섹션 1. 생애 최초 API 만들기

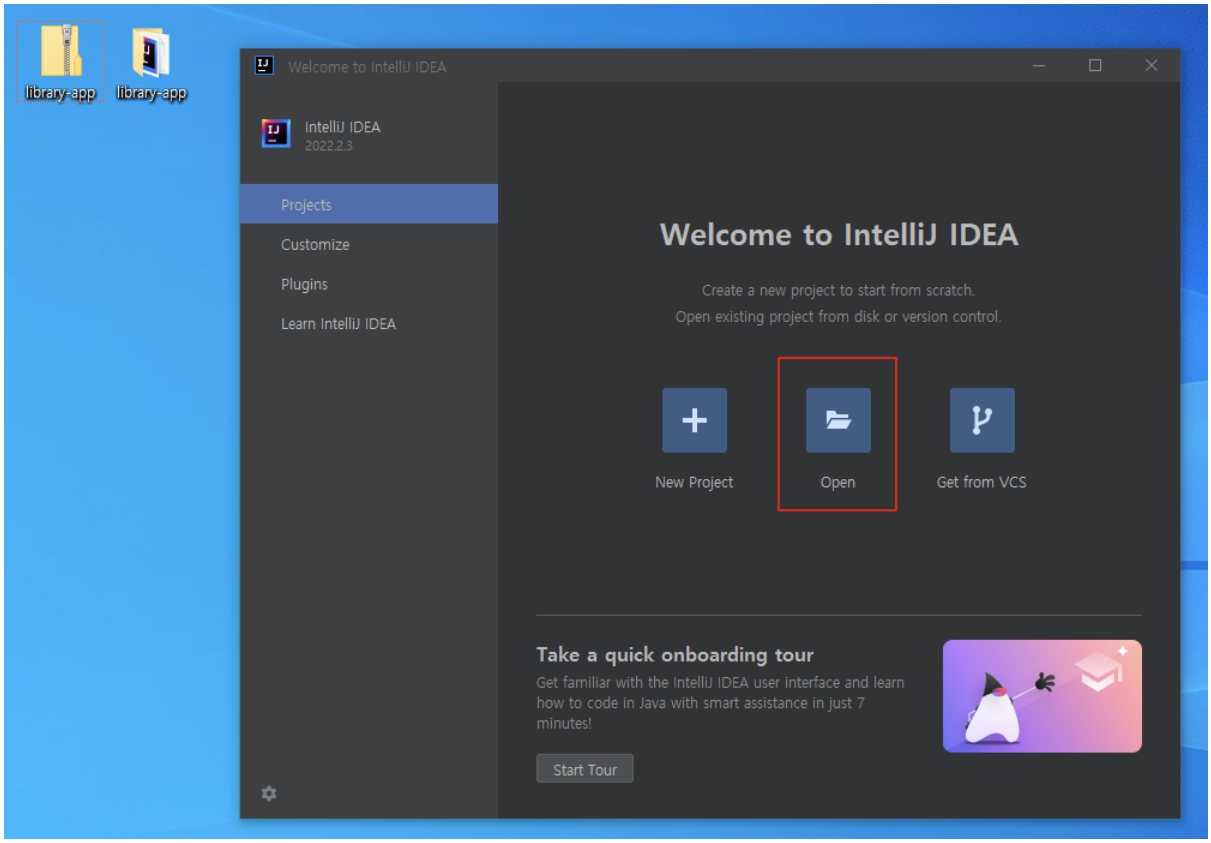
이번 섹션의 목표

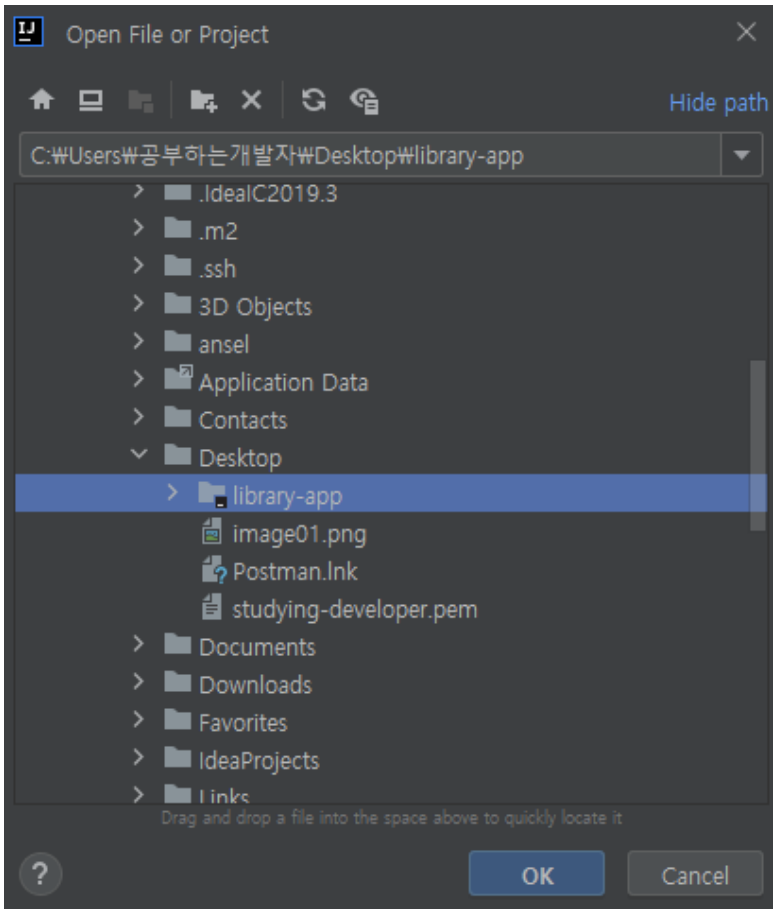
1. 스프링 부트 프로젝트를 설정해 시작하고 실행할 수 있다.
2. 서버란 무엇인지, 네트워크와 HTTP, API는 무엇인지, JSON은 무엇인지 등 서버 개발에 필요한 다양한 개념을 이해한다.
3. 스프링 부트를 이용해 간단한 GET API / POST API를 만들어 본다.

수업 소스코드 준비하기

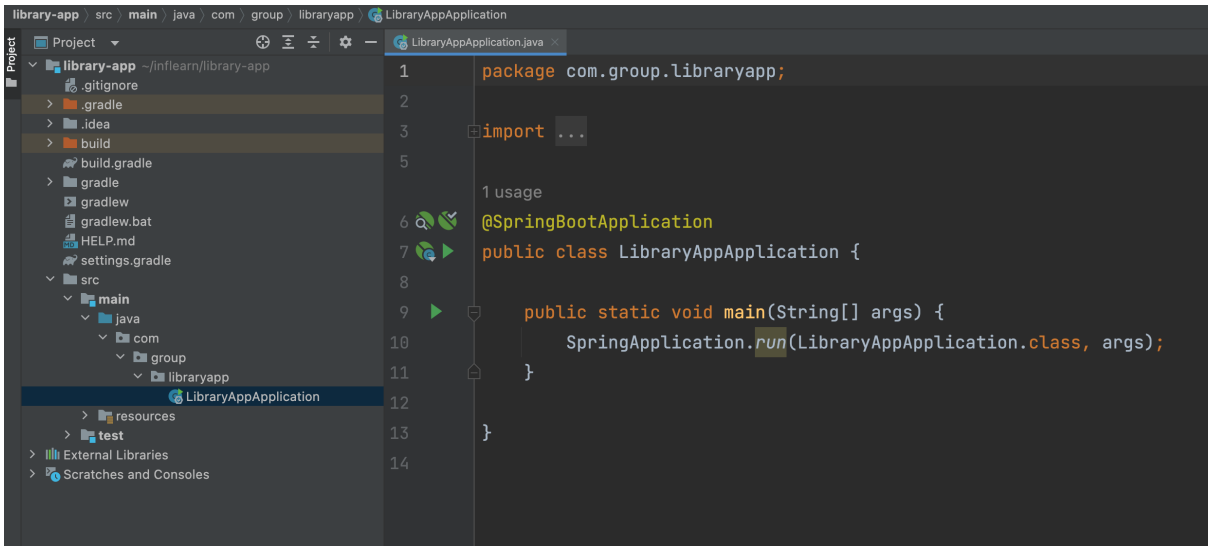
인프런 수업자료 압축 파일을 다운로드 받아보면, 강의 때 사용된 PDF, PPT 뿐만 아니라 웹 UI와 기본적인 설정이 모두 갖춰진 스프링 프로젝트가 들어있다.

