    </dependency>
    <dependency>
        <groupId>org.yaml</groupId>
        <artifactId>snakeyaml</artifactId> ❻
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Spring Boot

- Spring Boot Starter
  - ✓ 기본적으로 제공하는 의존성
    - spring-boot: 스프링 부트에서 기본 제공하는 의존성
    - spring-boot-autoconfigure: 스프링 부트의 자동 환경 구성에 필요한 의존성
    - spring-boot-starter-logging: 각종 로그를 사용하는데 필요한 의존성
    - javax.annotation-api: 소프트웨어의 결함을 탐지하는 어노테이션을 지원하는 의존성
    - spring-core: 스프링 코어를 사용하는 데 필요한 의존성
    - snakeyaml: yaml(사람이 쉽게 읽을 수 있는 데이터 직렬화 양식)을 사용하는데 필요한 의존성

Spring Boot

- Spring Boot Starter
  - ✓ 스프링 부트 도큐먼트: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
  - ✓ 스프링 부트 버전에 따라 달라진 점 확인: https://github.com/spring-projects/spring-boot/wiki



Home

Andy Wilkinson edited this page on 22 Jan · 35 revisions

Spring Boot takes an opinionated view of building production-ready Spring applications. Favors convention over configuration and is designed to get you up and running as quickly as possible.

See spring.io/projects/spring-boot for details.

Supported Versions

Spring Boot 2.3 and 2.4 are the actively maintained versions. See the "Supported Versions" page for further details and EOL dates.

Release Notes

See the following release notes for upgrade instructions and *new and noteworthy* features:

- v2.5 (preview)
- v2.4
- v2.3
- v2.2

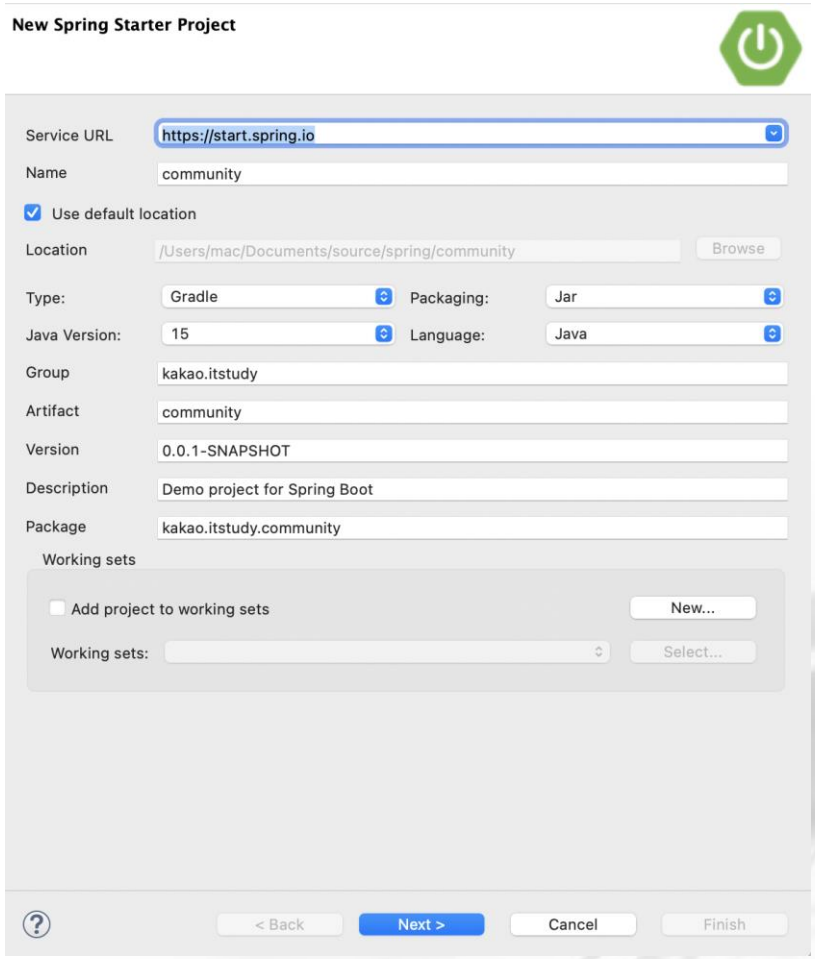
▶ Older versions

Spring Boot

- 개발 환경 설정
  - ✓ JDK
    - https://www.oracle.com/kr/java/technologies/javase-downloads.html 에 접속해서 8 버전 이상 다운로드 받아서 설치 – Spring Boot 2.x는 11이상 이어야 하고 Spring Boot 3.x는 17 이상 설치
  - ✓ IDE
    - https://www.jetbrains.com/idea/download/ 에 접속해서 IntelliJ 를 다운로드 받아서 설치
    - spring.io 에서 STS 다운로드

프로젝트 생성 및 실행

- STS
  - ✓ [File] – [New] – [Spring Starter Project]

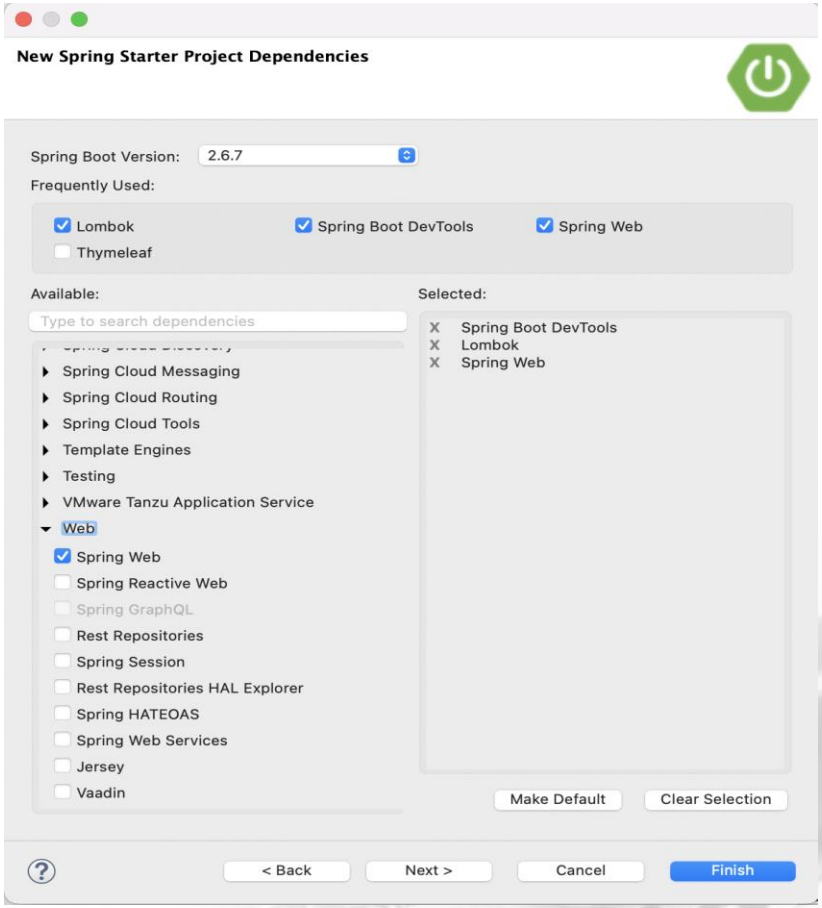


프로젝트 생성 및 실행

- STS
  - ✓ 옵션 설정
    - Type 콤보 박스는 Build 방법을 선택
    - Packaging 콤보 박스는 원하는 애플리케이션 형태를 선택
    - Java Version 콤보 박스는 자바 버전을 선택
    - Language 콤보 박스는 프로그래밍 언어를 선택
    - Group 은 패키지의 상위 경로를 적는데 보통 도메인 주소의 역순으로 작성하는데 예를 들어 도메인이 company.com 일 경우 패키지 상위 경로는 com.company
    - Artifact 에는 프로젝트 약칭을 적는데 라이브러리화 됐을 때 jar에 들어가는 이름
    - Version 은 버전 값
    - Description 은 프로젝트 설명
    - Package 에는 패키지 경로를 적는데 패키지 상위 경로 뒤에 프로젝트명을 추가하는 형태로 작성

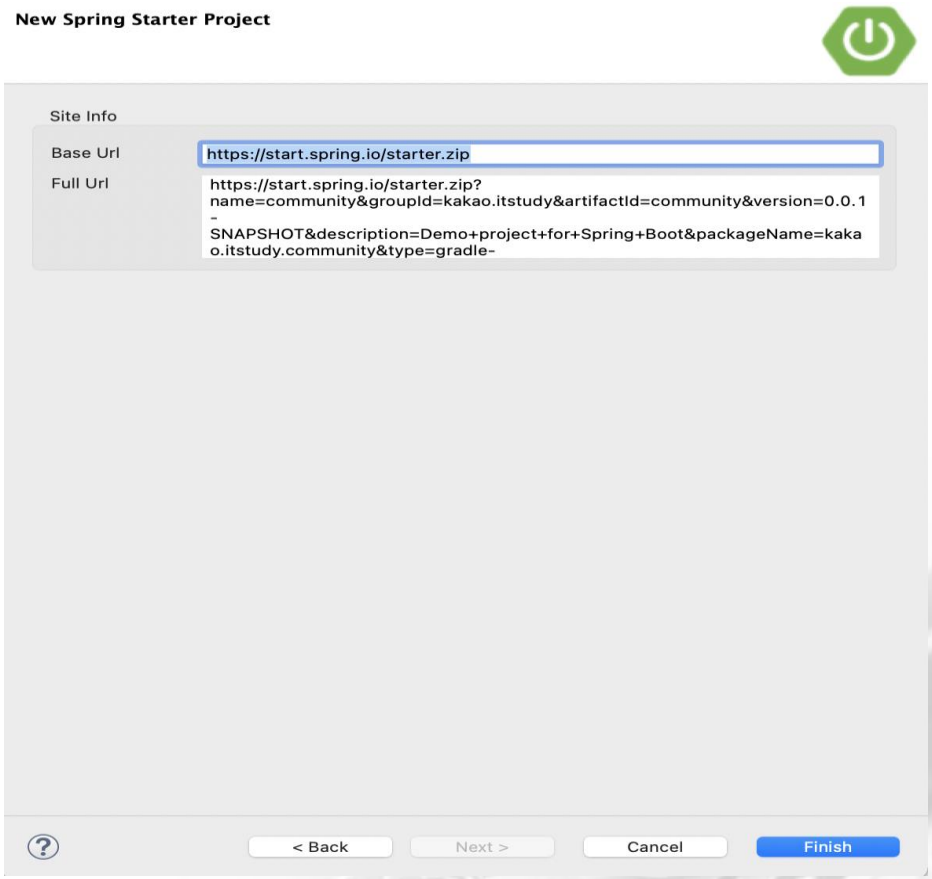
프로젝트 생성 및 실행

- STS
  - ✓ 의존성 설정: Spring Boot DevTools, Lombok, Spring Web



프로젝트 생성 및 실행

- STS
  - ✓ Url 설정



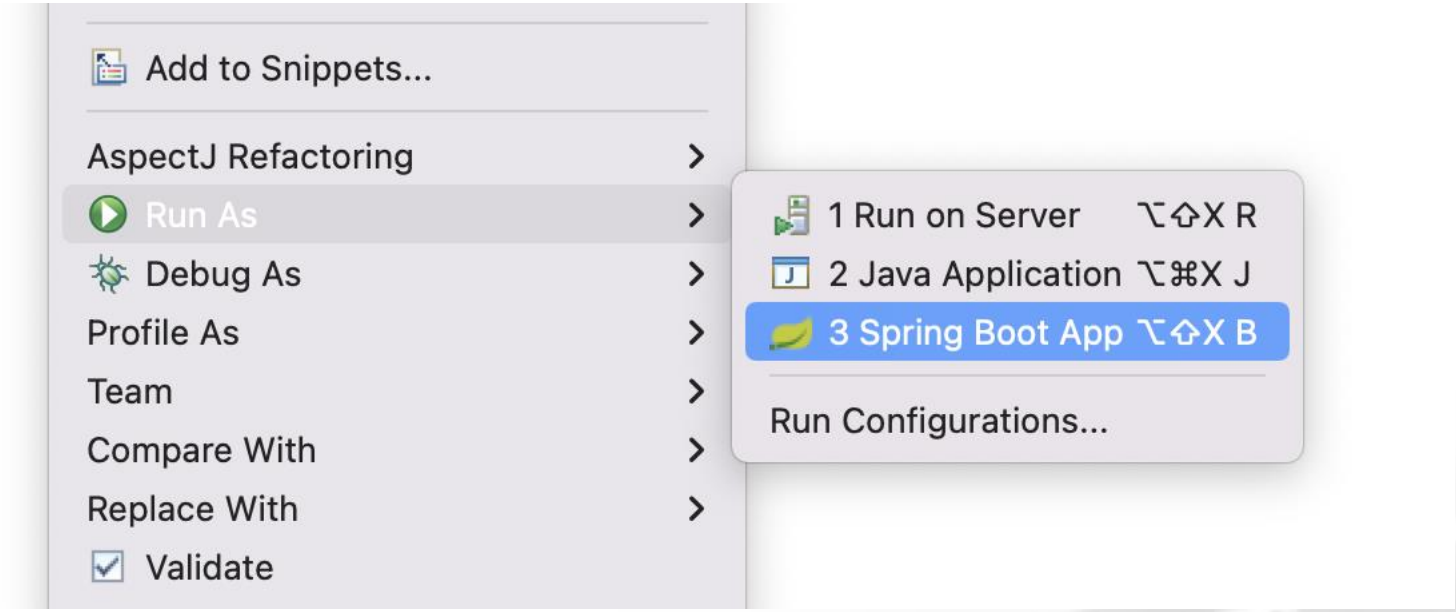
프로젝트 생성 및 실행

- STS
  - ✓ 기본 패키지에 Controller 클래스를 생성하고 작성 - CommonController

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CommonController {

    @GetMapping("/")
    public String home(){
            return "HelloSpringBoot";
    }

