

**3월 20일 수요일** - gradle 설정, yml 문법, 네이버 open api - 지역, 이미지, postman, intelliJ(naver, controller, test), main.html, main.js

앞으로 배울 내용들

1. Spring(boot)기반의 외부 정보 연결 및 OpenAPI 사용방법
2. JAVA Spring 기반의 스케줄 프로그램 만들기(ex) 아이템, 회원, 생산))
3. SCM에 대한 개론 학습
4. 간단한 테스트 기반(JUnit)의 현재고 데이터 생성
5. 스프링 Security 활용하기

\\192.168.0.104\github-pktjesus\2401-human-suwon-frontend1\spring-boot-study

SPA(Single Page Application)

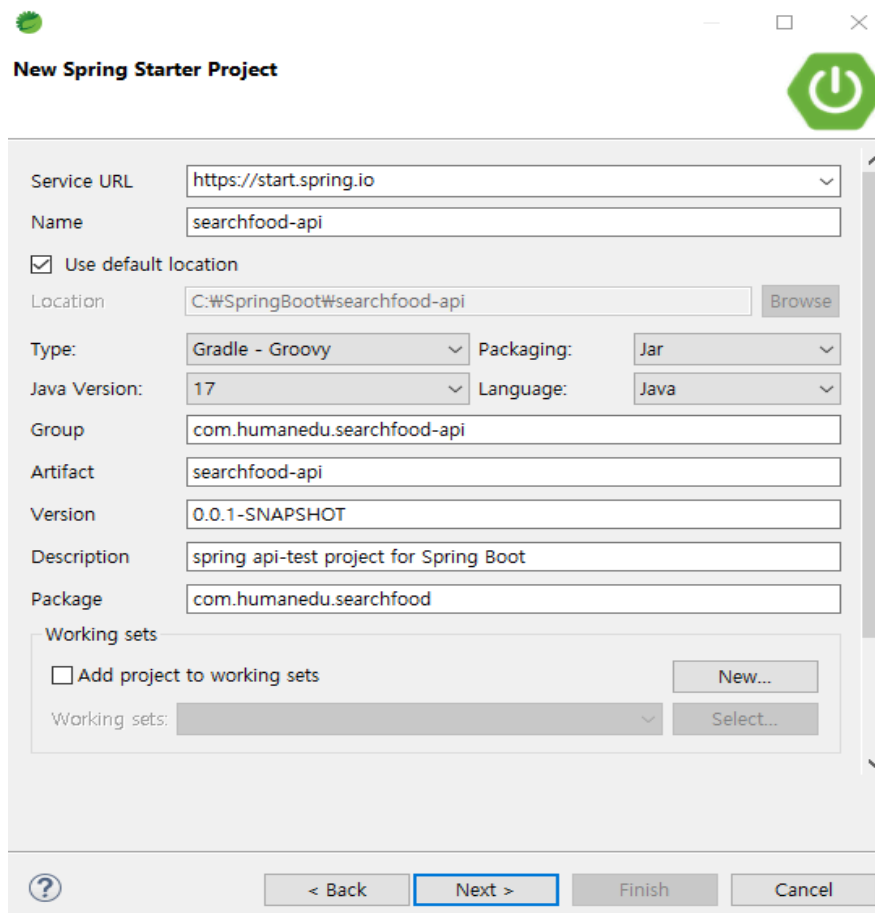
html + javascript + css(bootstrap) + vue + ajax