이 스프링 프로젝트를 다시 한 번 압축해제하면 `library-app` 이라는 폴더가 나온다! 이 폴더를 적당한 위치에 옮기고, IntelliJ를 열어 이 프로젝트를 열어 주면 된다.


library-app
library-app
Welcome to IntelliJ IDEA
IntelliJ IDEA
2022.2.3
Projects
Customize
Plugins
Learn IntelliJ IDEA
Welcome to IntelliJ IDEA
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
Take a quick onboarding tour
Get familiar with the IntelliJ IDEA user interface and learn how to code in Java with smart assistance in just 7 minutes!
Start Tour


Open File or Project
Hide path
C:₩Users₩공부하는개발자₩Desktop₩library-app
.IdeaIC2019.3
.m2
.ssh
3D Objects
ansel
Application Data
Contacts
Desktop
library-app
image01.png
Postman.lnk
studying-developer.pem
Documents
Downloads
Favorites
IdeaProjects
Links
Drag and drop a file into the space above to quickly locate it
OK
Cancel

IntelliJ에서 해당 프로젝트를 연 이후에는 최초로 필요한 이것저것을 다운로드를 받게 된다. 그 후 왼쪽에 있는 폴더 구조를 따라 LibraryAppApplication class를 열어주자.



이제 `public class LibraryApplication` 옆에 있는 초록색 세모 버튼을 눌러 Run을 고르면, 다음과 같이 `Started LibraryAppApplication in x seconds` 이라는 멘트가 나온다!

매우 좋다~!! 👍

이제 이 프로젝트를 이용해 본 강의 내용을 학습하게 된다!

1강. 스프링 프로젝트를 시작하는 두 번째 방법

> 💡 강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

우리는 이 날을 위해 기초적인 Java 문법을 모두 마스터하였다!! 자 가보자~!!! 🔥 😊

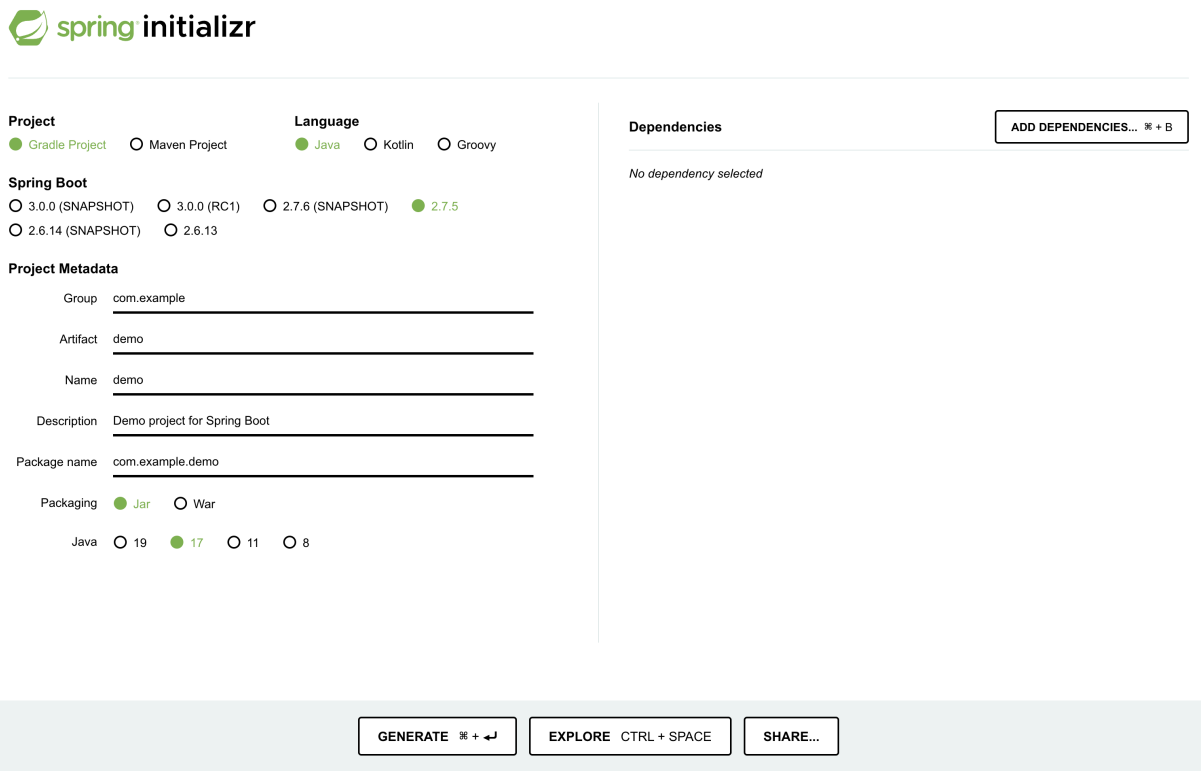
생애 최초 서버 만들기를 위해 알아야 할 지식들은 아직 매우 많지만, 우선 그 시작으로 스프링 프로젝트를 시작해 볼 것이다. 스프링 프로젝트를 어떻게 시작할 수 있을까?! 2가지 방법을 통해 알아보자.

첫 번째 방법은 이미 만들어져 있는 스프링 프로젝트를 다운로드 받아 IntelliJ를 통해 여는 것이다. 이 방법은 **<수업 소스코드 준비하기>**에서 살펴보았다.

두 번째 방법도 알아보자. 본 강의에서 활용할 방법은 아니지만 포트폴리오를 만들거나, 새로운 팀 프로젝트를 할 때 필요한 순간도 있으니 소개한다. 두 번째 방법은, 아무것도 없는 상태에서 새로 스프링 프로젝트를 시작하는 방법이다. 우선 아래 사이트에 접속하자.

- https://start.spring.io/

사이트에 접속하면, 아래와 같은 화면이 나오게 된다.



이 화면에서 나오는 요소 하나하나를 알아보자.

- Project - Maven Project / Gradle Project
  - 이 프로젝트에서 사용될 '빌드 툴'을 고르는 항목이다.
  - Gradle Project를 골라 진행하자.
- Language - Java / Kotlin / Groovy
  - 서버를 개발할 때 사용할 언어를 고르는 항목이다.
  - 요즘에는 최신 프로젝트들을 Kotlin으로 만드는 경향이 있지만, Java 역시 진작 만들어진 프로젝트가 무척 많고 Java를 먼저 사용해본 이후 Kotlin을 배우는 것이 효과적이기 때문에 본 강의에서는 Java로 진행할 것이다.
  - Java를 골라 진행하자.
- Spring Boot

    - 스프링 부트의 버전을 고르는 항목이다.
    - 옆에 알파벳이 붙지 않은, 가장 최신 버전을 골라 주자.
    - 알파벳이 붙어있다는 의미는 아직 개발중이거나 테스트 중이라 안정성이 떨어질 수 있다.
    - 버전은 시간이 지나면서 계속 변화하기 때문에, 숫자가 조금 다를 수 있다!
- Project Metadata
    - 우리의 프로젝트를 설정하는 항목이다. 다양한 이름을 짓는 항목이라고 생각하면 된다. 도서 관리 애플리케이션을 만든 다면 다음과 같이 적절히 바꿀 수 있을 것이다.
    - Group : com.group
    - Artifact : library-app
    - Name : library-app
    - Description : library application
    - Package name : com.group.libraryapp
    - Packaging : Jar
        - Spring Boot는 톰캣이 내장되어 있어 Jar을 설정하면 된다. 강의가 진행됨에 따라 톰캣이 무엇인지~ 내장되어 있다는 의미가 무엇인지~ 등을 모두 설명드릴 예정이다.
    - Java : 11버전
        - 2023년 1월 기준 가장 많이 사용되는 11버전을 골라주자.
        - 8버전 → 11버전 → 17버전을 거치며 변화들이 있었지만, 우리가 만들 애플리케이션에서는 새로운 기능들을 사용할 필요가 없어 큰 상관이 없다. 그러니 가장 많이 사용되는 버전을 골라주자.
- Dependencies
    - 한국어로 직역하면 의존성이라는 의미이다.
    - 의존성이란, 프로젝트에서 사용하는 라이브러리나 프레임워크를 의미한다.
    - 라이브러리 : 프로그래밍을 개발할 때 미리 만들어져 있는 기능을 가져다 사용하는 것
        - 프로그래밍을 요리로 비유한다면, 김치찌개를 만들 때 김치를 마트에서 사서 만들 수도 있고, 배추 농사부터 지을 수도 있다. 전자의 경우가 라이브러리를 활

용하는 경우이고 김치가 '라이브러리'이다.

- 프레임워크 : 프로그래밍을 개발할 때 미리 만들어져 있는 구조에 코드를 가져다 끼워 넣는 것
  - 프로그래밍을 요리로 비유한다면, 김치찌개를 만들 때 여러 재료를 사서 만들 수도 있고, 김치찌개 원데이 클래스에 가서 선생님이 시키는 것만 편하게 할 수도 있다. 후자의 경우가 프레임워크를 활용하는 경우이다.

매우 좋다~ 이렇게 설정을 모두 마쳤으면 Generate를 눌러 프로젝트를 만들어주자. 그러면, 다음과 같이 압축 파일이 다운로드 된다. 이제 첫 번째 방법과 똑같이 다운로드된 압축 파일을 해제하고 적절한 위치로 옮겨, IntelliJ로 열어주면 된다! 그 후 역시 첫 번째 방법과 동일하다.