}
```

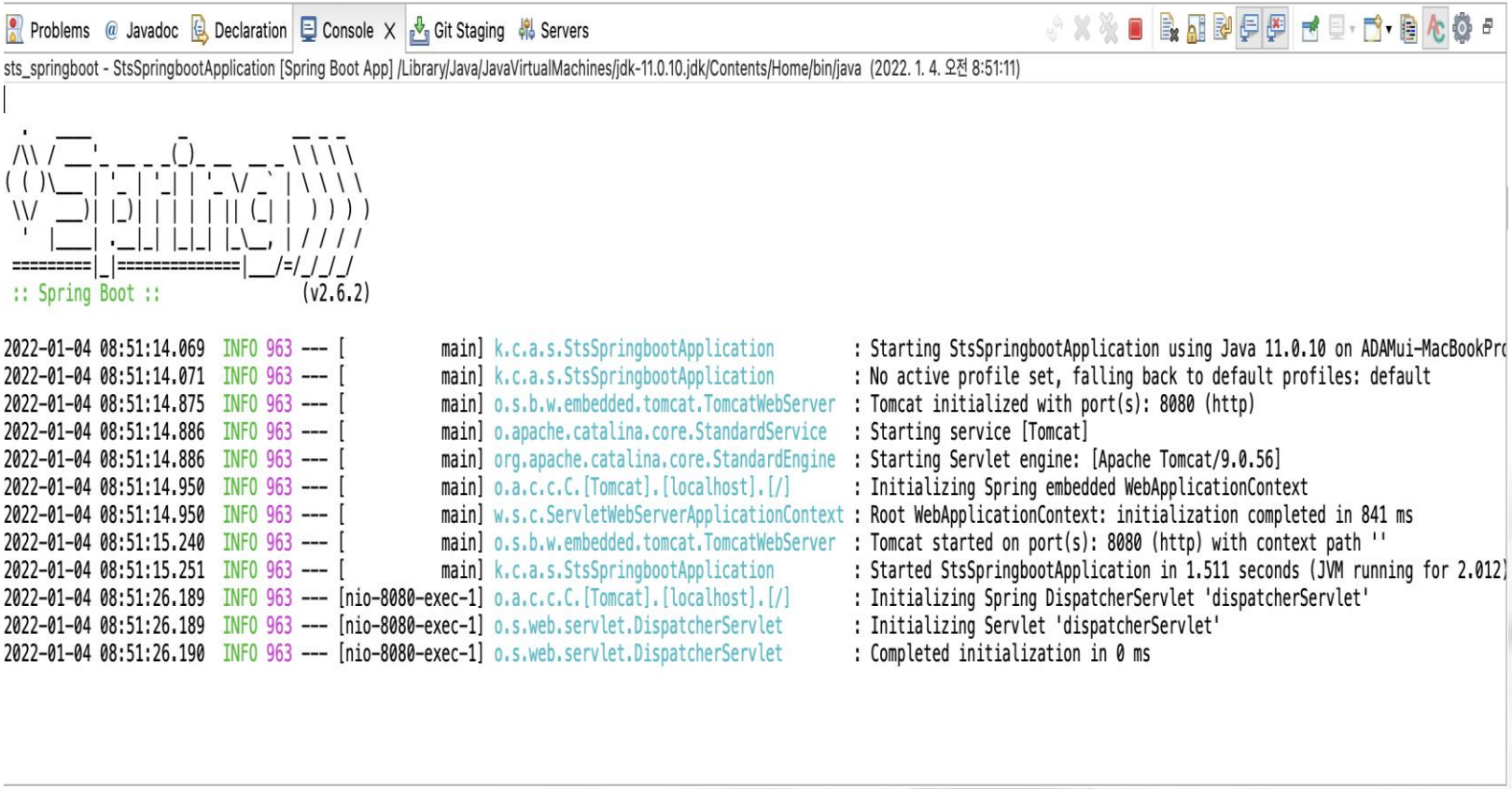
프로젝트 생성 및 실행

- STS
  - ✓ SpringbootApplication.java 파일을 선택한 후 마우스 오른쪽을 클릭 한 후 실행



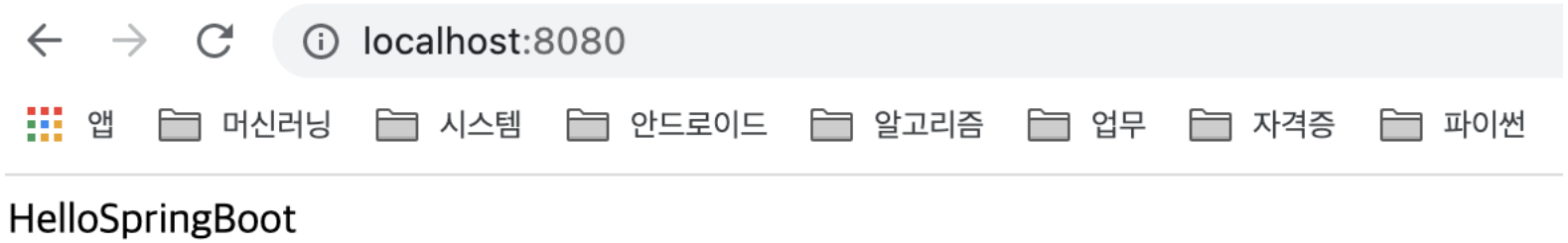
프로젝트 생성 및 실행

- STS
  - ✓ 마우스 오른쪽을 클릭 한 후 실행



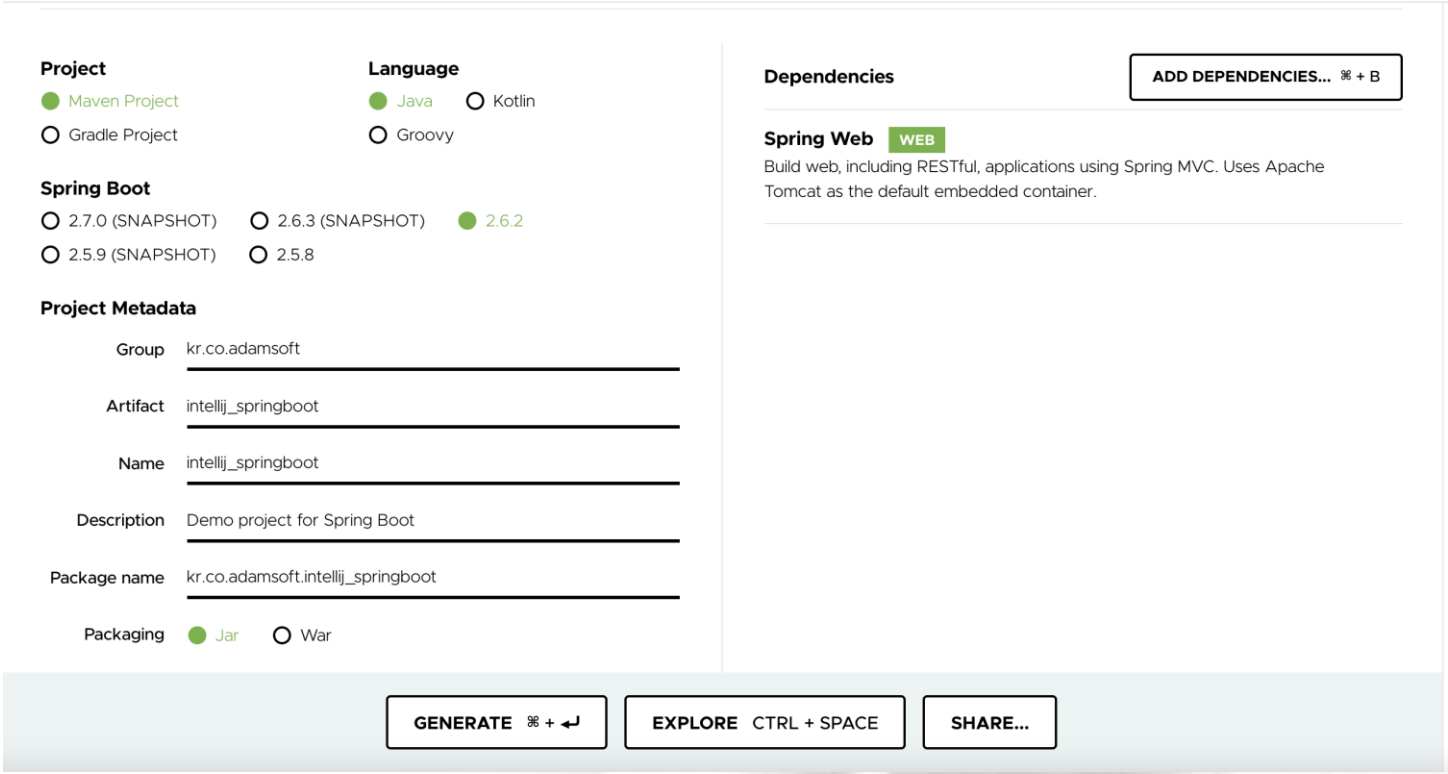
프로젝트 생성 및 실행

- STS
  - ✓ 브라우저를 실행시켜 http://localhost:8080 을 입력



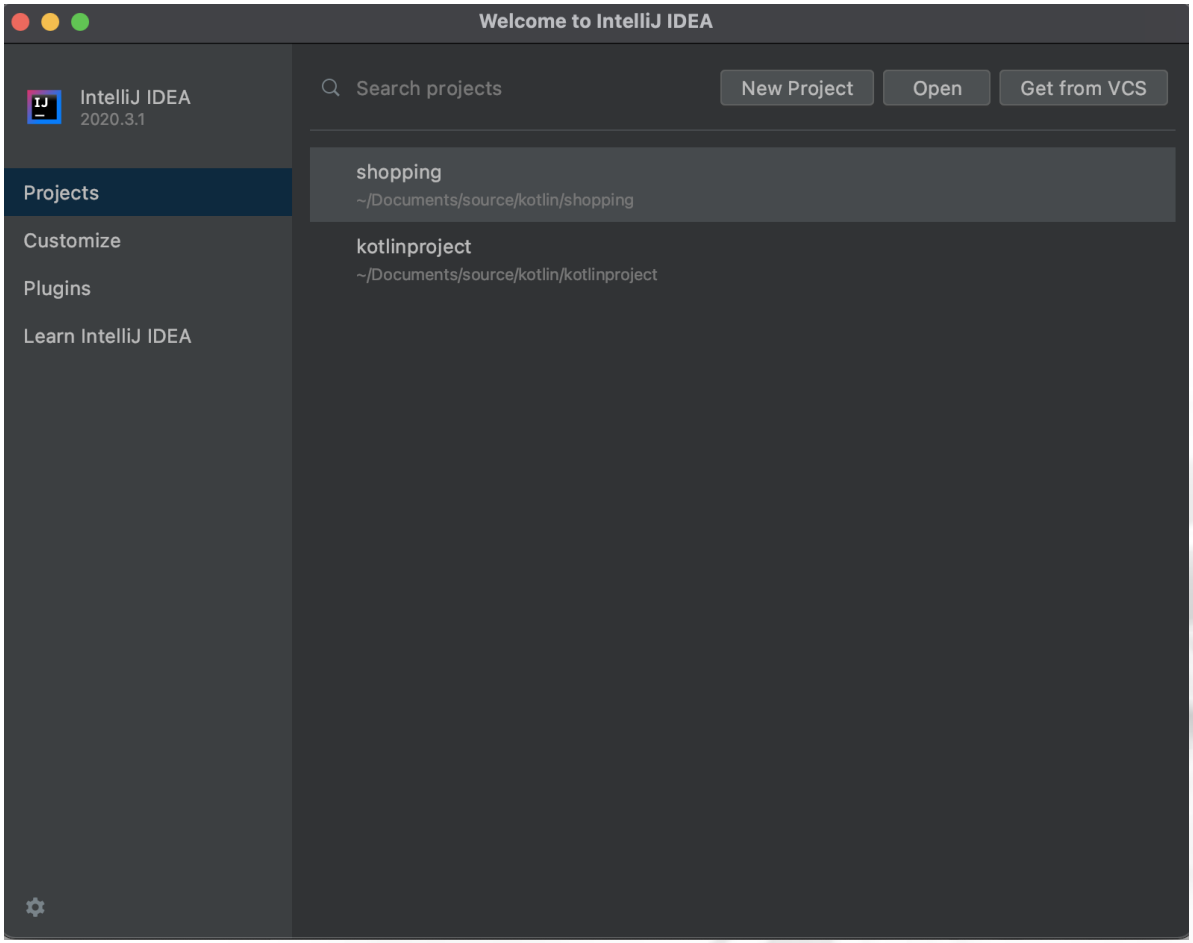
프로젝트 생성 및 실행

- Web 에서 프로젝트 생성
  - ✓ 스프링 이니셜라이저에 접속: https://start.spring.io/
  - ✓ 옵션을 설정하고 Generate 를 클릭



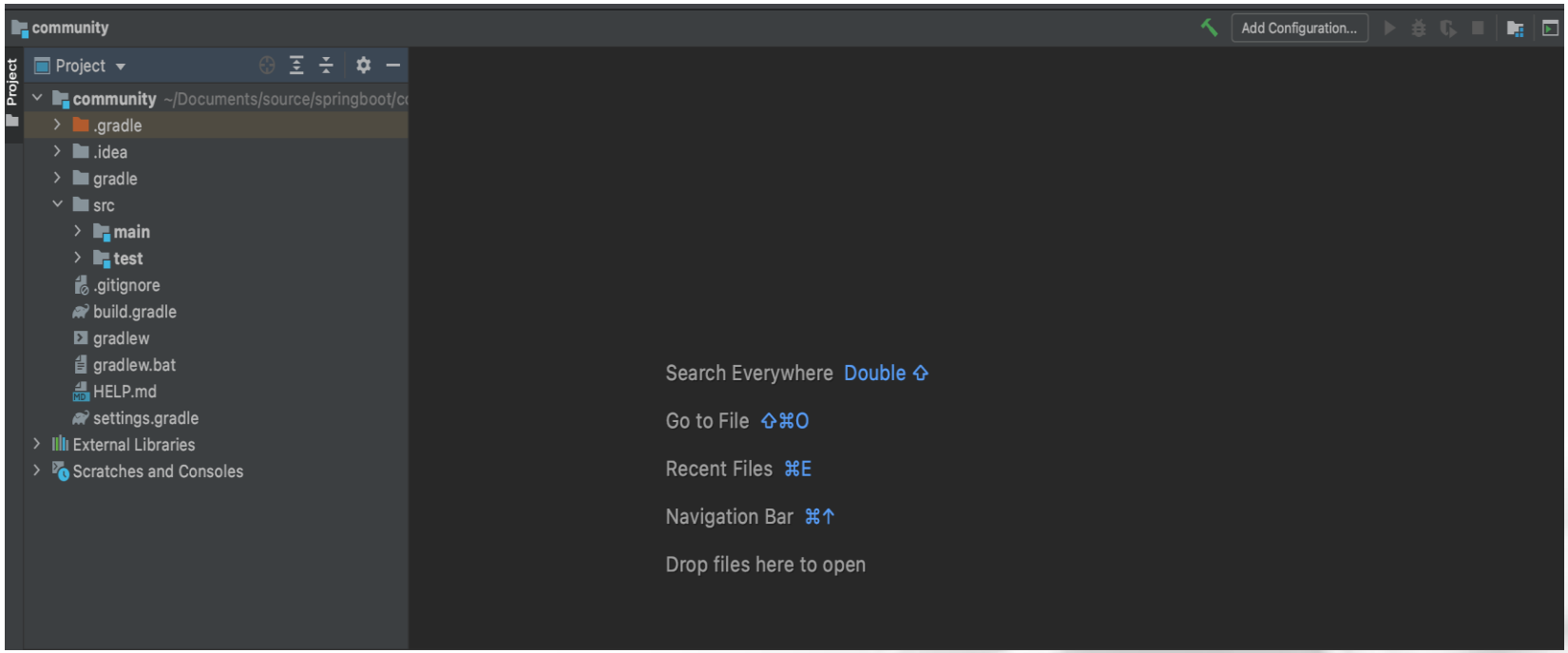
프로젝트 생성 및 실행

- Web 에서 프로젝트 생성
  - ✓ 다운로드 받은 zip 파일을 압축 해제
  - ✓ IntelliJ 실행



프로젝트 생성 및 실행

- Web 에서 프로젝트 생성
  - ✓ IntelliJ 에서 [Open] 메뉴를 누르고 압축을 해제한 디렉토리를 선택



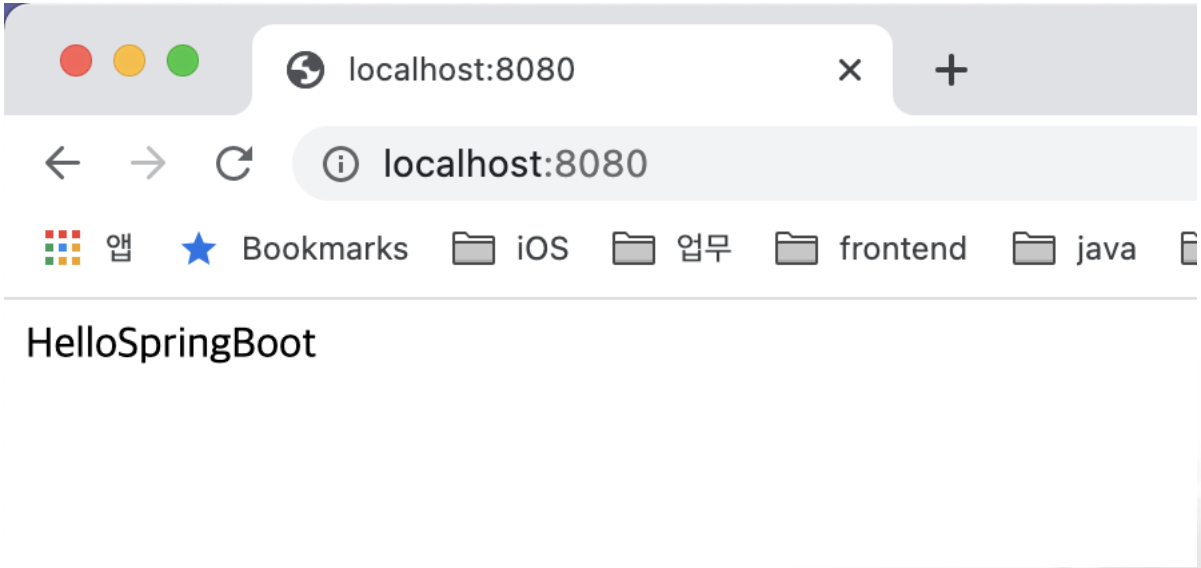
프로젝트 생성 및 실행

- Web 에서 프로젝트 생성
  - ✓ CommunityApplication.java 파일을 수정한 후 마우스 오른쪽을 클릭 한 후 실행

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@SpringBootApplication
public class CommunityApplication {

    public static void main(String[] args) {
            SpringApplication.run(CommunityApplication.class, args);
    }

    @GetMapping
    public String home(){
            return "HelloSpringBoot";
    }

}
```