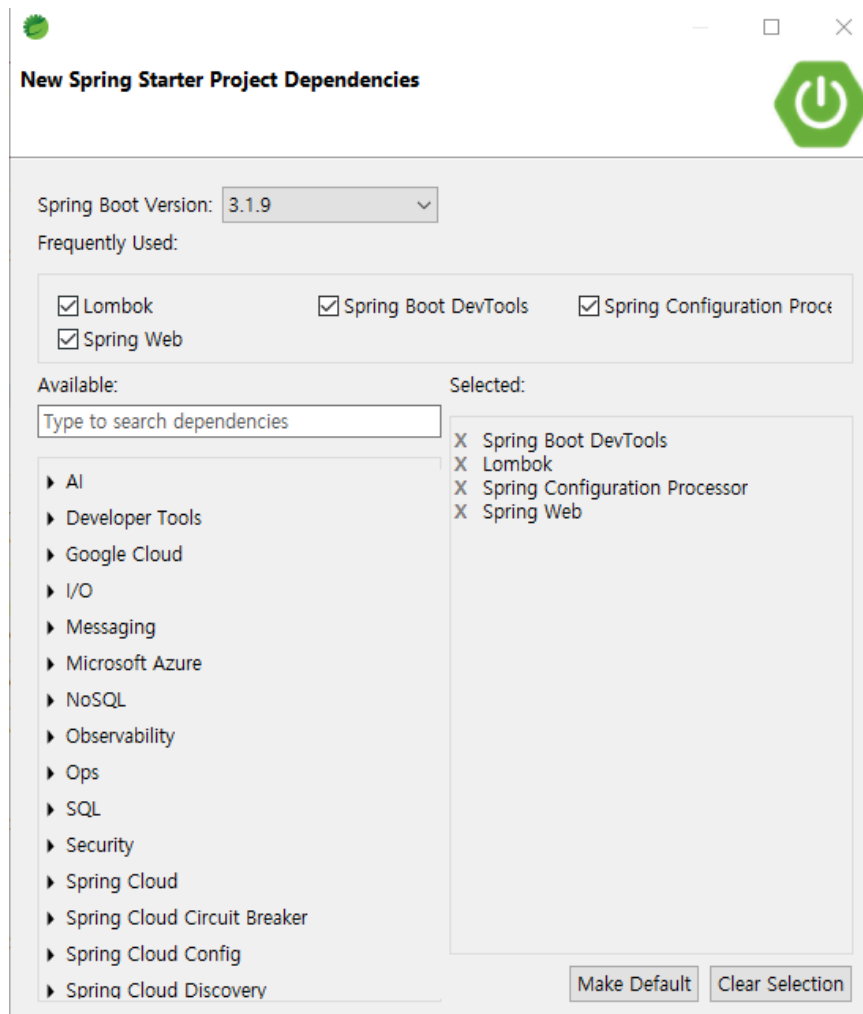
→

: 요즘 vue 는 잘 안쓰고 react를 더 많이 사용한다 ( react 공부해야한다)

→ Spring boot(thymeleaf) → backend: JSP 를 사용하지 않고 thymeleaf 사용한다

시작: STS4에 새로운 폴더 생성(스프링 부트는 설정하는 이름이 정말 중요하다.)





이번엔 Gradle-Groovy 로 설정해주었다. 이 둘의 차이는?

maven(pom.xml) VS gradle(build.gradle)