이제 모든 준비가 끝났다! 다음 시간에는 도대체 이 코드들이 무슨 의미인지, 그래서 서버는 무엇이고 어떻게 만드는 건지 알아보도록 하자. 😊

2강. @SpringBootApplication과 서버

> 💡 강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

이번 시간에는 **<수업 소스코드 준비하기>**에서 실행시켜보았던, 코드를 한 줄 한 줄 이해해보고~ '서버'란 무엇인지 알아보자.

```
@SpringBootApplication
public class LibraryAppApplication {

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

}
```

눈에 보이는 가장 첫 줄 `@SpringBootApplication` 은 **어노테이션**이다. 어노테이션은 Java의 문법으로 `@` 뒤에 이런저런 문자열을 붙이는 것이다. 종류도 매우 다양하다.

주석이 아닌 모든 코드는 그마다의 역할이 있다. `System.out.println()` 은 출력을 해주고 `1 + 1` 은 덧셈을 해준다. 그렇다면 어노테이션이 해주는 일은 무엇일까?!

Java에서 어노테이션 역시 **마법같은 일**을 해주는 기호이다. 이 마법같은 일은 어노테이션 종류에 따라 모두 다르다. 예를 들어 여기 있는 `@SpringBootApplication` 은 스프링을 실행시키기 위해 필요한 다양한 설정들을 모두 자동으로 해준다.

앞으로 계속해서 다양한 어노테이션이 나올 예정이다.

다음 줄은 이제 class이다. Java에서 모든 코드는 class 안에 있어야 하고, 실제로 스태틱 메소드인 main 메소드가 이 안에 존재한다.

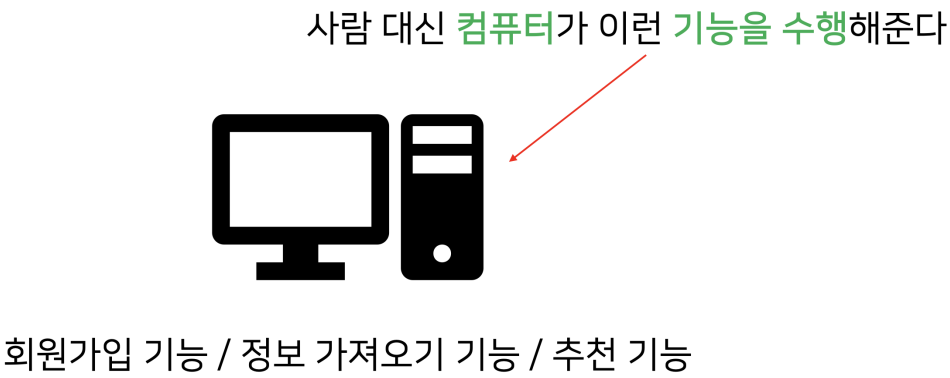
```
public class LibraryAppApplication {
  public static void main(String[] args) {

  }
}
```

마지막 줄은 `SpringApplication.run(LibraryAppApplication.class, args);` 이다.

영어의 의미를 살펴보면, 스프링 애플리케이션을 실행(run)한다라는 의미이다. 그렇다.

이 코드가 바로 우리의 서버를 실행시키는 코드이다. 앞서 눌렀던 초록색 버튼은 이 main 메소드를 실행시켜, 서버를 실행시킨 것이다. 크으~ 우리는 처음으로 서버라는 것을 실행시켰다.

잠깐, 방금 서버라고 했는데 도대체 서버라는게 무엇일까?



서버

- 어떠한 **기능을 제공**하는 프로그램
- 그 프로그램을 실행시키고 있는 **컴퓨터**

기능을 제공하기 위해서는 누군가 요청을 해야 한다. 예를 들어, 택시라는 기능을 제공 받으려면, 길거리에서 손을 흔들거나 콜택시를 불러야 하고, 진료라는 기능을 받으려면 병원에서 가서 진료 요청을 해야 한다.

그렇다면 서버는 컴퓨터라고 했는데, 컴퓨터에는 어떻게 요청을 보낼까?!

그 방법이 바로 **인터넷**이다!!

이제 다음 시간에는 인터넷, 네트워크 등에 대해 알아보자~~!! 🌐 💻

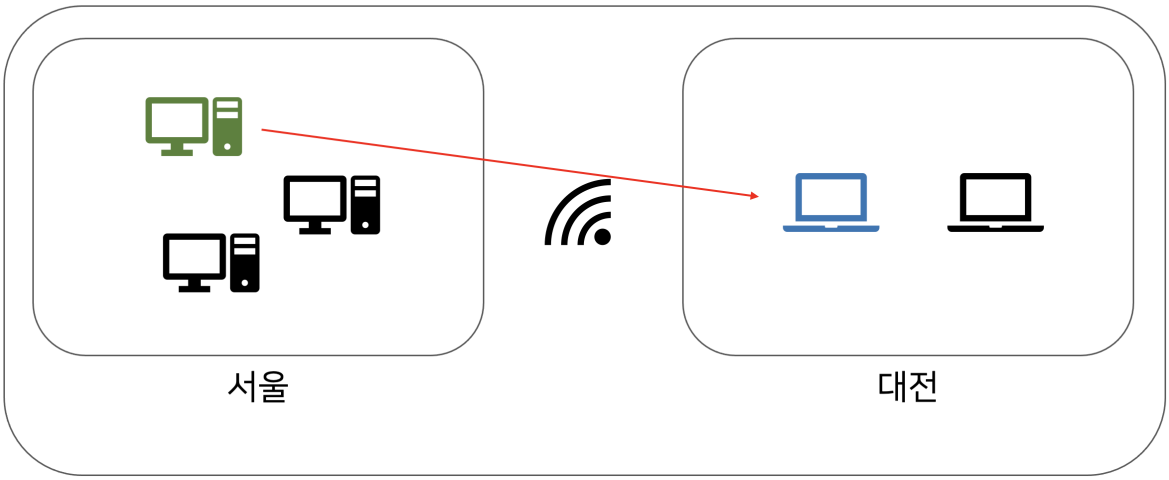
3강. 네트워크란 무엇인가?!

> 💡 강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

강의 영상에서 다음과 같은 내용을 다루고 있다.

- 네트워크
- IP
- DNS
- port

또한 인터넷을 통해 데이터를 주고 받을 수 있습니다.



이세계	현실 세계
택배 시스템	네트워크
집	컴퓨터
주소 / B부족 감자동 곰로 13번길 2	IP / 244.77.51.9
주소 별칭 / 파란집	도메인 이름 / spring.com
택배를 정말 받는 사람 / 둘째	port / 3000

4강. HTTP와 API란 무엇인가?!

> 💡 강의 영상과 PPT를 함께 보시면 **(특히)** 더욱 좋습니다 😊

강의 영상에서 다음과 같은 내용을 다루고 있다.

- HTTP
- HTTP Method
- Path
- Query와 Body
- HTTP Header와 Body

- HTTP status code
- API
- URL
- Client와 Server

스크린샷을 모두 PDF에 담기는 어려워 간단한 요약만 정리해둔다.

1. (웹을 통한) 컴퓨터 간의 통신은 HTTP 라는 표준화된 방식이 있다.
2. HTTP 요청은 HTTP Method (GET, POST)와 Path(/portion)가 핵심이다.
3. 요청에서 데이터를 전달하기 위한 2가지 방법은 쿼리와 바디이다.
4. HTTP 응답은 상태 코드가 핵심이다.
5. 클라이언트와 서버는 HTTP를 주고 받으며 기능을 동작하는데 이때 정해진 규칙을 API 라고 한다.

매우 좋다~ 이제 API를 개발하기 위한 네트워크 이론 지식은 충분히 갖추었다!!

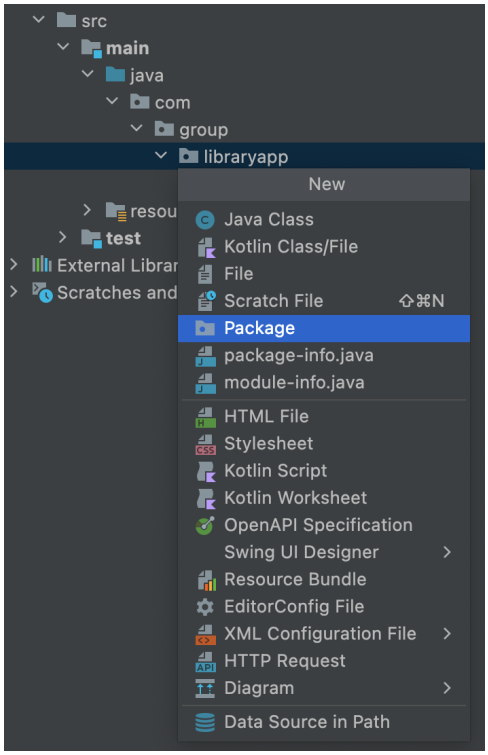
다음 시간에는 연습용 API를 개발하고 테스트해 보자.