프로젝트 생성 및 실행

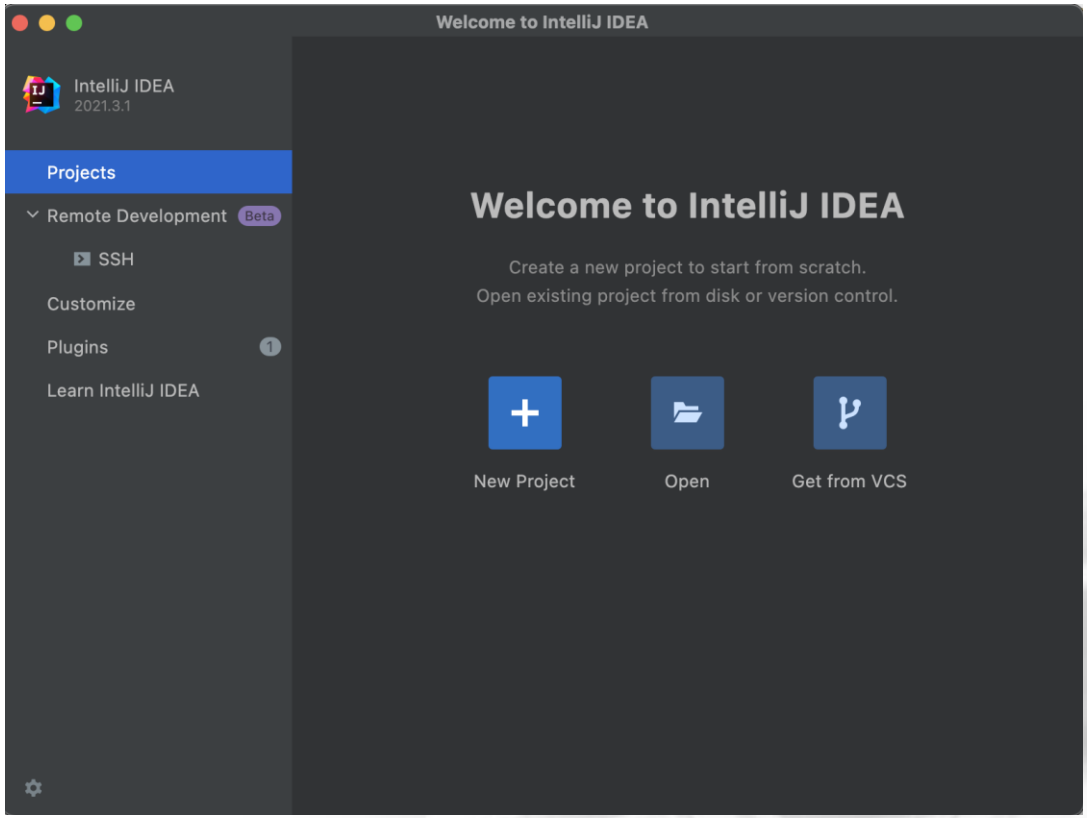
- Web 에서 프로젝트 생성
  - ✓ Browser를 실행한 후 http://localhost:8080을 입력



프로젝트 생성 및 실행

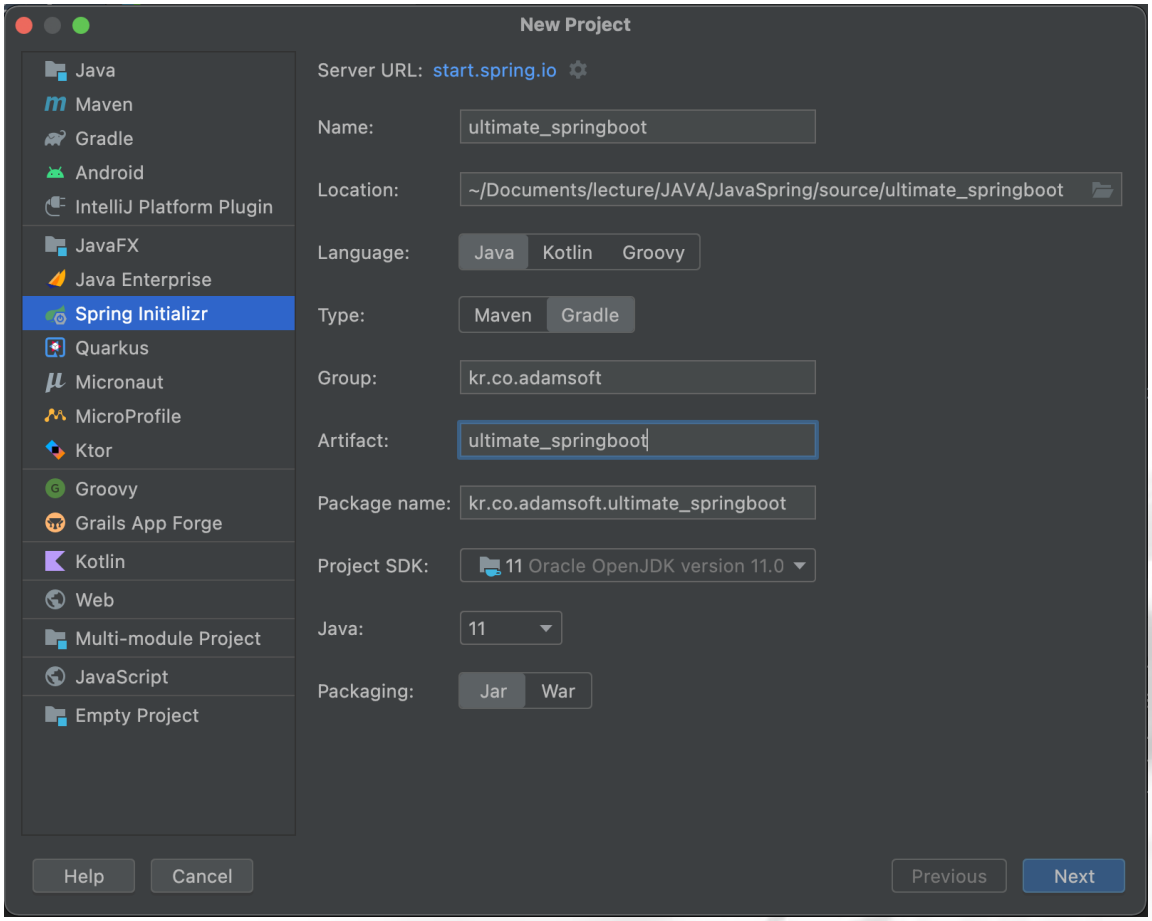
❖ IntelliJ – Ultimate 버전에서 Spring Boot Project 생성 및 실행

✓ 첫 화면에서 New Project 선택: [File] – [New] – [Project]



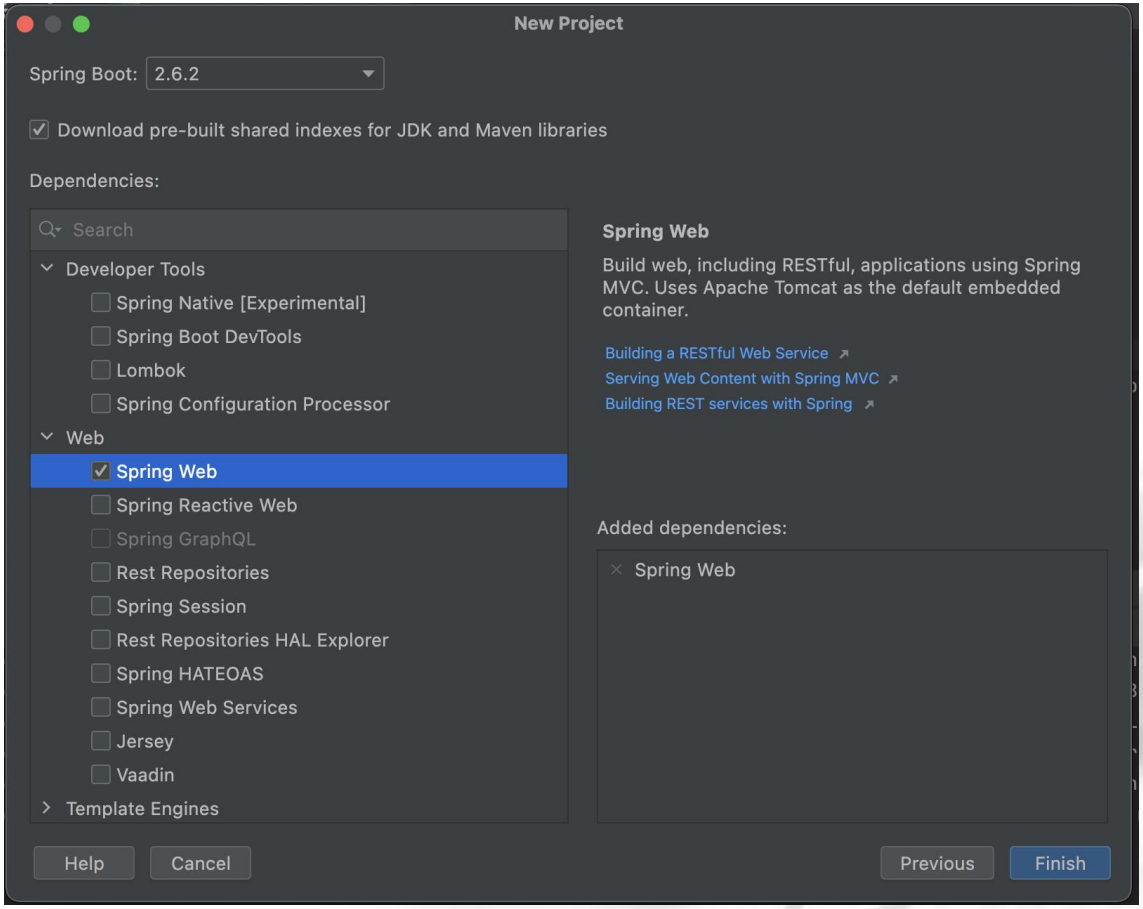
프로젝트 생성 및 실행

- IntelliJ – Ultimate 버전에서 Spring Boot Project 생성 및 실행
  - ✓ Spring Initializer 선택



프로젝트 생성 및 실행

- IntelliJ – Ultimate 버전에서 Spring Boot Project 생성 및 실행
  - ✓ 필요한 의존성 선택



프로젝트 생성 및 실행

- IntelliJ – Ultimate 버전에서 Spring Boot Project 생성 및 실행
  - ✓ SampleController 클래스를 생성한 후 실행한 후 브라우저에 localhost:포트번호/hello를 입력