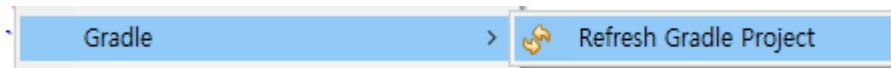
<https://www.notion.so/Maven-Gradle-c1870b93482e4ca881eee27ee5d05da1?pvs=4>  
[https://kotlinworld.com/320#google\\_vignette](https://kotlinworld.com/320#google_vignette)

```

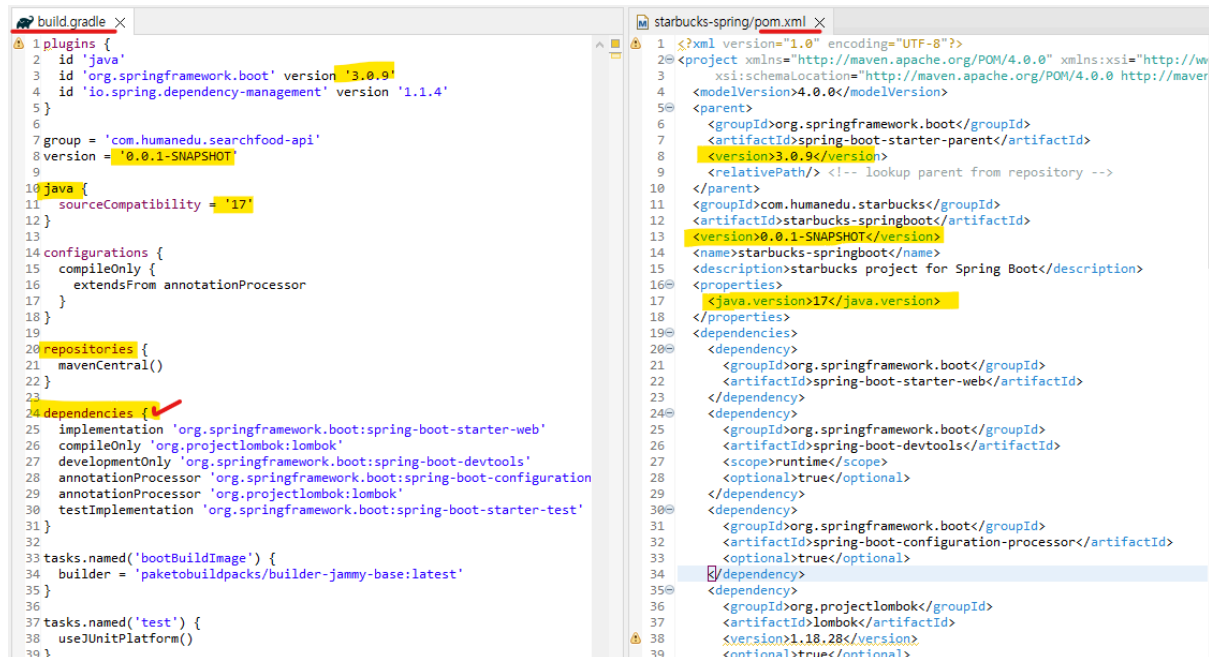
build.gradle
1 plugins {
2   id 'java'
3   id 'org.springframework.boot' version '3.0.9'
4   id 'io.spring.dependency-management' version '1.1.4'
5 }
6
7 group = 'com.humaneu.searchfood-api'
8 version = '0.0.1-SNAPSHOT'
9
10 java {
11   sourceCompatibility = '17'
12 }
  
```

3.1.9를 3.0.9로 바꿔준다

메이븐처럼 refresh

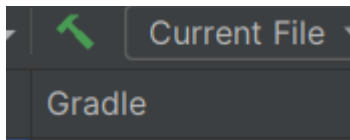


새 스프링설정시 넣었던 dependencies

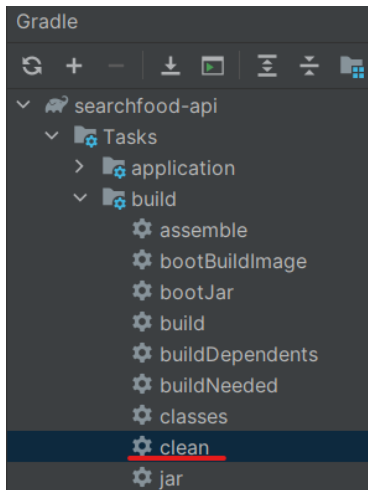


dependencies 줄바꿈으로 정리 및 추가

```
24 dependencies {
25     implementation 'org.springframework.boot:spring-boot-starter-web'
26     implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
27
28     compileOnly 'org.projectlombok:lombok'
29
30     developmentOnly 'org.springframework.boot:spring-boot-devtools'
31
32     annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
33     annotationProcessor 'org.projectlombok:lombok'
34
35     testImplementation 'org.springframework.boot:spring-boot-starter-test'
36     testImplementation 'org.junit.jupiter:junit-jupiter'
37
38     testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
39 }
```