5강. GET API 개발하고 테스트하기

> 💡 강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

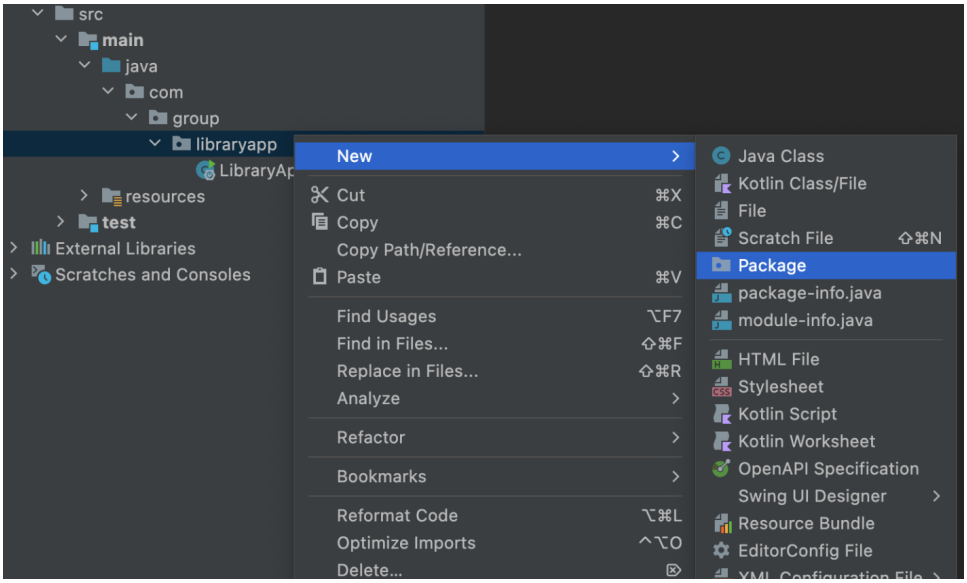
이번 시간에는 Java와 Spring Boot를 이용해 첫 API를 개발하고 테스트해 볼 예정이다.

가장 먼저 Controller를 담을 **패키지**를 만들어 주자!

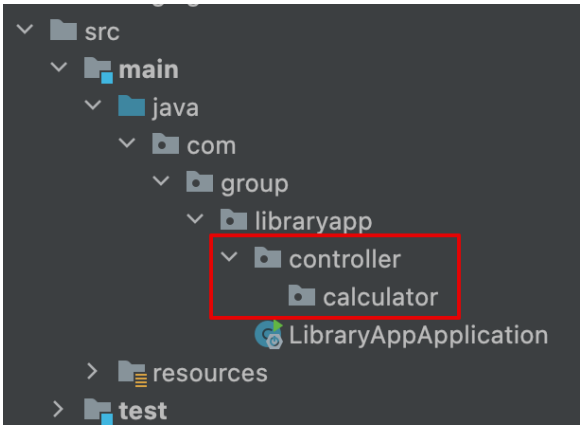


패키지를 만드는 첫 번째 방법은 libraryapp 패키지를 클릭하고 `command + N (윈도우/리눅스는 ctrl + N)` 을 누른 후 `Package` 를 골라, 패키지 이름을 입력해주는 것이다.

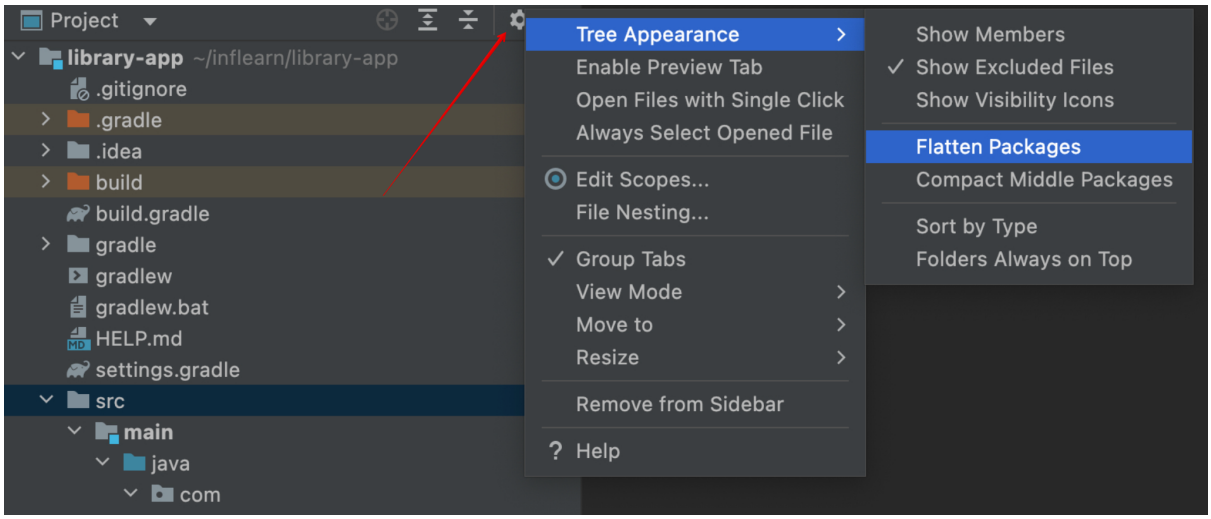
두 번째 방법은 libraryapp 패키지에 커서를 놓고 `우클릭 > New > Package` 를 골라, 패키지 이름을 입력하는 것이다.



둘 중 어떤 방법을 사용하건 상관없다. `controller` 패키지를 만들고, 그 안에 `calculator` 패키지를 만들면 아래 이미지처럼 패키지가 생기게 된다.



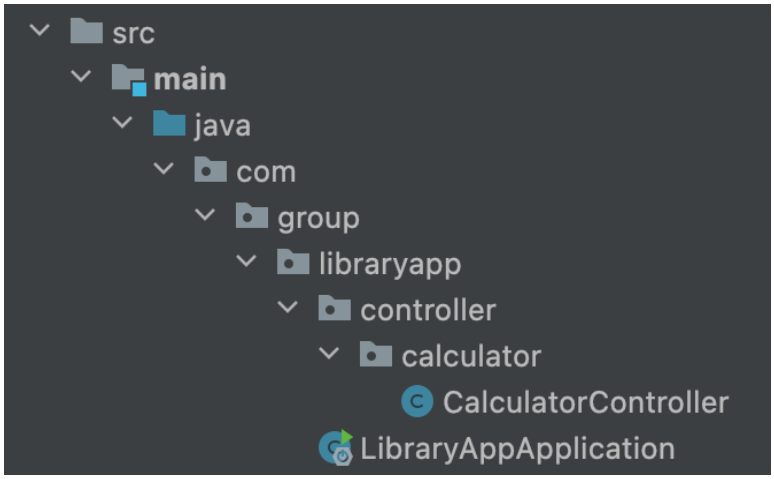
혹시나 패키지 모양이 이미지와 완전히 동일하지 않고
`com.group.libraryapp.controller.calculator` 처럼 합쳐져 있다면



왼쪽 탭 오른쪽 상단의 `톱니바퀴 > Tree Apperance > Flatten Packages` 와 `Compact Middle Packages` 를 눌러보자!

이 옵션에 따라 패키지를 IDE에서 보여주는 모습이 바뀌게 된다~ 👍

좋다~ 이제 만들었던 패키지 안에 CalculatorController라고 클래스를 만들어주자. 클래스 역시 패키지와 비슷하게 만들되 `Package` 대신 `Java Class` 를 선택하면 된다.




```
public class CalculatorController {
}
```

매우 좋다~~ 👍 이제 본격적으로 API를 개발해볼 것이다. 우리가 개발할 API는 바로 **덧셈 API**이다. 두 수를 받아, 두 수의 덧셈 결과를 돌려줄 것이다.

API 코드를 작성하기 전에 잠시 API가 무엇이었는지, API를 이루고 있는 요소가 무엇인지 기억을 떠올려보자.

API를 이루고 있는 요소

```
GET /portion?color=red&count=2
Host: spring.com:3000
```

HTTP Method

HTTP Path

쿼리 (key와 value)

API의 반환 결과

그렇다. API를 개발하기 전에는 API의 HTTP method를 어떻게 할 것인지, path는 어떻게 할 것인지, 쿼리를 사용할 것인지 바디를 사용할 것인지, 쿼리를 사용한다면 어떤 key를 이용할 것인지, 결과는 어떤 형태로 줄 것인지를 고민해서 결정해야 한다.

이러한 것을 `API Specification(명세)` 줄여서 `API 스펙`이라고도 한다.

보통 이러한 스펙을 클라이언트 개발 담당자와 함께 정하게 된다. 현재는 연습이니 바로 정해보자.

- HTTP Method : GET
- HTTP Path : /add

- 쿼리 사용
  - int number1
  - int number2
- 결과 반환
  - int 쿼리로 들어온 두 숫자의 합

매우 좋다~ 👍 이제 정한 스펙에 맞춰 개발을 진행해 보자.

```
@RestController
public class CalculatorController {

  @GetMapping("/add")
  public int addTwoNumbers(
      @RequestParam int number1,
      @RequestParam int number2
  ) {
    return number1 + number2;
  }

}
```

- `@RestController`
  - 주어진 Class를 Controller로 등록한다. (Controller : API의 진입 지점)
- `@GetMapping("/add")`
  - 아래 함수를 HTTP Method가 GET이고 HTTP path가 /add인 API로 지정한다.
- `@RequestParam`
  - 주어지는 쿼리를 함수 파라미터에 넣는다.