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SampleController {
    @GetMapping("/hello")
    public String [] hello(){
        return new String[]{"STS", "IntelliJ"};
    }
}
```

프로젝트 생성 및 실행

- IntelliJ – Ultimate 버전에서 Spring Boot Project 생성 및 실행
  - ✓ 제공되는 클래스를 실행

프로젝트 생성 및 실행

- 프로젝트 실행 실패
  - ✓ port 충돌이 발생한다면 src/main/resources 디렉토리의 application.properties 파일에 server.port=기존에 사용하던 포트르 제외한 포트번호 를 추가해서 실행

Build Tool

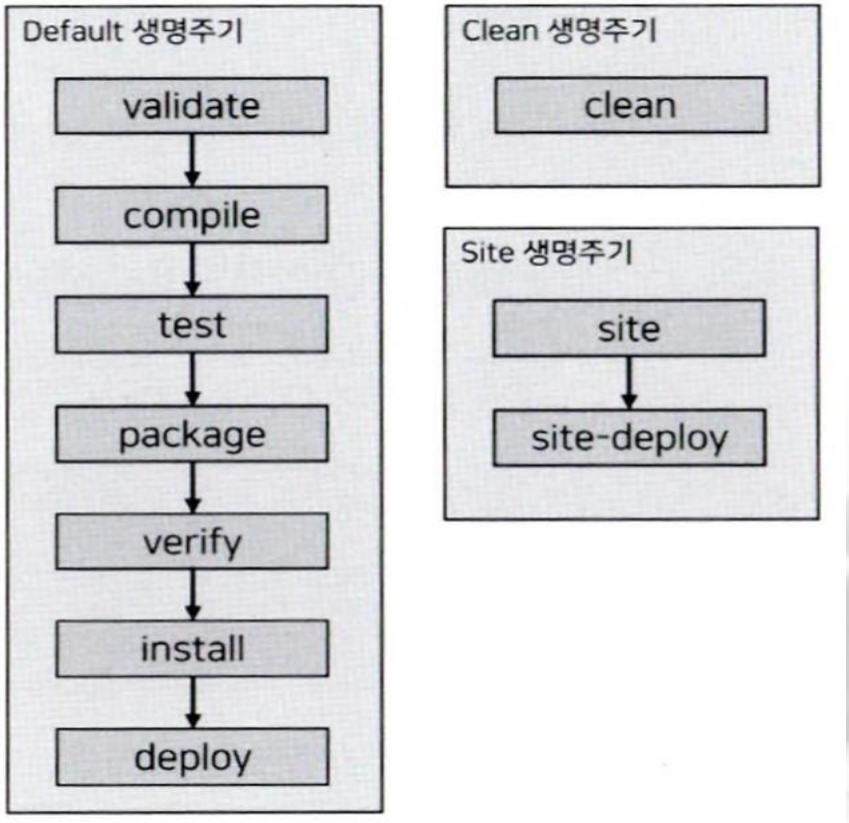
- Maven
  - ✓ Maven
    - 아파치 메이븐은 자바 기반의 프로젝트를 빌드하고 관리하는 도구 - https://maven.apache.org
    - 초창기 자바 프로젝트의 대표적 관리 도구였던 Ant를 대체하기 위해 개발
    - pom.xml 파일에 필요한 라이브러리를 추가하면 해당 라이브러리에 필요한 라이브러리까지 함께 내려받아 관리
    - 기능
      - 프로젝트 버전과 아티팩트를 관리
      - 의존성을 관리하고 설정된 패키지 형식으로 빌드를 수행
      - 빌드를 수행하기 전에 단위 테스트를 통해 작성된 애플리케이션 코드의 정상 동작 여부를 확인
      - 배포 빌드가 완료된 패키지를 원격 저장소에 배포

Build Tool

- Maven
  - ✓ Maven
    - ● 수명 주기
      - ◆ clean: 이전 빌드가 생성한 모든 파일을 제거
      - ◆ validate: 프로젝트를 빌드하는 데 필요한 모든 정보를 사용할 수 있는지 검토
      - ◆ compile: 프로젝트의 소스 코드를 컴파일
      - ◆ test: 단위 테스트 프레임워크를 사용해 테스트를 실행
      - ◆ package: 컴파일한 코드를 가져와서 JAR 등의 형식으로 패키징을 수행
      - ◆ verify: 패키지가 유효하며 일정 기준을 충족하는지 확인
      - ◆ install: 프로젝트를 사용하는 데 필요한 패키지를 로컬 저장소에 설치
      - ◆ deploy: 프로젝트를 통합 또는 릴리스 환경에서 다른 곳에 공유하기 위해 원격 저장소에 패키지를 복사
      - ◆ site: 메이븐의 설정 파일 정보를 기반으로 프로젝트의 문서 사이트를 생성
      - ◆ site-deploy: 생성된 사이트 문서를 웹 서버에 배포

Build Tool

- ❖ Maven
  - ✓ Maven
    - ● 수명 주기

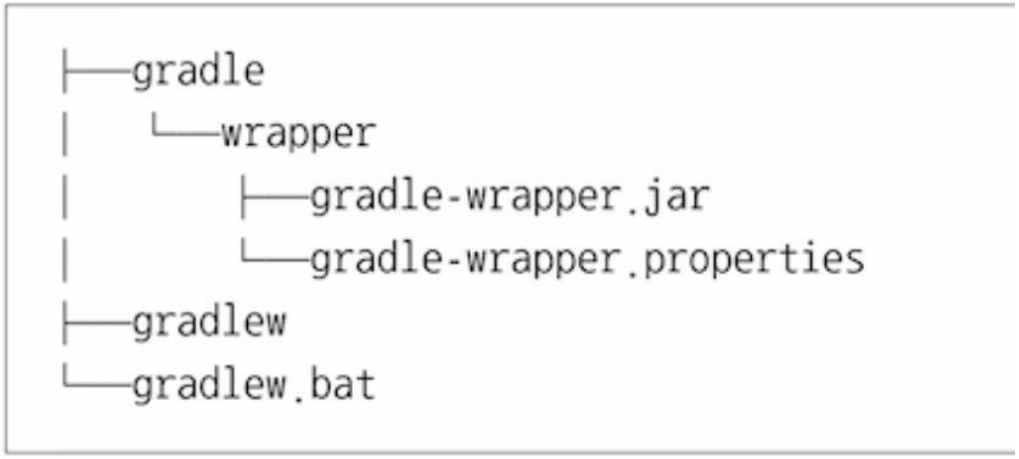


Build Tool

- gradle
  - ✓ Groovy 기반의 빌드 도구
  - ✓ Ant 로부터 기본적인 빌드 도구의 기능을 가져오고 Maven 으로 부터 의존 라이브러리 관리 기능을 가져와서 사용
  - ✓ Multi Project 구성 시에는 Maven 처럼 상속 구조가 아닌 설정 주입 방식을 사용하여 훨씬 유연하게 빌드 환경을 구성
  - ✓ gradle 설치: https://gradle.org/install/
    - Mac - $ brew install gradle
    - Windows는 다운로드 받아서 설치
  - ✓ gradle 설치 이유
    - gradle wrapper 와 관련된 설정을 해주는데 첫 설정 시 gradle 관련 빌드 설정을 자동으로 해주며 이를 이용해서 Git 과 같은 Version Control System 에서 관리하면 공동 작업자들이 gradle 설치 및 버전 관리를 편리하게 할 수 있음

Build Tool

- gradle
  - ✓ gradle 설정 관련 기본 구조



- gradlew: 리눅스 및 Mac OS 용 Shell Script
- gradlew.bat: 윈도우즈 용 Batch Script
- gradle-wrapper.jar: Wrapper JAR
- gradle-wrapper.properties: gradle 설정 정보 프로퍼티 파일(버전 정보 등)

실행 환경 속성 설정

- 실행 환경 속성 파일
  - ✓ src/main/resources 디렉토리에 있는 application.properties 파일을 이용해서 설정 - .을 이용해서 속성의 레벨을 설정하고 = 을 이용해서 값을 설정
  - ✓ 기존 파일을 제거하고 동일한 디렉토리에 application.yml 파일을 이용해서 설정 - :을 이용해서 속성의 레벨을 설정하고 값은 : 다음에 설정

실행 환경 속성 설정

- application.properties
  - ✓ 스프링 부트 애플리케이션 실행 시 사용하는 여러 가지 설정 값들을 정의하는 파일
  - ✓ src/main/resources 디렉토리 아래에 자동으로 생성되며 자동으로 생성되지 않았다면 직접 생성해서 사용 가능
  - ✓ 개발 환경, 테스트 환경, 운영 환경에 따라서 연결해야 할 데이터베이스, port, debug level 등을 나눠야 한다면 다음 명명 규칙으로 설정 파일을 생성

    application-{profile}.properties

    - 개발 환경의 설정 파일은 application-dev.properties로 만들고 운영 환경의 설정 파일은 application-prod.properties로 생성
    - 실행되는 환경에 따라서 어떤 설정 파일을 사용할지를 jar 파일 실행 시 VM 옵션 등을 통해 지정할 수 있음
  - ✓ application.properties에 설정해 둔 값을 자바 코드에서 사용해야 한다면 @Value 어노테이션을 통해서 읽어올 수 있음

실행 환경 속성 설정

- YAML을 이용한 설정
  - ✓ YAML
    - YAML은 XML, C, 파이썬, 펄, RFC2822에서 정의된 e-mail 양식에서 개념을 얻어 만들어진 사람이 쉽게 읽을 수 있는 데이터 직렬화 양식
    - YAML 이라는 이름은 "YAML은 마크업 언어가 아니다(YAML Ain't Markup Language)" 라는 재귀적인 이름에서 유래
    - YAML의 뜻은 또 다른 마크업 언어 (Yet Another Markup Language)였으나 YAML의 핵심은 문서 마크업이 아닌 데이터 중심에 있다는 것을 보여주기 위해 이름을 변경
    - 오늘날 XML과 JSON이 데이터 직렬화에 쓰이기 시작하면서 많은 사람들이 YAML을 가벼운 마크업 언어 로 사용하려 함
  - ✓ YAML을 이용한 설정
    - src/main/resources 디렉토리의 application.properties 파일을 삭제
    - src/main/resources 디렉토리의 application.yml 파일을 생성하고 작성한 후 실행

```
server:
  port: 80
```

REST API

- REST API
  - ✓ REST
    - Representational State Transfer의 약자로 WWW 와 같은 분산 하이퍼미디어 시스템 아키텍처의 한 형식
    - 주고받는 자원(Resource)에 이름을 규정하고 URI에 명시해 HTTP 메서드(GET, POST, PUT, DELETE)를 통해 해당 자원의 상태를 주고받는 것을 의미
  - ✓ REST API
    - REST 아키텍처를 따르는 시스템/애플리케이션 인터페이스
    - REST 아키텍처를 구현하는 웹 서비스를 RESTful 하다 라고 표현

REST API

- REST API
  - ✓ 특징
    - 유니폼 인터페이스
      - 일관된 인터페이스를 의미
      - REST 서버는 HTTP 표준 전송 규약을 따르기 때문에 어떤 프로그래밍 언어로 만들어졌느냐 와 상관없이 플랫폼 및 기술에 종속되지 않고 타 언어, 플랫폼, 기술 등과 호환해 사용할 수 있다는 것을 의미
    - 무상태성
      - 무상태성이란 서버에 상태 정보를 따로 보관하거나 관리하지 않는다는 의미
      - 서버는 클라이언트가 보낸 요청에 대해 세션이나 쿠키 정보를 별도 보관하지 않음
      - 하나의 클라이언트가 여러 요청을 보내든 여러 클라이언트가 각각 하나의 요청을 보내든 개별적으로 처리
      - 서버가 불필요한 정보를 관리하지 않으므로 비즈니스 로직의 자유도가 높고 설계가 단순

REST API

- REST API
  - ✓ 특징
    - 캐시 가능성
      - HTTP의 캐싱 기능을 적용할 수 있음
      - 기능을 이용하기 위해서는 응답과 요청이 모두 캐싱 가능한지(Cacheable) 명시가 필요하며 캐싱이 가능한 경우 클라이언트에서 캐시에 저장해두고 같은 요청에 대해서는 해당 데이터를 가져다 사용
      - 서버의 트랜잭션 부하가 줄어 효율적이며 사용자 입장에서 성능이 개선
    - 레이어 시스템
      - 네트워크 상의 여러 계층으로 구성될 수 있음(Layered System)
      - 서버의 복잡도와 관계없이 클라이언트는 서버와 연결되는 포인트만 알면 됨
    - 클라이언트 - 서버 아키텍처
      - REST 서버는 API를 제공하고 클라이언트는 사용자 정보를 관리하는 구조로 분리해 설계
      - 서로에 대한 의존성을 낮추는 기능

REST API

- REST API
  - ✓ URL 설계 규칙
    - URI의 마지막에는 / 를 포함하지 않음
    - 언더바 대신 하이픈(-}을 이용
    - URL에는 행위(동사)가 아닌 결과{명사}를 포함
      - 옳은 예) http://localhost.com/product/123
      - 잘못된 예) http://localhost.com/delete-product/123
      - 행위는 HTTP 메서드로 표현
    - URI는 소문자로 작성
    - 파일의 확장자는 URL에 포함하지 않음 – HTTP에서 제공하는 Accept 헤더를 사용하는 것을 권장

REST API

- REST API 테스트
  - ✓ 프로젝트 생성
    - build.gradle 수정: Spring Boot DevTools, Lombok 의 의존성 추가 – 추가한 후 reload

```
dependencies {
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    annotationProcessor 'org.projectlombok:lombok'

    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

REST API

- GET API
  - ✓ GET API는 웹 애플리케이션 서버에서 값을 가져올 때 사용하는 API
  - ✓ 클라이언트 요청을 처리할 Controller 클래스 생성 – controller.GetController

```
@RestController
@RequestMapping("/api/v1/get-api")
public class GetController {

}
```

    - 클래스 수준에서 @RequestMapping을 설정하면 내부에 선언한 메서드의 URL 리소스 앞에 @RequestMapping 의 값이 공통 값으로 추가됨

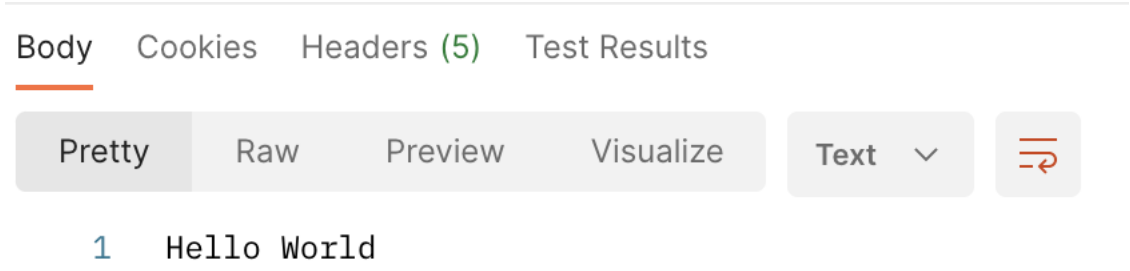
REST API

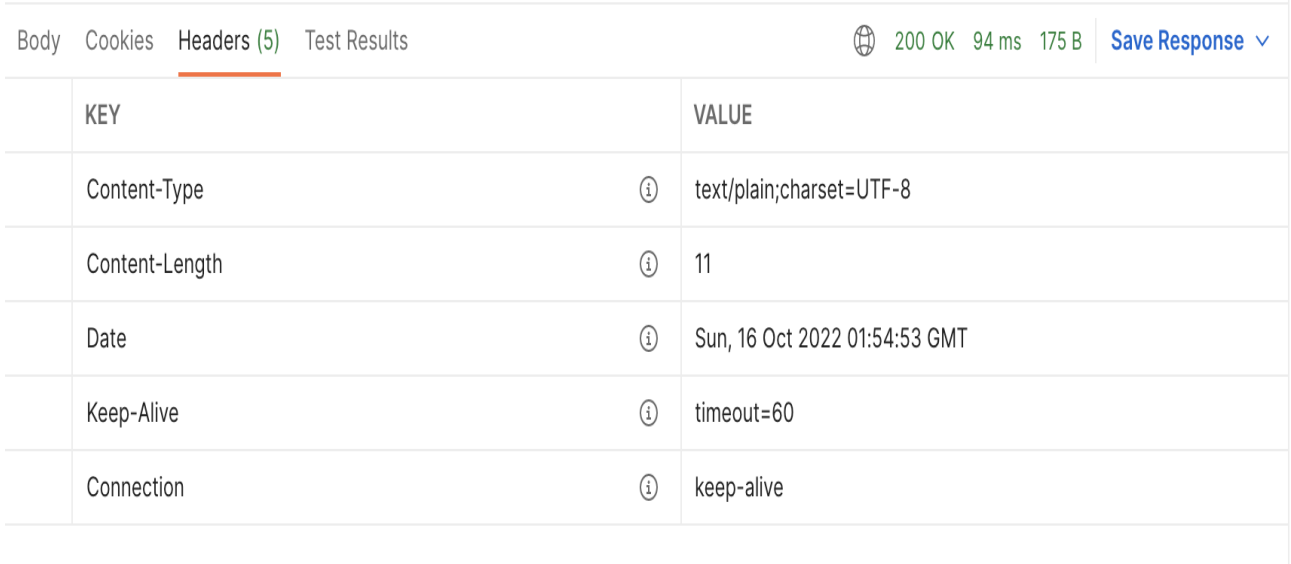
- GET API
  - ✓ @RequestMapping 을 이용한 요청 처리
    - GetController 클래스에 메서드를 추가하고 실행한 후 테스트

```
// http://localhost/api/v1/get-api/hello
@RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String getHello() {
        return "Hello World";
}
```

REST API

- GET API
  - ✓ @RequestMapping 을 이용한 요청 처리





REST API

- GET API
  - ✓ @RequestMapping 을 이용한 요청 처리
    - 스프링 4.3 버전 이후 추가된 HTTP 메서드 요청에 따른 어노테이션
      - @GetMapping
      - @PostMapping
      - @PutMapping
      - @DeleteMapping

REST API

- GET API
  - ✓ @PathVariable을 활용한 GET 요청 처리
    - 데이터를 URL 자체에 담아 요청할 수 있음
    - 중괄호({ })로 표시된 위치의 값을 받아 처리 - 실제 요청 시 중괄호는 들어가지 않으며 값만 존재
    - 값을 간단히 전달할 때 주로 사용하는 방법이며 GET 요청에서 많이 사용
    - @GetMapping 어노테이션의 값 으로 URL을 입력할 때 중괄호를 사용해 어느 위치에서 값을 받을지 지정해야 하고 메서드의 매개변수와 그 값을 연결하기 위해 @PathVariable을 명시하며, @GetMapping 어노테이션 과 @PathVariable에 지정된 변수의 이름을 동일하게 맞춰야 함