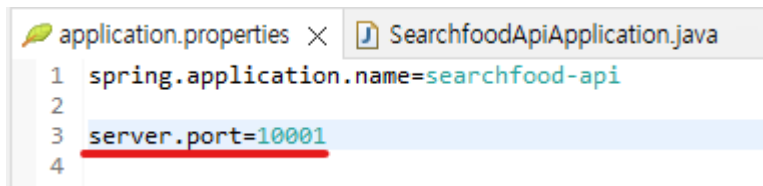
완성 후 intellij로 실행 > gradle



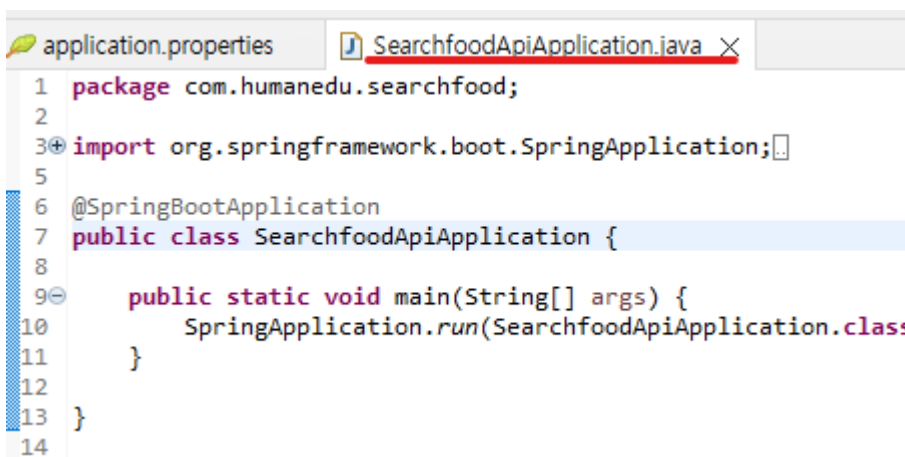
메이븐과 같이 그레들도 클린해줄 수 있다

디버깅을 실행시켜서 스프링이 작동되는지 확인한다

다시 STS4 에서 아래를 추가 (위아래가 바뀌어도 상관이 없다)



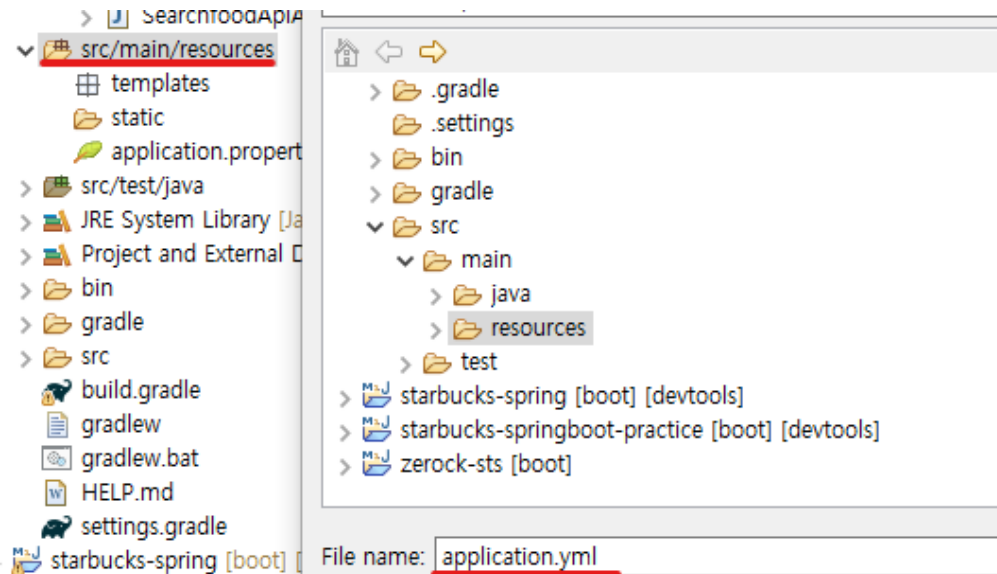
아래에서 실행시켜준다



성공하면 아래처럼 콘솔에 나온다

```
ServerApplicationContext : Root WebApplicationContext: initialization completed in 811 ms
naliveReloadServer : Unable to start LiveReload server
d.tomcat.TomcatWebServer : Tomcat started on port(s): 10001 (http) with context path ''
SearchfoodApiApplication : Started SearchfoodApiApplication in 1.529 seconds (process ru
```