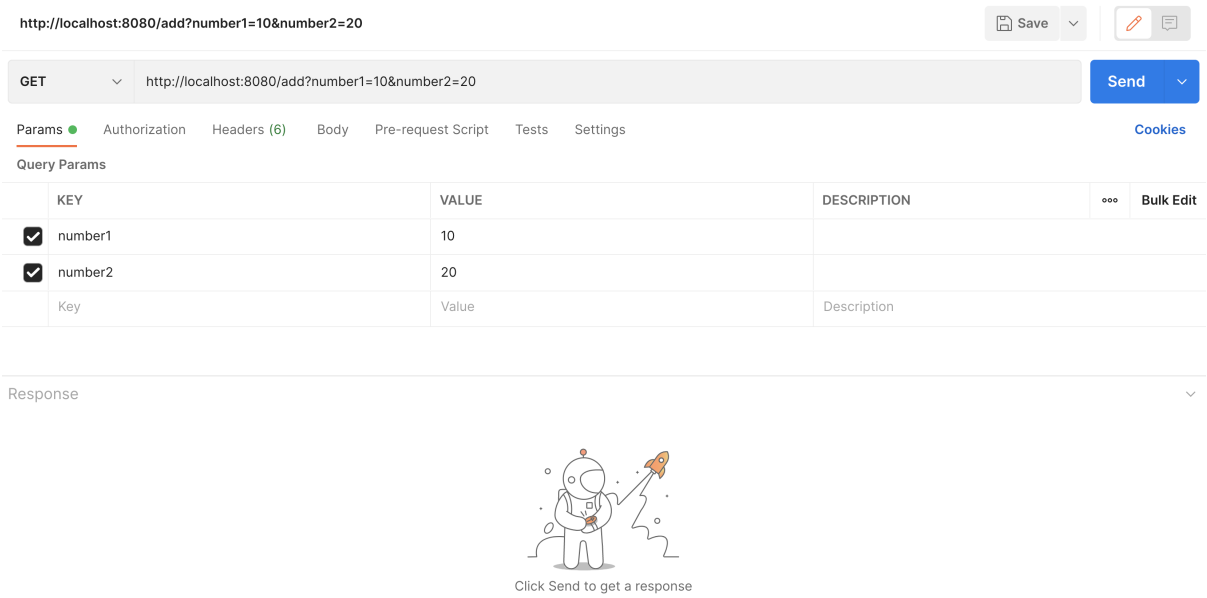
GET /add?number1=10&number2=20

같은 이름을 가진
쿼리의 값이 들어온다.

```java
@RestController
public class CalculatorController {

  @GetMapping("/add")
  public int addTwoNumbers(
      @RequestParam int number1,
      @RequestParam int number2
  ) {
    return number1 + number2;
  }

}
```

우리가 개발한 API가 잘 동작하는지 확인해 보자. API 테스트는 강의 준비 영상에서 다운로드 받았던 POST MAN을 사용할 수 있다.

POST MAN에 다음과 같이 URL을 입력해주도록 하자



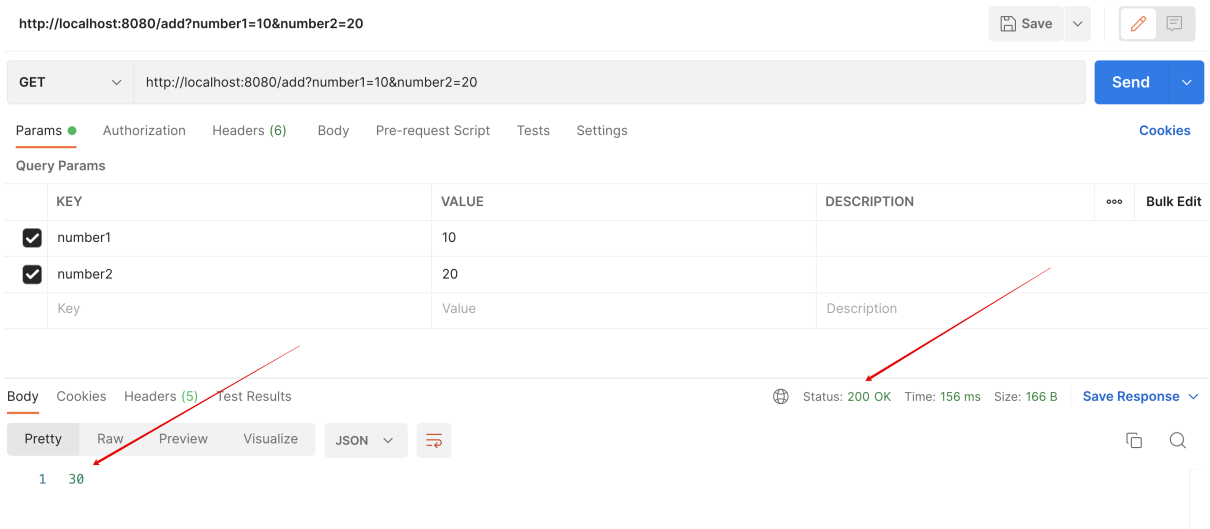
- GET API를 만들었으므로 GET으로 설정하였다.
- HTTP를 사용했기 때문에 URL은 `http`로 시작한다.
- API를 요청하려는 서버가 우리의 컴퓨터에 있기 때문에 `localhost`라고 입력해준다.
- 스프링 부트의 기본 port는 `8080`이다.
  - 서버를 실행시켰을 때 `Tomcat started on port(s): 8080 (http) with context path ''`라는 문구를 확인할 수 있다.

- HTTP path가 `/add` 이므로 입력해준다.
- 쿼리로 서버에 보내줄 `number1`에는 `10`, `number2`에는 `20`을 넣어준다.
- 중간 중간 `://` `:` `/` `?` `=` `&` 와 같은 기호들을 잘 넣어주어야 한다!

모두 조합하면 `http://localhost:8080/add?number1=10&number2=20` 가 된다.

그 다음 우리의 서버를 동작시켜주자! 서버를 키기 위해서는 1강에서 했던 것처럼, `LibraryAppApplication` 옆에 있는 세모 버튼을 누르면 된다.

이제 PostMan에 있는 `Send`를 눌러주면..!!



매우 좋다~!! 😊 다음과 같이 `10+20` 의 결과인 30이 나오는 것을 확인할 수 있다!

오른쪽에는 성공했다는 의미인 HTTP 상태코드 `200 OK`가 표시되고 있다.

우리는 GET API를 성공적으로 만들었다~~~!!!! 🎉🎉

여기서 한 발자국 더 전진해 보자!

현재는 쿼리에 들어오는 값이 number1과 number2로 2개이지만, API가 복잡해지면, @RequestParam이 매우 많아질 수도 있다. 즉, Controller에 있는 함수의 파라미터가 많아져야 한다는 뜻이다.

일반적으로 함수의 파라미터가 많아지면 코드가 길어져 깔끔하지 못하고, 실수할 여지가 많아진다. 어떻게 하면 쿼리가 많아지더라도, 파라미터 개수를 줄일 수 있을까?!

이럴 때 바로 쿼리를 받는 Class를 만들 수 있다!

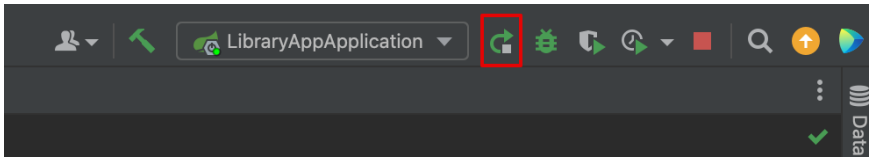
`com.group.libraryapp` 패키지 안에 `dto.calculator.request`를 만들어주고 number1과 number2를 가지고 있는 클래스를 만들어 보자!

```
public class CalculatorAddRequest {

  private final int number1;

  private final int number2;

  public CalculatorAddRequest(int number1, int number2) {
    this.number1 = number1;
    this.number2 = number2;
  }

  public int getNumber1() {
    return number1;
  }

  public int getNumber2() {
    return number2;
  }

}
```

이제 Controller 코드에서도 이 클래스를 사용하게 바꿔주자!! `@RequestParam` 어노테이션은 제거해도 괜찮다!

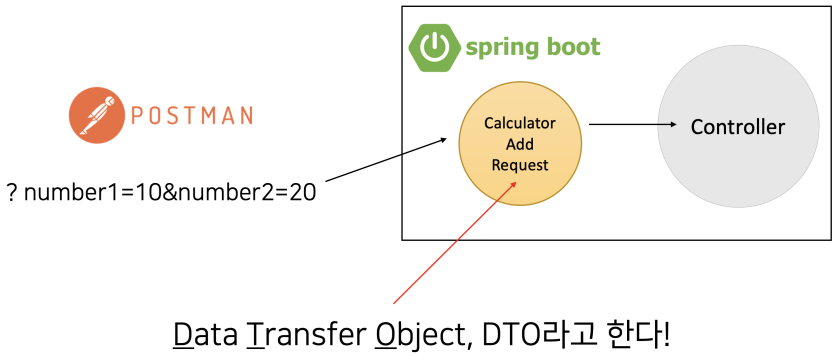
```
@RestController
public class CalculatorController {

  @GetMapping("/add")
  public int addTwoNumbers(CalculatorAddRequest request) {
    return request.getNumber1() + request.getNumber2();
  }

}
```

그 다음 서버를 재시작하고 다시 POST MAN을 실행해 보자! IntelliJ 오른쪽 상단에 다시시작 버튼도 있다!