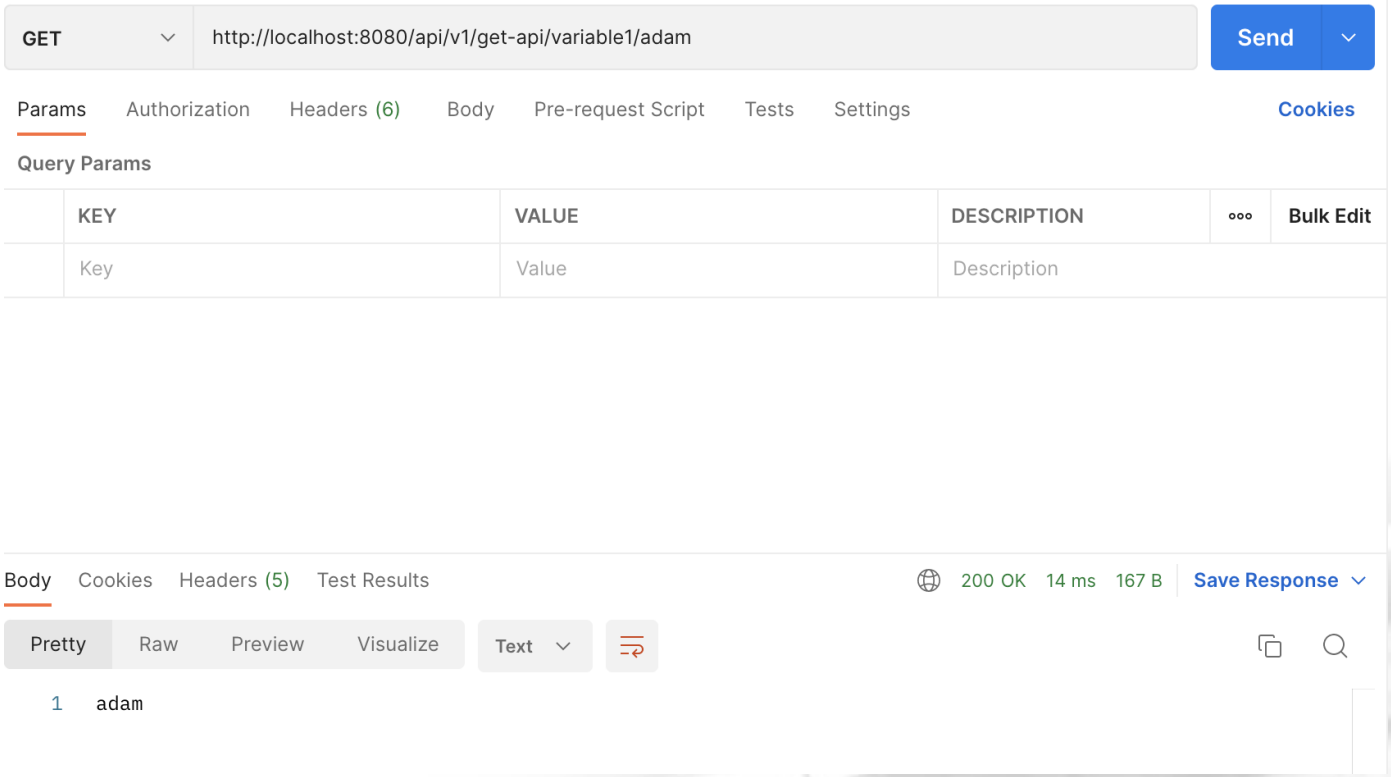
REST API

- GET API
  - ✓ @PathVariable을 활용한 GET 요청 처리
    - GetController 클래스에 메서드를 추가하고 확인

```
// http://localhost/api/v1/get-api/variable1/{String 값}
@GetMapping(value = "/variable1/{variable}")
public String getVariable1(@PathVariable String variable) {
        return variable;
}
```

REST API

- GET API
  - ✓ @PathVariable을 활용한 GET 요청 처리



REST API

- GET API
  - ✓ @RequestParam 을 활용한 GET 요청 처리
    - 쿼리 형식으로 값을 전달할 수 있음
    - URI에서 ? 를 기준으로 우측에 {키}={값} 형태로 구성해서 요청을 전송
    - 애플리케이션에서 이 같은 형식을 처리하려면 @RequestParam을 활용하면 되는데 매개변수 부분에 @RequestParam 어노테이션을 명시해 쿼리 값과 매핑

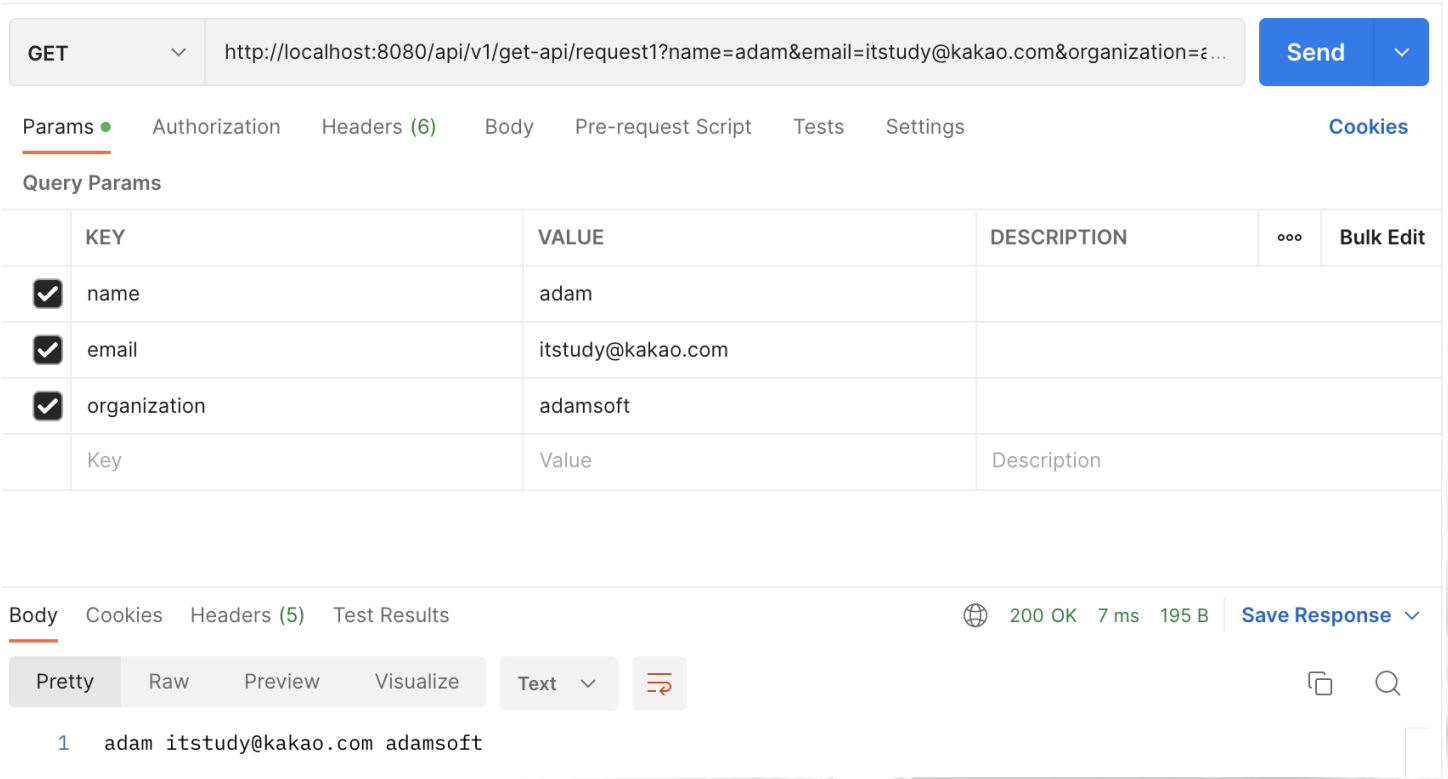
REST API

- GET API
  - ✓ @RequestParam 을 활용한 GET 요청 처리
    - GetController 클래스에 메서드를 추가하고 확인

```
// http://localhost/api/v1/get-
    api/request1?name=adam&email=itstudy@kakao.com&organization=adamsof
    t
@GetMapping(value = "/request1")
public String getRequestParam1(
        @RequestParam String name,
        @RequestParam String email,
        @RequestParam String organization) {
        return name + " " + email + " " + organization;
}
```

REST API

- GET API
  - ✓ @RequestParam 을 활용한 GET 요청 처리



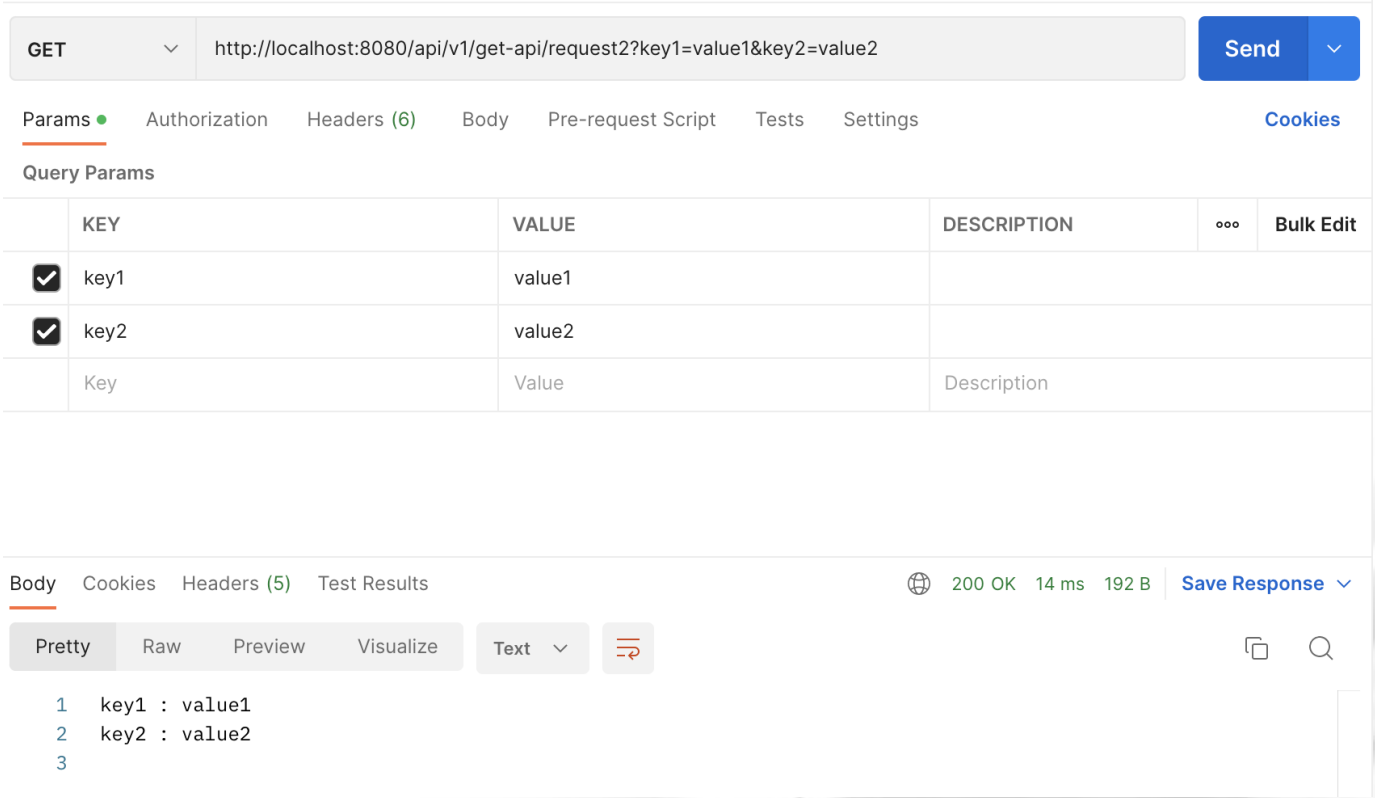
REST API

- GET API
  - ✓ @RequestParam 을 활용한 GET 요청 처리
    - 쿼리 스트링에 어떤 값이 전달될 지 모른다면 Map을 이용해서 받는 것이 가능
    - GetController 클래스에 메서드를 추가하고 확인

```
// http://localhost/api/v1/get-api/request2?key1=value1&key2=value2
@GetMapping(value = "/request2")
public String getRequestParam2(@RequestParam Map<String, String> param) {
        StringBuilder sb = new StringBuilder();

        param.entrySet().forEach(map -> {
            sb.append(map.getKey() + " : " + map.getValue() + "₩n");
        });

        return sb.toString();
}
```

REST API

- GET API
  - ✓ @RequestParam 을 활용한 GET 요청 처리



REST API

- GET API
  - ✓ DTO를 활용한 GET 요청 처리
    - 쿼리스트링을 받기 위한 클래스를 별도로 생성해서 처리하는 것이 가능
    - 하나의 데이터를 표현할 DTO 클래스 생성 – dto.MemberDTO

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public class MemberDTO {
    private String name;
    private String email;
    private String organization;
}
```

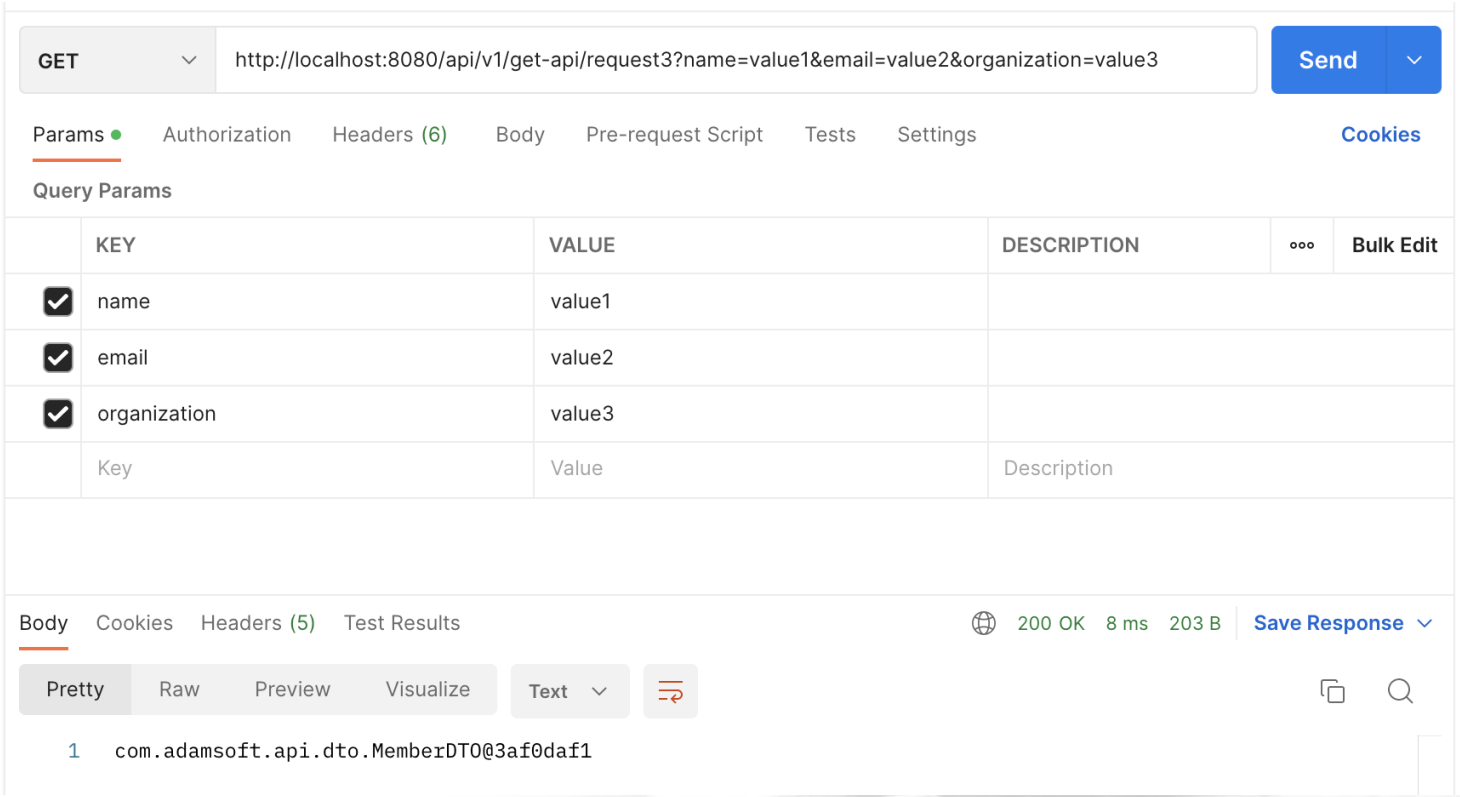
REST API

- GET API
  - ✓ DTO를 활용한 GET 요청 처리
    - GetController 클래스에 메서드를 추가하고 확인