아까 입력했던 URL로 POST MAN Send를 눌러보면, 이번에도 동일하게 30이라는 결과가 나오는 것을 확인할 수 있다~! 이제 쿼리의 개수가 늘어나더라도 깔끔한 코드를 작성할 수 있다.

방금사용했던 `CalculatorAddRequest` 는 '쿼리' 라는 데이터를 외부에서 서버 안 Controller로 전달하는 역할을 맡았다. 이런 객체를 **Data Transfer Object**, 줄여서 **DTO**라고 부른다.



매우 좋다~ 😄 이제 다음 시간에는 곱셈 기능을 가진 POST API를 만들어 보자!

6강. POST API 개발하고 테스트하기

> 💡 강의 영상과 PPT를 함께 보시면 더욱 좋습니다 😊

이번 시간에는 지난 시간에 이어, 곱셈 기능을 가진 API를 만들어볼 것이다. 저번에 GET 연습을 해봤기 때문에 이번에는 POST API를 만들어 보려고 한다.

자 그러면 지난 시간에 했던것처럼 빠르게 API 스펙부터 정해보려 하는데! POST API 경우는 쿼리를 사용하지 않고 바디(body)를 사용한다.

그렇다면 이 body에는 데이터를 어떻게 담아주는 걸까?! 바로 `JSON`을 사용한다!

JSON은 JavaScript Object Notation의 약자로, 웹 통신에서 객체를 표기하는 기법이다.

'객체를 표기하는 기법'의 뜻을 풀어보면, 무언가를 표현하기 위한 형식이라는 의미이다.

예를 들어 사람이 있다고 해보자! 사람에게는 이름도 있고 나이도 있다. 이런 사람을 JSON으로 표현하면 다음과 같다.

```
{
  "name": "최태현",
  "age": 99
}
```

이때 이름 / 나이와 같은 각각의 속성은 쉼표(`,`)를 이용해 구분한다.

Java로 비유하면 `Map<Object, Object>`와 유사하다.

위의 사람을 에시로 들면, key가 `name`이고 value가 `최태현`인 값이 하나 들어 있고, key가 `age`이고 value가 `99`인 값이 또 하나 들어 있는 것이다.

여기서 포인트는 `Object`라는 것이다. 즉, 다양한 값들이 value에 들어갈 수 있다!

예를 들어, 어떤 사람이 강아지들을 키운다고 해보자! 그렇다면 JSON은 다음과 같이 바뀌게 될 것이다.

```
{
  "name": "최태현",
  "age": 99,
  "dogs": ["코코", "초코"]
}
```

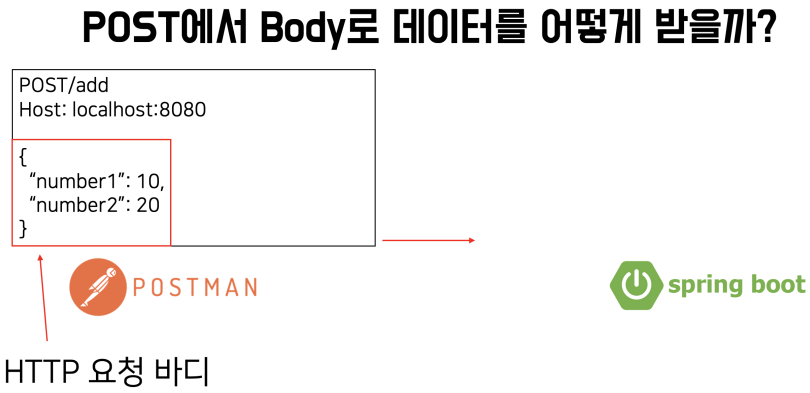
key가 `dogs` 이고 value에는 `List<String>` 이라 할 수 있는 `["코코", "초코"]`가 들어가 있다.

만약 집 정보를 추가하고 싶다면?

```
{
  "name": "최태현",
  "age": 99,
  "house": {
    "address": "대한민국 서울",
    "hasDoor": true
  }
}
```

다음과 같이 다시 한번 JSON 형식이 들어갈 수도 있다!! 정말 `Object` 가 들어간 셈이다.

좋다~ 이제 JSON이 무엇인지는 간략히 알았다! 이 JSON이 HTTP Body에 담기게 될 것이다!



매우 좋다~~ 😊 그럼 이제 API의 스펙을 정하고, 실제 코딩을 하러 가보자~!! 🔥

- HTTP Method : POST
- HTTP Path : /multiply
- HTTP Body (JSON)

```
{
  "number1": 숫자,
  "number2": 숫자
}
```

- 결과 반환
  - 곱셈 결과, 숫자 타입

지난 시간에 만들었던 `CalculatorController` 클래스에 다음의 함수를 만들어주자!

Controller는 API의 진입 지점으로 한 Controller Class에 여러 API를 만들어 줄 수 있다~! 👍

```
@PostMapping("/multiply")
public int multiplyTwoNumbers(
  @RequestBody CalculatorMultiplyRequest request
) {
  return request.getNumber1() * request.getNumber2();
}
```

```
public class CalculatorMultiplyRequest {

  private final int number1;

  private final int number2;

  public CalculatorMultiplyRequest(int number1, int number2) {
    this.number1 = number1;
    this.number2 = number2;
  }

  public int getNumber1() {
    return number1;
  }

  public int getNumber2() {
    return number2;
  }

}
```

- `@PostMapping("/multiply")`
  - 아래 함수를 HTTP Method가 POST이고 Path가 /multiply인 API로 지정한다.
- `@RequestBody`
  - HTTP Body로 들어오는 JSON을 `CalculatorMultiplyRequest`로 바꿔준다!
- `CalculatorMultiplyRequest`
  - HTTP body를 객체로 바꾸는 `@RequestBody` 를 사용하는 경우는, 생성자를 만들지 않아도 괜찮다.
  - 즉 다음의 코드로 변경하더라도 정상 동작한다.

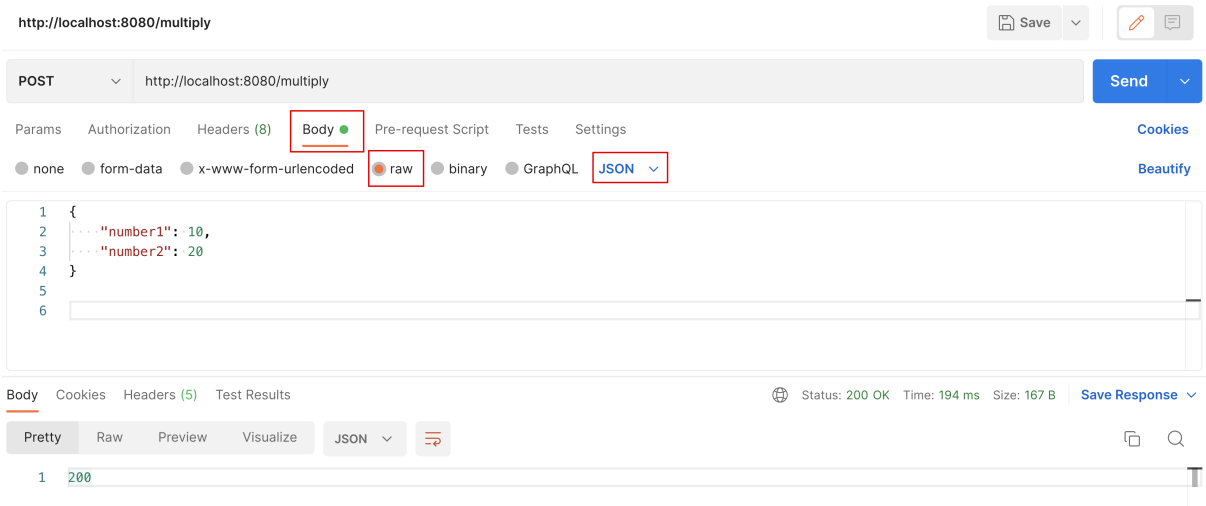
```
public class CalculatorMultiplyRequest {

  private int number1;

  private int number2;

  public int getNumber1() {
    return number1;
  }

  public int getNumber2() {
    return number2;
  }

}
```

매우 좋다~ 이번에도 DTO를 사용했다! 주의할 점으로는 POST API의 경우에는 Body를 통해 데이터가 들어오기 때문에 `@ReqestBody`를 꼭 사용해주어야 한다는 점이다.

이제 POST MAN을 이용해 테스트 해보자! 서버를 다시 한번 시작시켜 주고, POST MAN에 다음과 같이 입력해주자! 이번에는 쿼리가 아니라 바디에 두 숫자를 넣어주어야 한다!!

- URL : `http://localhost:8080/multiply`
- Body > raw > JSON 선택 후 JSON 입력

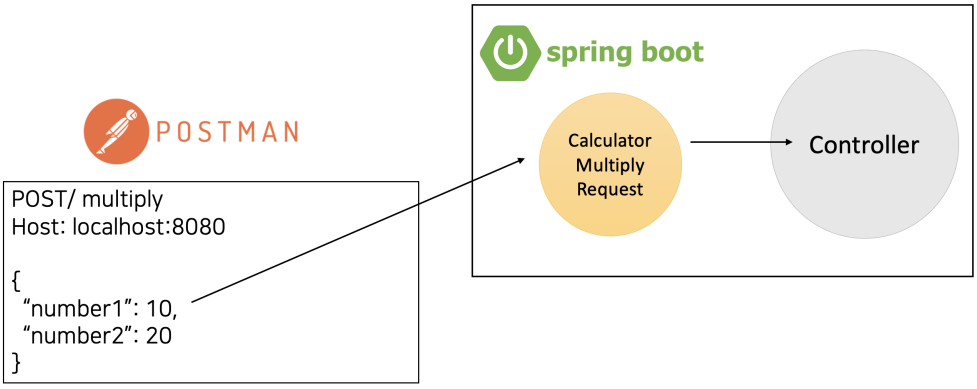
```
{
  "number1": 10,
  "number2": 20
}
```



좋다~!! 테스트 결과 10 * 20 = 200 이라는 결과가 정상적으로 나왔다!! 😊

HTTP Body에 있던 JSON이 `@RequestBody`를 통해 `CalculatorMultiplyRequest`에 매핑되고, Controller로 들어가 최종 결과를 반환해준 것이다!

HTTP Body는 CalculatorMultiplyRequest에 매핑된다!



추가로, 원래 POST API의 의미는 어떤 값을 저장한다는 의미이기 때문에 곱셈 기능을 하는 경우는 POST API를 작성하면 안 된다! 하지만 현재는 'HTTP 바디' 사용 방법을 연습하기 위해 POST로 만들어본 것이다!

매우 좋다~!! 🎉🎉 이제 다음 시간에는 진정한 POST API이자 도서 관리 애플리케이션의 첫 API인 유저 생성 API 개발을 해보자!

7강. 유저 생성 API 개발

이번 시간에는 유저 생성 API를 개발해 볼 예정이다! 이 API를 본격적으로 개발하기 전에 우리가 만들려고 하는 <도서 관리 애플리케이션>이 어떤 요구사항을 가지고 있는 살펴보자!

사용자

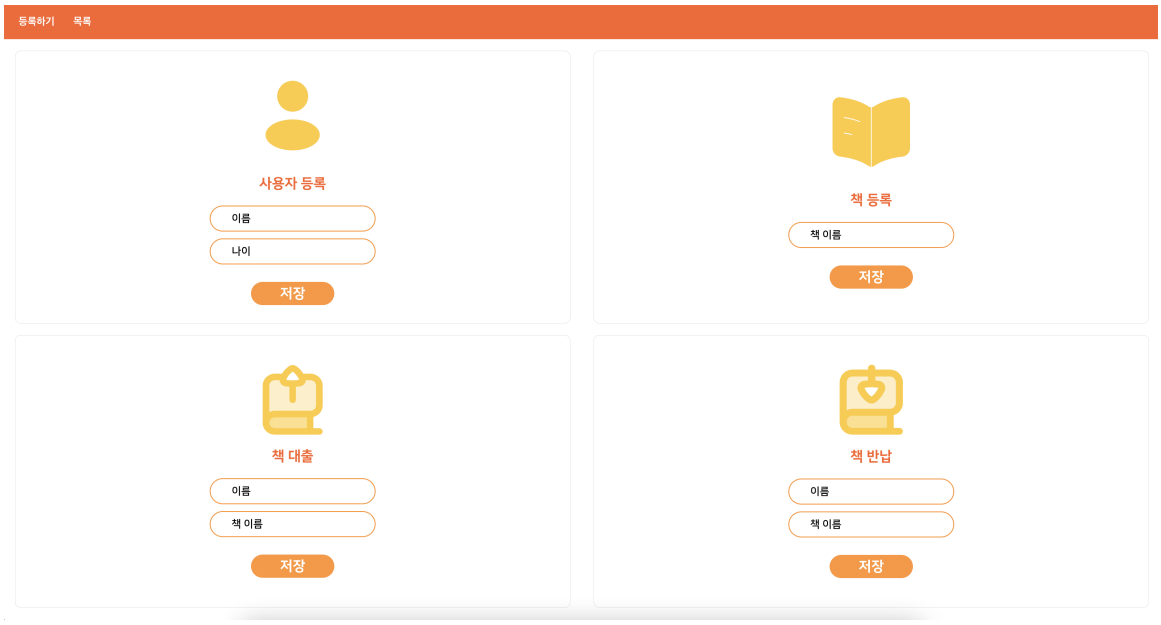
- 도서관의 사용자를 등록할 수 있다. (이름 필수, 나이 선택)
- 도서관 사용자의 목록을 볼 수 있다.
- 도서관 사용자 이름을 업데이트 할 수 있다.
- 도서관 사용자를 삭제할 수 있다.

책

- 도서관에 책을 등록할 수 있다.
- 사용자가 책을 빌릴 수 있다.
    - 다른 사람이 그 책을 진작 빌렸다면 빌릴 수 없다.

- 사용자가 책을 반납할 수 있다.

좋다~ 👍 관련해서 준비되어 있는 화면은 다음과 같다.



이번 시간에는 `도서관의 사용자를 등록할 수 있다 (이름 필수, 나이 선택)` 기능 개발을 해볼 것이다!

자 먼저 API를 설계해야 한다! 이제까지는 온전히 우리 마음대로 API를 설계했지만, 실제 현업에서는 그럴 수 없다! API 라는 것은 '약속'으로 클라이언트, 즉 화면 UI를 담당하는 개발자와 커뮤니케이션을 통해 '약속'을 맞춰야 한다. 앞으로 작성하게되는 도서관리 애플리케이션의 경우는 화면이 먼저 개발되어 있기에 화면에서 요구하는 API 스펙을 맞춰줘야 한다.

우리가 개발해야 하는 API는 다음과 같다.

- HTTP Method : POST
- HTTP Path : /user
- HTTP Body (JSON)

```
{
  "name": String (null 불가능),
  "age": Integer
}
```

- 결과 반환 X (HTTP 상태 200 OK이면 충분하다)

나이는 선택이기에 `age` 필드의 경우, null이 들어올 수도 있는 `Integer` 타입으로 되어 있다.

`name` 역시 null이 들어올 수도 있는 `String` 타입이다. 때문에 실제 로직에서 null인지 아닌지, 비어 있는 값은 아닌지 검증을 해주어야 할 것이다.

위 스펙을 코드로 구현해 보자~!

`com.group.libraryapp.controller` 패키지 안에 `user` 패키지를 만들고 `UserController` 를 만들어 주었다!

이제 연습했던 내용을 바탕으로 차근차근 코딩을 할 것이다. 우선 API의 뼈대를 잡아보자.

```
@RestController
public class Usercontroller {

  @PostMapping("/user")
  public void saveUser(@RequestBody UserCreateRequest request) {

  }

}
```

```
public class UserCreateRequest {

  private String name;
  private Integer age;

  public String getName() {
    return name;
  }

  public Integer getAge() {
    return age;
  }

}
```

- age에는 null이 들어올 수 있으니 `int` 대신 `Integer`를 타입으로 했다.

매우 좋다~ 새로운 개념은 없다! UserCreateRequest라는 DTO의 경우, 코드를 조금 더 간결하게 만들기 위해 final과 생성자를 사용하지 않았다. 본 강의에서는 앞으로도 DTO를 만들 때 꼭 필요한 경우가 아니라면, final과 생성자를 넣지 않을 것이다.

이제 뼈대를 잡았으니 실제 기능을 만들어야 한다. 원하는 것은 유저가 **저장**되는 것이다. 즉, 서버를 키고, API를 호출해 유저를 저장하면, 다시 불러올 수 있어야 한다.

가장 간단하게 API가 호출되면, User라는 클래스의 인스턴스를 만들고 이 데이터를 List에 저장하도록 하자. User 클래스는 새로 만들어야 한다. `com.group.libraryapp` 안에 `domain.user` 패키지를 만들어 User 클래스를 만들어주자.

```
public class User {

  private String name;
  private Integer age;

  public User(String name, Integer age) {
    if (name == null || name.isBlank()) {
      throw new IllegalArgumentException(String.format("잘못된 name(%s)이 들어왔습니다", na
me));
    }
    this.name = name;
    this.age = age;
  }

}
```

- name에는 null이 들어오면 안 되고, 이름이 비어 있을 수도 없으므로 생성자에서 값을 검증해주도록 했다!

이제 POST(저장) API에서 User 클래스를 List에 담도록 하면, 다음과 같다!

```
@RestController
public class UserController {

  private final List<User> users = new ArrayList<>();

  @PostMapping("/user")
  public void saveUser(@RequestBody UserCreateRequest request) {
    User newUser = new User(request.getName(), request.getAge());
    users.add(newUser);
  }

}
```

매우매우 좋다~~!! 👍 이번 시간에 유저를 저장하는 API를 개발하였으니, 이제 다음 시간에는 유저 조회 API를 개발할 것이다. 🔥

8강. 유저 조회 API 개발과 테스트

이번 시간에는 지난 시간에 이어 유저 조회 API를 개발하고 테스트할 예정이다.