```
// http://localhost/api/v1/get-
    api/request3?name=value1&email=value2&organization=value3
@GetMapping(value = "/request3")
public String getRequestParam3(MemberDTO memberDTO) {
        return memberDTO.toString();
}
```

REST API

- GET API
  - ✓ DTO를 활용한 GET 요청 처리



REST API

- POST API
  - ✓ POST API는 웹 애플리케이션을 통해 데이터베이스 등의 저장소에 리소스를 저장할 때 사용되는 API
  - ✓ GET API에서는 URL의 경로나 파라미터에 변수를 넣어 요청을 보냈지만 POST API에서는 저장하고자 하는 리소스나 값을 HTTP 바디(body)에 담아 서버에 전달
  - ✓ @RequestBody는 HTTP의 Body 내용을 해당 어노테이션이 지정된 객체에 매핑하는 역할
  - ✓ POST API 처리를 위한 Controller 추가 – controller.PostController

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/v1/post-api")
public class PostController {


}
```

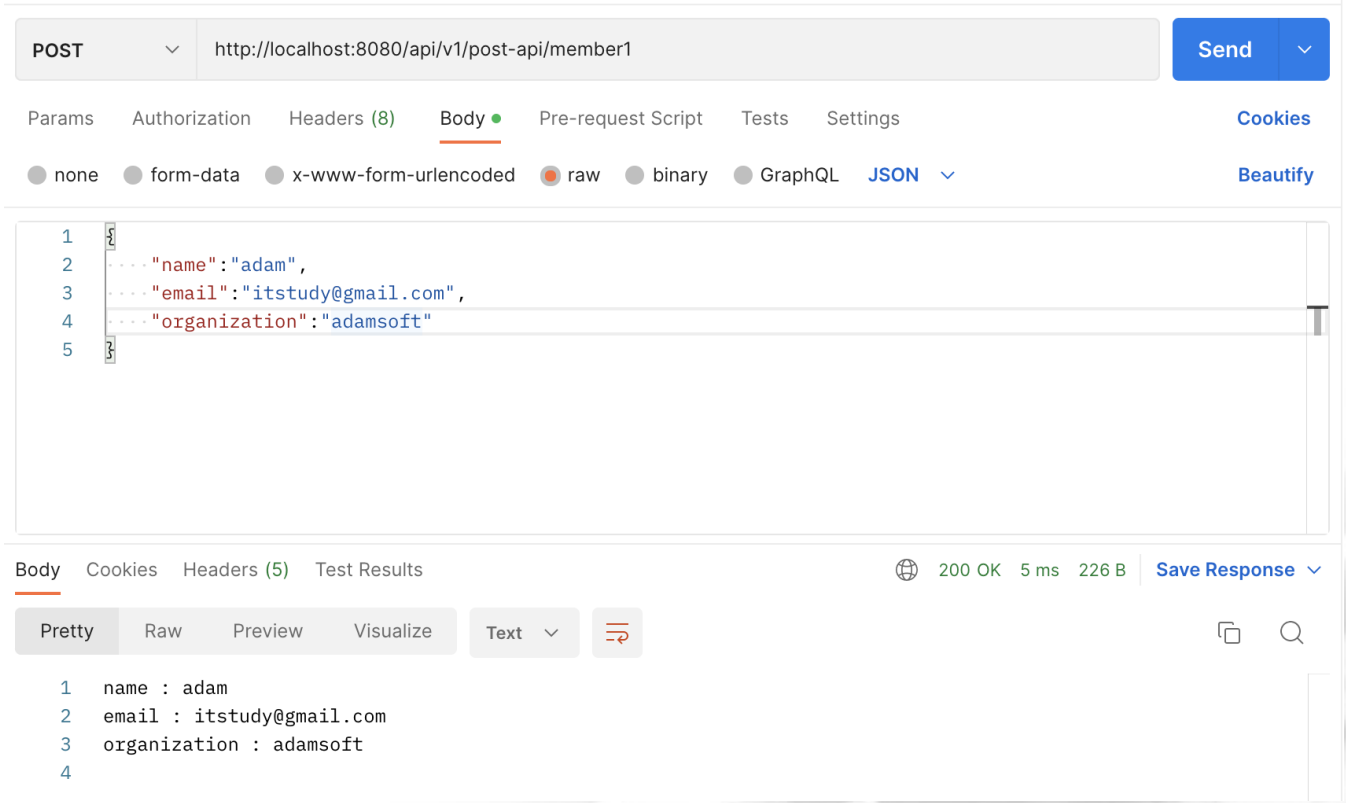
REST API

- POST API
  - ✓ PostController 에 메서드 추가

```
// http://localhost/api/v1/post-api/member1
@PostMapping(value = "/member1")
public String postMember(@RequestBody Map<String, Object> postData) {
    StringBuilder sb = new StringBuilder();
    postData.entrySet().forEach(map -> {
        sb.append(map.getKey() + " : " + map.getValue() + "₩n");
    });
    return sb.toString();
}

// http://localhost/api/v1/post-api/member2
@PostMapping(value = "/member2")
public String postMemberDto(@RequestBody MemberDTO memberDTO) {
    return memberDTO.toString();
}
```

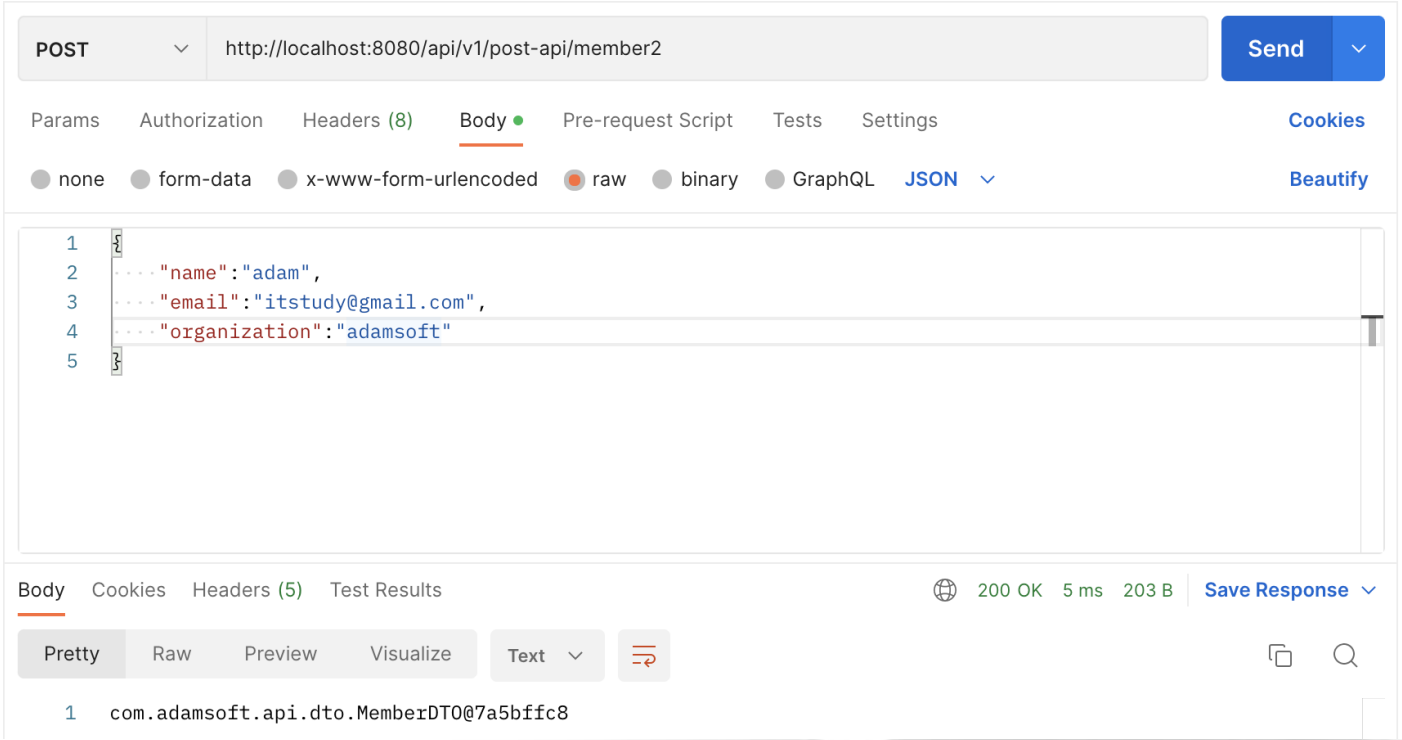
REST API

- POST API
  - ✓ PostController 에 메서드 추가



REST API

- POST API
  - ✓ PostController 에 메서드 추가



REST API

- PUT API
  - ✓ PUT API는 웹 애플리케이션을 통해 데이터베이스 등의 저장소에 리소스 값을 수정할 때 사용되는 API
  - ✓ PUT API에서는 저장하고자 하는 리소스나 값을 HTTP body에 담아 서버에 전달
  - ✓ PUT API 처리를 위한 Controller 추가 – controller.PutController

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/v1/put-api")
public class PutController {


}
```

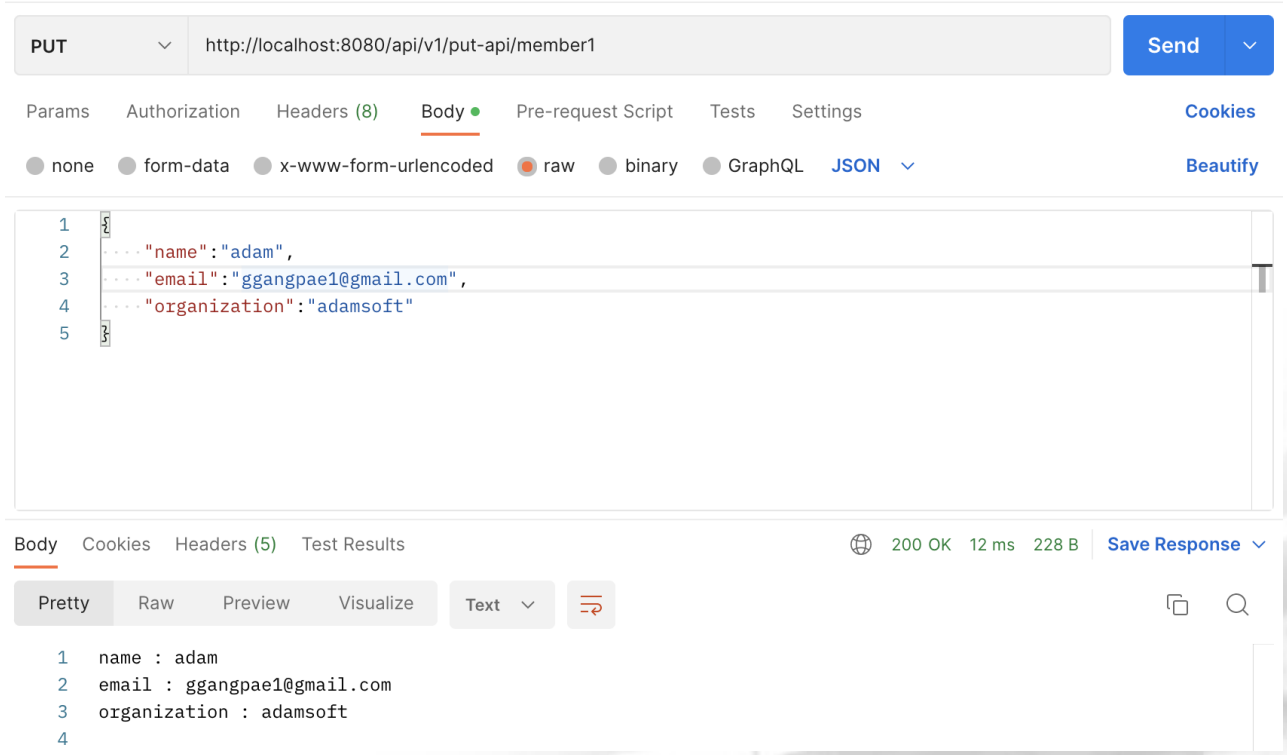
REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가

```
// http://localhost/api/v1/put-api/member1
@PutMapping(value = "/member1")
public String putMember1(@RequestBody Map<String, Object> putData) {
        StringBuilder sb = new StringBuilder();

        putData.entrySet().forEach(map -> {
            sb.append(map.getKey() + " : " + map.getValue() + "₩n");
        });

        return sb.toString();
}
```

REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가



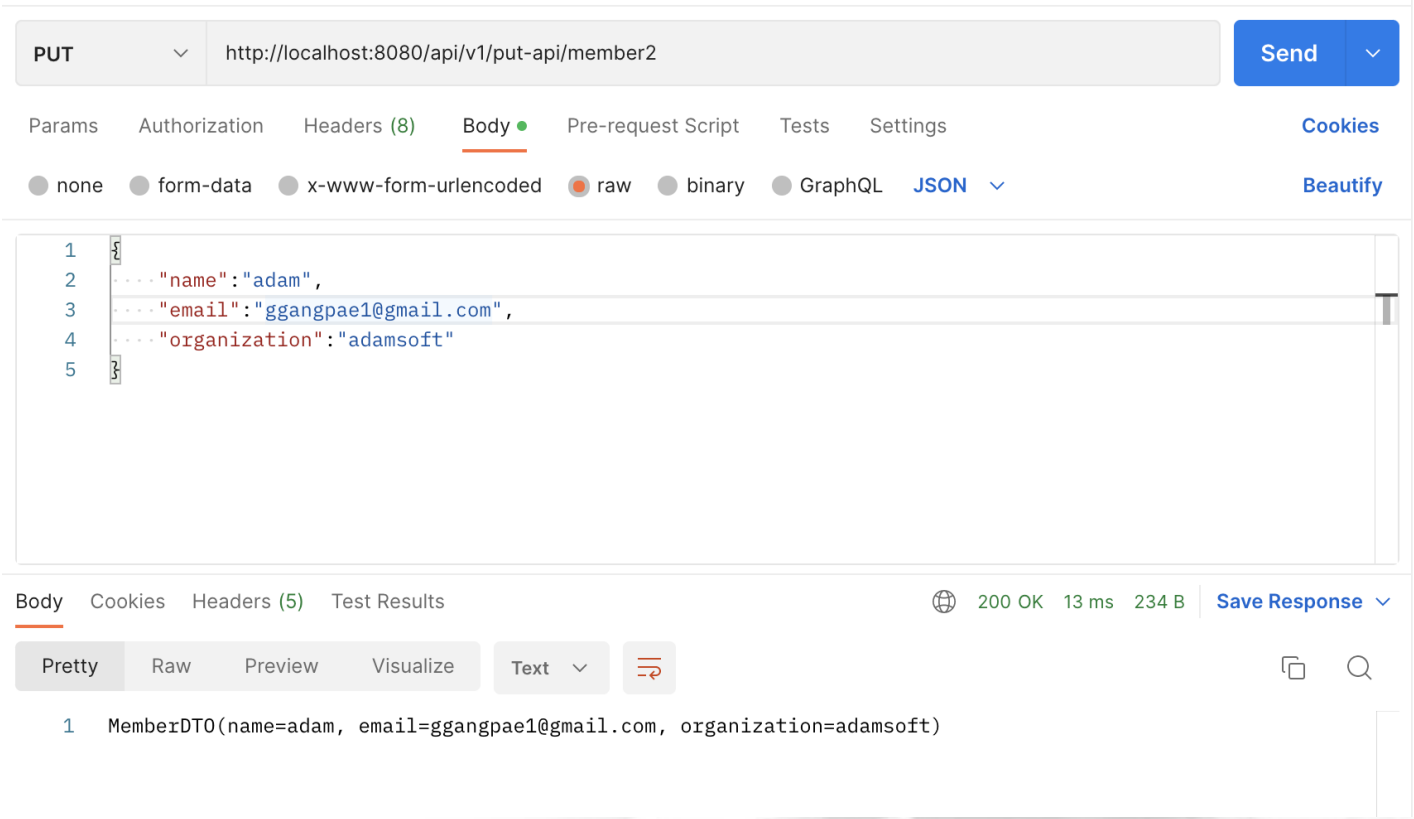
REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가