바로 한 번 API 스펙부터 살펴보자!!

- HTTP Method : GET
- HTTP Path : /user
- 쿼리 : 없음
- 결과 반환

```
[{
  "id": Long,
  "name": String (null 불가능),
  "age": Integer
}, ...]
```

우리가 개발해야 하는 API 스펙과 관련해서, 2가지 생각해 볼거리가 있다.

[1. 결과 반환이 JSON?!]

우리가 연습했던 API와는 다르게 HTTP의 결과를 JSON으로 반환하는 모습이다. 어떻게 결과를 JSON으로 반환할 수 있을까?!

API의 응답 결과를 JSON으로 반환하는 방법은 간단하다! Controller에서 그냥 객체를 반환하면, JSON으로 응답이 가게 된다. **이때 객체에는 getter가 있어야 한다!**

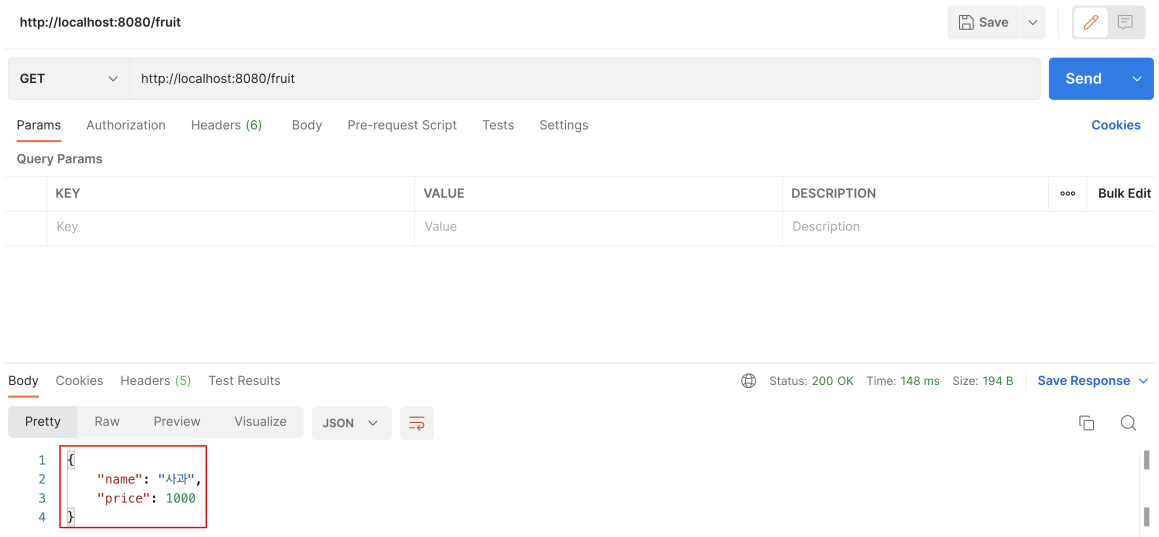
간단하게 API를 하나 만들어 확인해 보자!

```
public class Fruit {

  private String name;
  private long price;

  public Fruit(String name, long price) {
    this.name = name;
    this.price = price;
  }

  public String getName() {
```

```
    return name;
  }

  public long getPrice() {
    return price;
  }

}
```

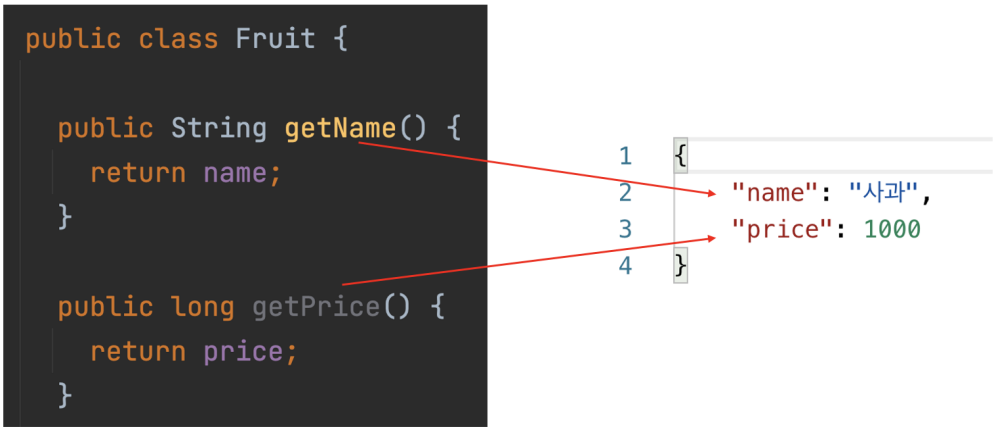
```
// 어차피 지울 코드니 UserController안에 넣어 주었다.
@GetMapping("/fruit")
public Fruit getFruit() {
  return new Fruit("사과", 1000L);
}
```

API를 만들고 POST MAN으로 간단히 확인해볼 수 있다!



POST MAN 결과를 확인해 보니 Body 부분에 우리가 설정해준 “사과", 1000L이 정상적으로 JSON에 담겨 오는 것을 확인할 수 있다!

Fruit에 있는 getter 이름과 JSON에 있는 key 이름이 동일하다.



이는 우리가 `@RestController` 어노테이션을 클래스에 붙여준 덕분에 가능한 일이다!

[2. id는 무엇인가?!]

id 라는 필드가 존재한다. 일반적으로 id란 데이터별로 겹치지 않는 유일한 번호를 의미한다.

때문에 API 스펙에 id가 있다는 의미는, User별로 고유한 번호를 API 응답 결과로 반환해주어야 한다는 점이다. List에 담겨 있는 유저의 순서를 id로 해주면, User별로 고유한 번호라고 생각할 수 있을 것이다!

매우 좋다~ 그럼 이제 GET API를 만들어 보자!

우선 API의 응답이 될 DTO부터 만들어주자. `com.group.libraryapp.dto.user` 패키지에 `response` 패키지를 추가로 만들어 `UserResponse` 클래스를 다음과 같이 만들어 주었다.

```
public class UserResponse {

  private long id;
  private String name;
  private Integer age;

  public UserResponse(long id, User user) {
    this.id = id;
    this.name = user.getName();
    this.age = user.getAge();
  }

  public long getId() {
    return id;
  }

  public String getName() {
    return name;
  }
```

```
  public Integer getAge() {
    return age;
  }

}
```

이 클래스를 만들기 위해서 `User` 쪽에서 2개의 getter를 추가해주었다.

```
// User.java에 추가된 2개의 함수
public String getName() {
  return name;
}

public Integer getAge() {
  return age;
}
```

Controller에는 다음과 같은 함수가 추가되었다.

```
@GetMapping("/user")
public List<UserResponse> getUsers() {
  List<UserResponse> responses = new ArrayList<>();
  for (int i = 0; i < users.size(); i++) {
    responses.add(new UserResponse(i + 1, users.get(i)));
  }
  return responses;
}
```

- `List<UserResponse>` : 여러명 있는 유저의 정보를 반환하기 때문에 List를 사용했다. 실제 API 스펙에서도 리스트임이 표기되어 있다.
- `for문` : id를 주기 위해 forEach 대신 for문을 사용하였다.
- `i + 1` : id는 일반적으로 1부터 시작하기 때문에 0부터 시작하는 index i에 1을 더해주었다.

매우 좋다~~ 😊 드디어 두 API가 모두 완성되었다. 두 API는 POST MAN을 사용해 테스트해도 되고, 실제 만들어져 있는 웹 UI로 들어가 테스트를 해도 된다.

웹 UI를 들어가 테스트 해보니 기능이 정상 동작한다~!! 🎉🎉

이제 다음 시간에는 이번 Section에서 다루었던 내용을 정리할 예정이다.

9강. Section1 정리. 다음으로!

이번 Section에서 우리는 API를 만들기 위해 필요한 기본적인 네트워크 지식을 익히고 2개의 API를 만들었다!! 이를 진행하며 다음과 같은 내용 역시 배울 수 있다. 👍

1. 스프링 프로젝트를 시작하는 방법과 실행하는 방법
2. 네트워크, IP, 도메인, 포트, HTTP 요청과 응답 구조, 클라이언트 - 서버 구조, API와 같은 기반 지식
3. Spring Boot를 이용해 GET API와 POST API를 만드는 방법

자 그런데~~ 우리가 개발한 API는 사실 한 가지 문제가 있다!! 바로 서버를 껐다 키면 유저 정보가 사라진다는 것이다! 조금 어렵게 말하면, 유저 정보는 메모리에서만 유지되고 있다.

이제 다음 섹션에서는 이러한 문제가 왜 일어나는지, 그리고 해결하기 위해 무엇이 필요한지 배우고 우리 서버에 적용해 볼 것이다! 🔥 🏃