```
// http://localhost/api/v1/put-api/member2
@PutMapping(value = "/member2")
public String putMember2(@RequestBody MemberDTO memberDto) {
        return memberDto.toString();
}

// http://localhost:/api/v1/put-api/member3
@PutMapping(value = "/member3")
public MemberDTO putMember3(@RequestBody MemberDTO memberDto) {
        return memberDto;
}
```

REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가



REST API

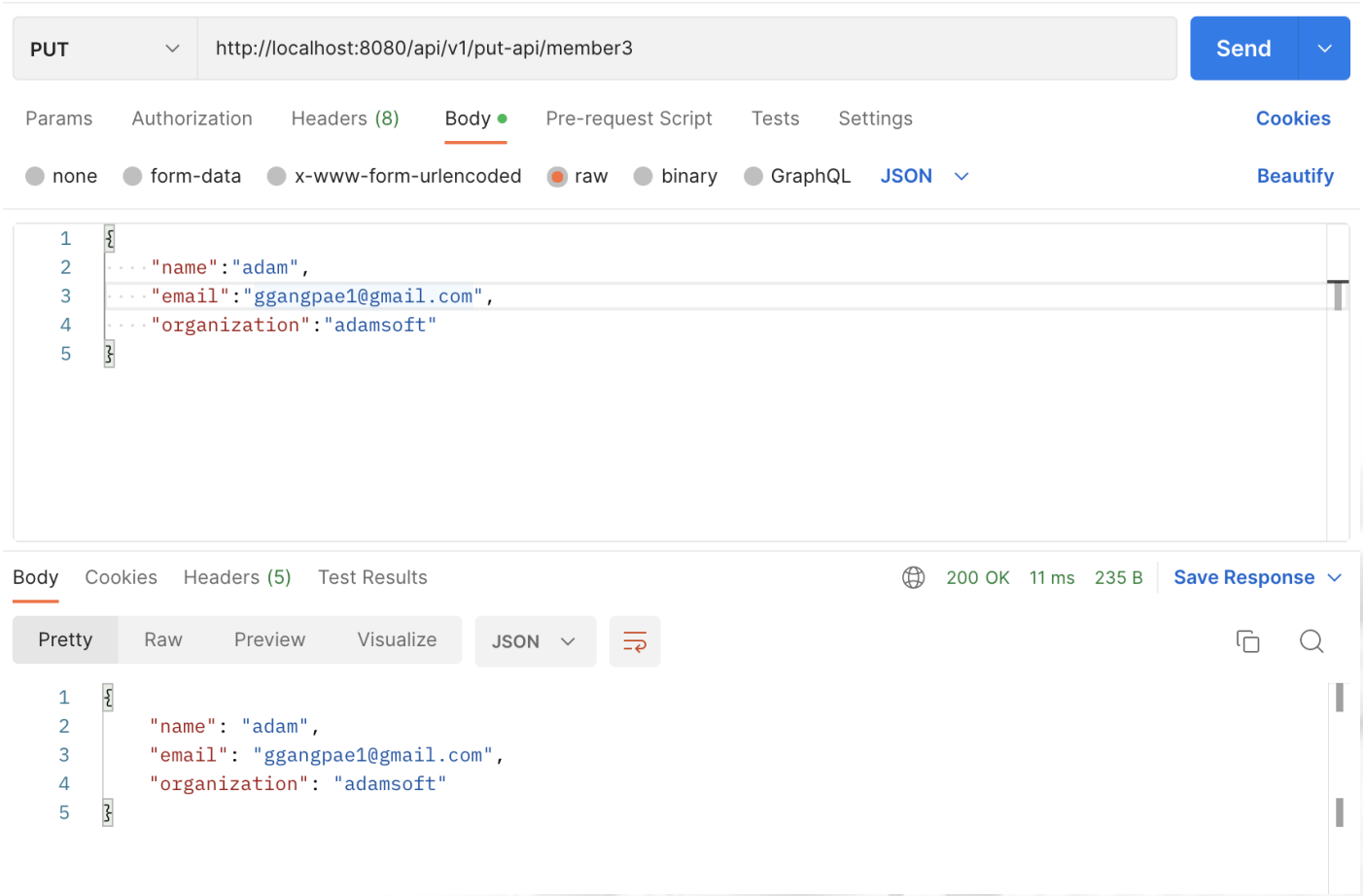
- PUT API
  - ✓ PutController 클래스에 메서드 추가

Body Cookies Headers (5) Test Results 200 OK 8 ms 234 B Save Response

KEY	VALUE
Content-Type	text/plain;charset=UTF-8
Content-Length	70
Date	Sun, 16 Oct 2022 02:57:57 GMT
Keep-Alive	timeout=60
Connection	keep-alive

REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가



REST API

- PUT API
  - ✓ PutController 클래스에 메서드 추가

Body Cookies Headers (5) Test Results 200 OK 8 ms 234 B Save Response

KEY	VALUE
Content-Type	text/plain;charset=UTF-8
Content-Length	70
Date	Sun, 16 Oct 2022 02:57:57 GMT
Keep-Alive	timeout=60
Connection	keep-alive

REST API

- ResponseEntity
  - ✓ HttpEntity는 헤더(Header) 와 Body로 구성된 HTTP 요청과 응답을 구성하는 역할을 수행
  - ✓ RequestEntity 와 ResponseEntity는 HttpEntity를 상속받아 구현한 클래스
  - ✓ ResponseEntity는 서버에 들어온 요청에 대해 응답 데이터를 구성해서 전달할 수 있는데 ResponseEntity 는 HttpEntity로부터 HttpHeaders와 Body를 가지고 자체적으로 HttpStatus를 구현

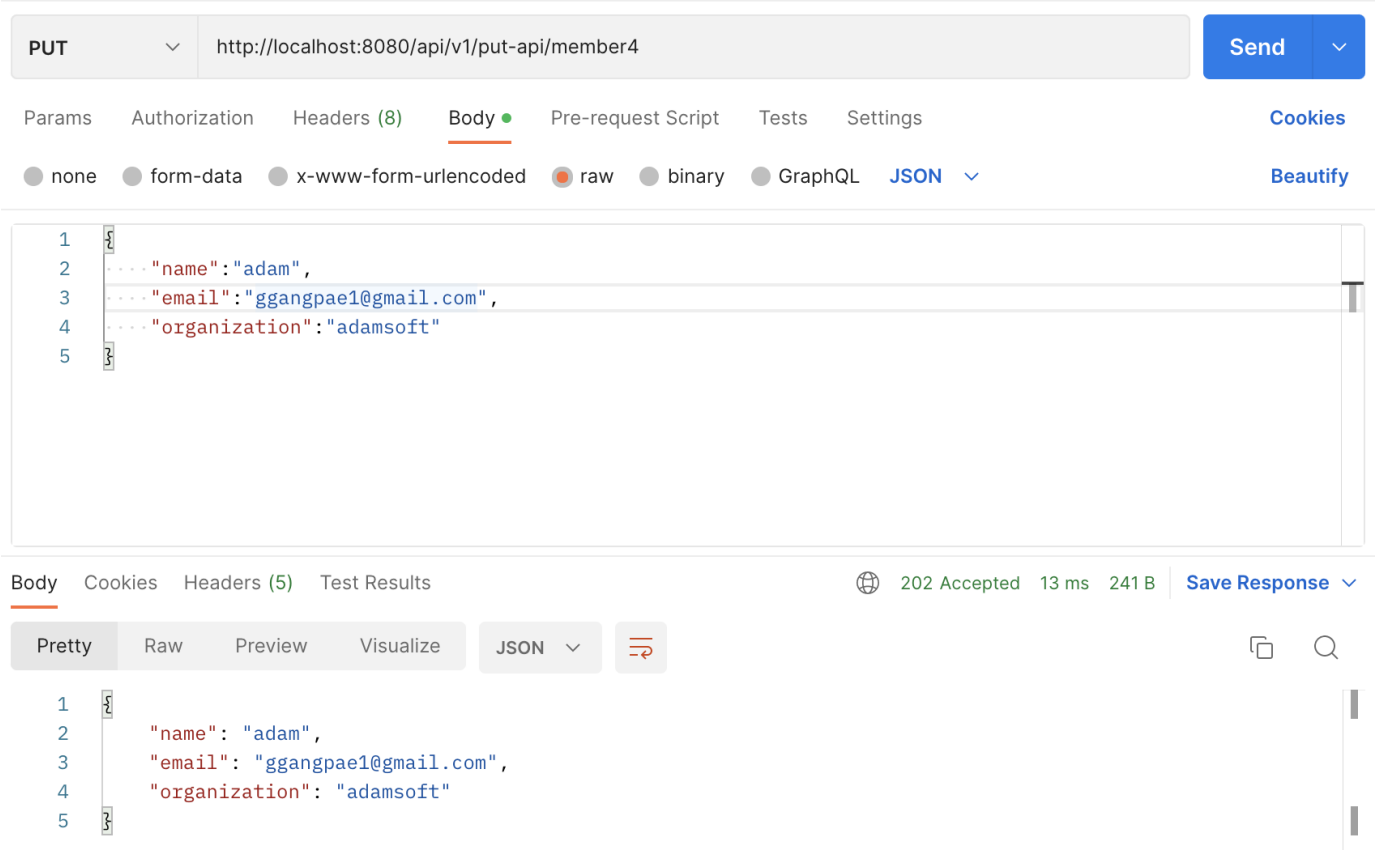
REST API

- ResponseEntity
  - ✓ PutController 클래스에 메서드 추가

```
// http://localhost/api/v1/put-api/member4
@PutMapping(value = "/member4")
public ResponseEntity<MemberDTO> putMember4(@RequestBody MemberDTO
    memberDto) {
        return ResponseEntity
            .status(HttpStatus.ACCEPTED)
            .body(memberDto);
}
```

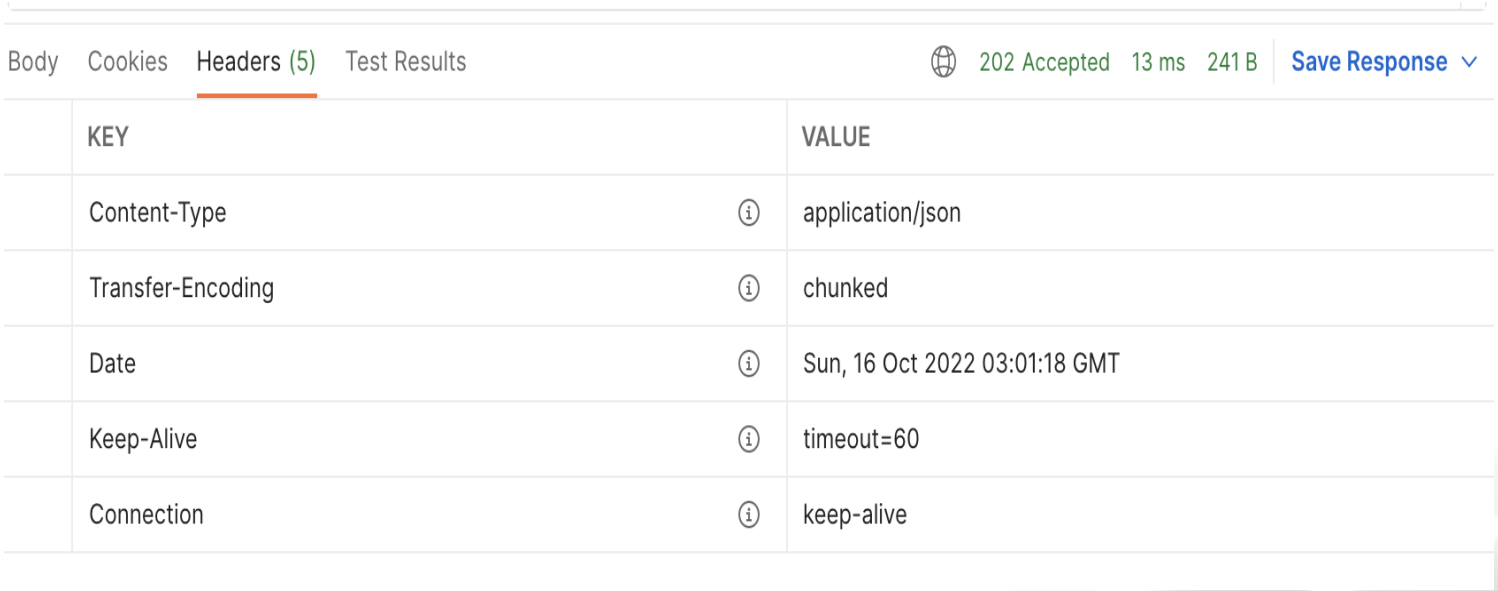
REST API

- ResponseEntity



REST API

- ResponseEntity



REST API

- DELETE API
  - ✓ DELETE API는 웹 애플리케이션 서버를 거쳐 데이터베이스 등의 저장소에 있는 리소스를 삭제할 때 사용
  - ✓ 서버에서는 클라이언트로부터 리소스를 식별할 수 있는 값을 받아 데이터베이스나 캐시에 있는 리소스를 조회하고 삭제하는 역할을 수행
  - ✓ 컨트롤러를 통해 값을 받는 단계에서는 간단한 값을 받기 때문에 GET 메서드와 같이 URI에 값을 넣어 요청을 받는 형식으로 구현

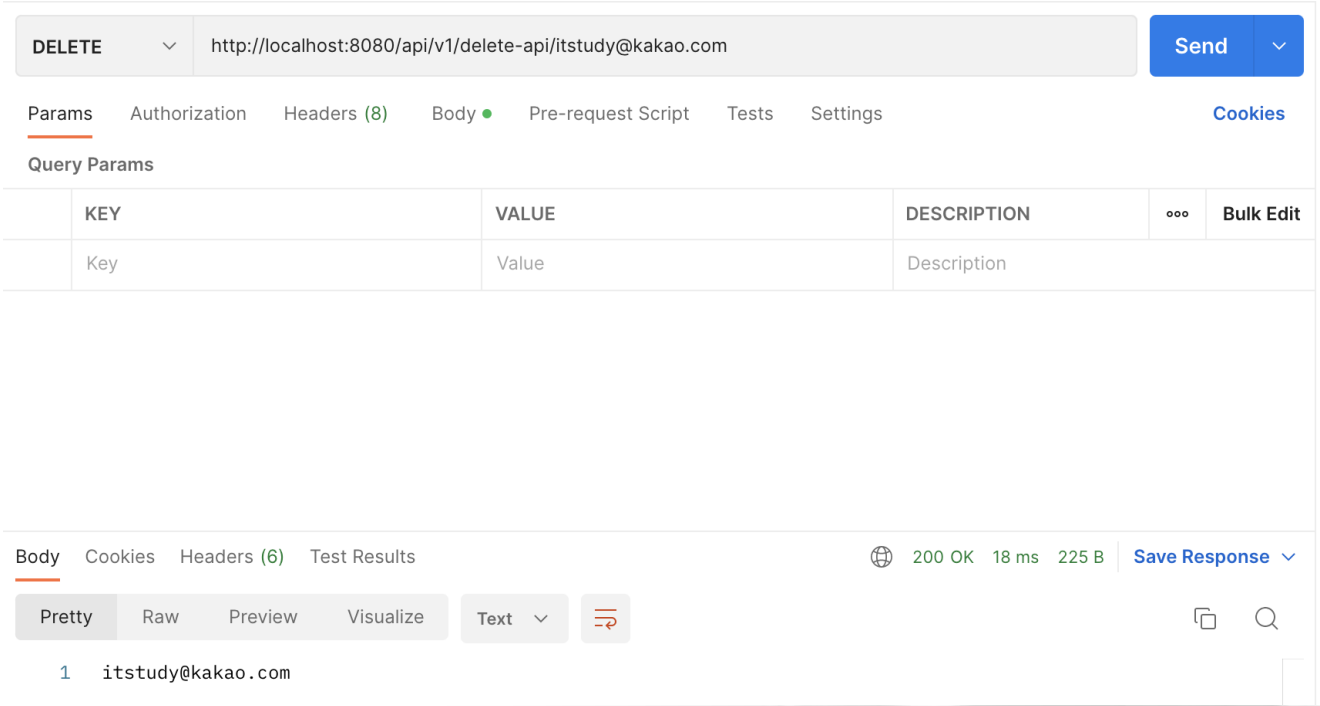
REST API

- DELETE API
  - ✓ DeleteController 클래스를 생성하고 메서드 추가

```
@RestController
@RequestMapping("/api/v1/delete-api")
public class DeleteController {
    // http://localhost:8080/api/v1/delete-api/{String 값}
    @DeleteMapping(value = "/{variable}")
    public String DeleteVariable(@PathVariable String variable) {
        return variable;
    }

    // http://localhost:8080/api/v1/delete-api/request1?email=value
    @DeleteMapping(value = "/request1")
    public String DeleteParameter(@RequestParam String email) {
        return "e-mail : " + email;
    }
}
```

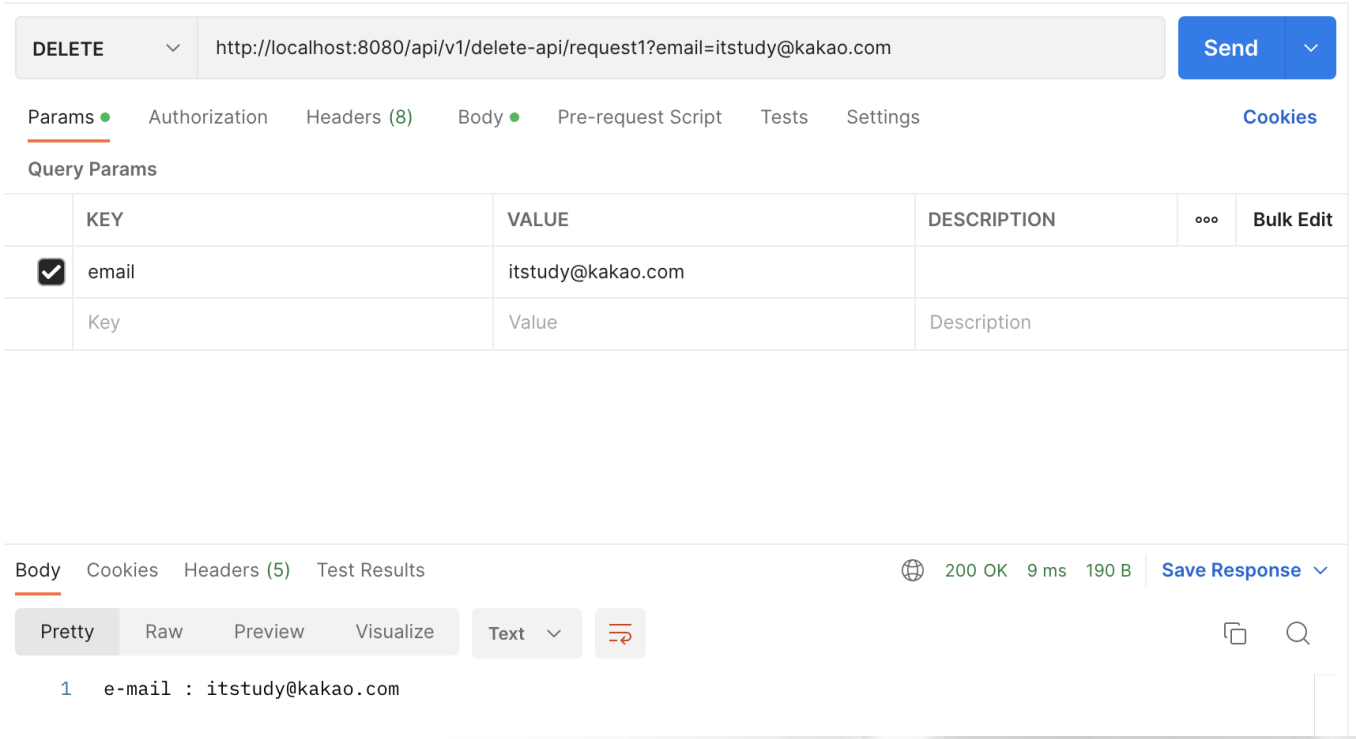
REST API

- DELETE API
  - ✓ DeleteController 클래스를 생성하고 메서드 추가



REST API

- DELETE API
  - ✓ DeleteController 클래스를 생성하고 메서드 추가



로그 출력

- 로깅 라이브러리
  - ✓ 로깅(logging)이란 애플리케이션이 동작하는 동안 시스템의 상태나 동작 정보를 시간순으로 기록하는 것을 의미
  - ✓ 로깅은 개발 영역 중 비기능 요구사항에 속하지만 로깅은 디버깅하거나 개발 이후 발생한 문제를 해결할 때 원인을 분석하는 데 꼭 필요한 요소
  - ✓ 자바 진영에서 가장 많이 사용되는 로깅 프레임워크는 Logback

로그 출력

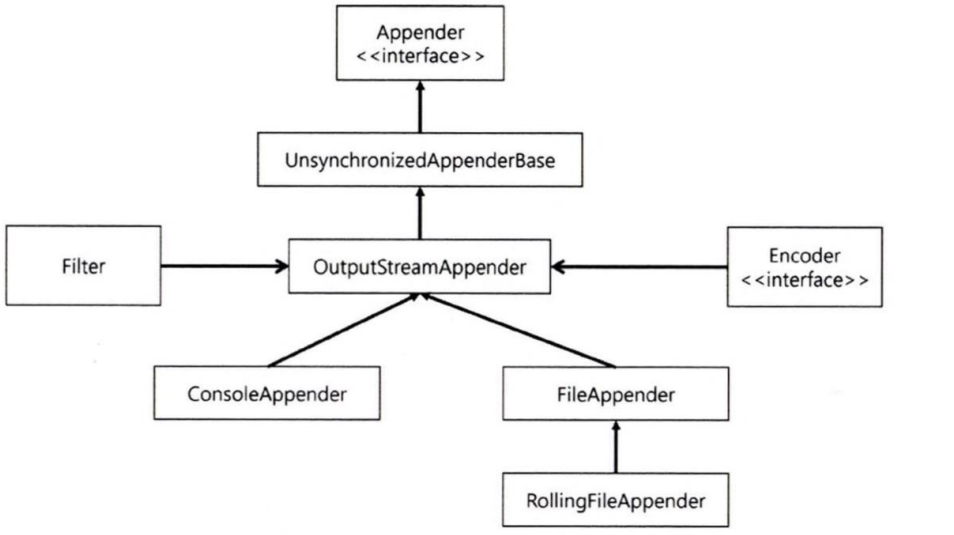
- Logback
  - ✓ log4j 이후에 출시된 로깅 프레임워크로서 slf4j를 기반으로 구현됐으며 과거에 사용되던 log4j에 비해 성능이 향상됨
  - ✓ 스프링 부트의 spring-boot-starter-web 라이브러리 내부에 내장돼 있어 별도의 의존성을 추가하지 않아도 사용할 수 있음
  - ✓ https://logback.qos.ch/manual/introduction.html
  - ✓ 특징
    - 5개의 로그 레벨(TRACE, DEBUG. INFO. WARN. ERROR}을 설정할 수 있음
      - ERROR: 심각한 문제가 발생해서 애플리케이션의 작동이 불가능한 경우를 의미
      - WARN: 시스템 에러의 원인이 될 수 있는 경고 레벨을 의미
      - INFO: 애플리케이션의 상태 변경과 같은 정보 전달을 위해 사용
      - DEBUG: 애플리케이션의 디버깅을 위한 메시지를 표시하는 레벨
      - TRACE: DEBUG 레벨보다 더 상세한 메시지를 표현하기 위한 레벨
    - 운영 환경 과 개발 환경에서 각각 다른 출력 레벨을 설정해서 로그 확인 가능
    - 설정 파일을 일정 시간마다 스캔해서 애플리케이션을 재가동하지 않아도 설정을 변경할 수 있음
    - 별도의 프로그램 지원 없이도 자체적으로 로그 파일을 압축할 수 있음
    - 저장된 로그 파일에 대한 보관 기간 등을 설정해서 관리할 수 있음

로그 출력

- Logback
  - ✓ 설정
    - 일반적으로 클래스패스(classpath)에 있는 설정 파일을 자동으로 참조하므로 Logback 설정 파일은 리소스 폴더 안에 생성
    - 파일명의 경우 일반적인 자바 또는 스프링 프로젝트에서는 logback.xml이라는 이름으로 참조하지만 스프링 부트에서는 logback-spring.xml 파일을 참조
    - 영역
      - Property 영역
      - Appender 영역
      - Encoder 영역
      - Pattern 영역
      - Root 영역

로그 출력

- ❖ Logback
  - ✓ 설정
    - ● Appender 영역
      - ◆ Appender 영역은 로그의 형태를 설정하고 어떤 방법으로 출력할지를 설정하는 곳
      - ◆ Appender 자체는 하나의 인터페이스를 의미하며 하위에 여러 구현체가 존재



로그 출력

- Logback
  - ✓ 설정
    - Appender 영역
      - ◆ 구현체 – class 속성을 이용해서 지정
        - ConsoleAppender: 콘솔에 로그를 출력
        - FileAppender: 파일에 로그를 저장
        - RollingFileAppender: 여러 개의 파일을 순회하면서 로그를 저장
        - SMTPAppender: 메일로 로그를 전송
        - DBAppender: 데이터베이스에 로그를 저장

로그 출력

- ❖ Logback
  - ✓ 설정
    - ● Appender 영역
      - ◆ encoder – 표현 형식을 패턴으로 정의
        - ▪ %Logger{length}: 로거 이름
        - ▪ %-5level: 로그 레벨로 -5는 출력 고정폭
        - ▪ %msg(%message): 로그 메시지
        - ▪ %d: 로그기록시간
        - ▪ %p: 로깅 레벨
        - ▪ %F: 로깅이 발생한 애플리케이션 파일명
        - ▪ %M: 로깅이 발생한 메서드 이름
        - ▪ %l: 로깅이 발생한 호출지의 정보
        - ▪ %c: 로깅이 발생한 클래스명
        - ▪ %m: 로그 메시지
        - ▪ %n: 줄바꿈
        - ▪ %r: 애플리케이션 실행 후 로깅이 발생한 시점까지의 시간
        - ▪ %L: 로깅이 발생한 호출 지점의 라인 수

로그 출력

- Logback
  - ✓ 설정
    - Root 영역
      - 설정 파일에 정의된 Appender를 활용하려면 Root 영역에서 Appender를 참조해서 로깅 레벨을 설정
      - 특정 패키지에 대해 다른 로깅 레벨을 설정하고 싶다면 root 대신 logger를 사용

로그 출력

- Logback
  - ✓ 적용
    - resources 디렉토리에 logback-spring.xml 파일을 생성하고 작성

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <property name="LOG_PATH" value="./logs"/>

    <!-- Appenders -->
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <encoder>
            <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%-5level]
    [%thread] %logger %msg%n</pattern>
        </encoder>
    </appender>
```

로그 출력

- Logback
  - ✓ 적용
    - resources 디렉토리에 logback-spring.xml 파일을 생성하고 작성

```
<appender name="INFO_LOG"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>INFO</level>
    </filter>
    <file>${LOG_PATH}/info.log</file>
    <append>true</append>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${LOG_PATH}/info_${type}.%d{yyyy-MM-
dd}.gz</fileNamePattern>
        <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder>
        <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%-5level]
[%thread] %logger %msg%n</pattern>
    </encoder>
</appender>
```

로그 출력

- Logback
  - ✓ 적용
    - resources 디렉토리에 logback-spring.xml 파일을 생성하고 작성

```
    <!-- TRACE > DEBUG > INFO > WARN > ERROR > OFF -->
    <!-- Root Logger -->
    <root level="INFO">
        <appender-ref ref="console"/>
        <appender-ref ref="INFO_LOG"/>
    </root>
</configuration>
```

로그 출력

- ❖ Logback
  - ✓ 적용
    - GETController 클래스 수정 한 후 재실행하고 GET 요청 후 콘솔 확인

```
@RestController
@RequestMapping("/api/v1/get-api")
@Api(value="ApiController v1")
public class GetController {
    private final Logger LOGGER = LoggerFactory.getLogger(GetController.class);

    // http://localhost:8080/api/v1/get-api/hello
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String getHello() {
        LOGGER.info("getHello 메소드가 호출되었습니다.");
        return "Hello World";
    }

    // http://localhost:8080/api/v1/get-api/variable1/{String 값}
    @GetMapping(value = "/variable1/{variable}")
    public String getVariable1(@PathVariable String variable) {
     LOGGER.info("@PathVariable을 통해 들어온 값 : {}", variable);
        return variable;
    }
}
```

개발자 도구

- spring-boot-devtool – 개발자 도구
  - ✓ 포함된 기능
    - 애플리케이션 restart 와 reload 자동화
    - 환경 설정 정보 기본값 제공
    - 자동 설정 변경사항 로깅
      - application.properties 파일에 loggin.level.web=DEBUG를 설정하면 로그 출력
    - 정적 자원 제외
      - /META-INF/maven
      - /META-INF/resources
      - /resources
      - /static
      - /public
      - /templates
    - Live Reload 지원