new > file 파일명은 꼭! 명확히 해줘야한다



사용하는 문법이 다르다>> **yml** 문법 구글 검색 <https://lejewk.github.io/yaml-syntax/>

**json**과 다르게 주석이 되고 가독성이 좋다

<https://somefood.tistory.com/entry/%ED%8F%AC%EB%A7%B7-%EC%A0%95%EB%A6%AC-YAML-%EB%AC%B8%EB%B2%95-%EC%A0%95%EB%A6%AC>

## 가독성 비교

### Xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <apiVersion>v1</apiVersion>
  <kind>Pod</kind>
  <metadata>
    <name>hello-pod</name>
    <labels>
      <app>hello</app>
    </labels>
  </metadata>
  <spec>
    <containers>
      <name>hello-container</name>
      <image>tmkube/hello</image>
      <ports>
        <containerPort>8000</containerPort>
      </ports>
    </containers>
  </spec>
</root>
```

### Json

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "hello-pod",
    "labels": {
      "app": "hello"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "hello-container",
        "image": "tmkube/hello",
        "ports": [
          {
            "containerPort": 8000
          }
        ]
      }
    ]
  }
}
```

### Yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
  labels:
    app: hello
spec:
  containers:
  - name: hello-container
    image: tmkube/hello
    ports:
    - containerPort: 8000
```

띄어쓰기 두번 꼭 해줘야한다.

```
application.yml ×
1 server:
2   port: 10002
3
```

두개를 다른 port 번호를 사용하면 properties 가 작동되어 10001이 된다. 즉 properties가 먼저

파일 이름을 다르게하면 스프링부트에서 읽지를 못한다 예시) \_\_application.properties

```
application.yml ×      application.properties ×
1 server:
2   port: 10002
3
1 server.port=10001
2 spring.application.n
3
```

이름을 \_\_application.properties 로 변경해주면 아래처럼 10002가 나온다

```
: Unable to start LiveReload server
: Tomcat started on port(s): 10002 (http) with context path ''
: Started SearchfoodApiApplication in 0.22 seconds (process runni
```

```
build.gradle  __application.properties  application.properties ×
1 server.port=8081
2 server.error.whitelabel.enabled=true
3
4 spring.mvc.view.prefix=/WEB-INF/views/
5 spring.mvc.view.suffix=.jsp
```

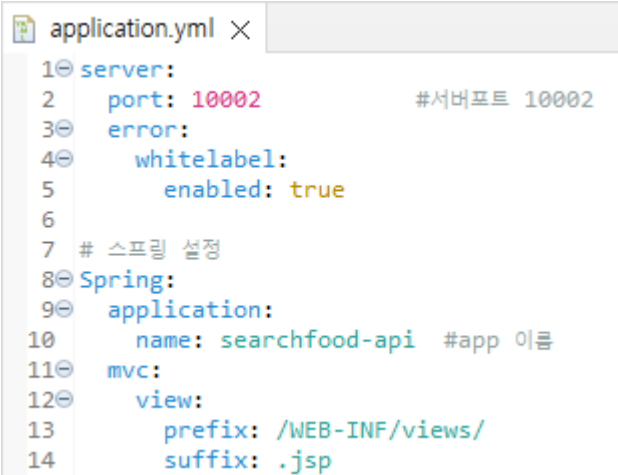
를 yml 문법으로 바꿔보기

<띄어쓰기 중요/ 주석은 #>

```
server:
  port: 10002      #서버포트 10002
  error:
    whitelabel:
      enabled: true
# 스프링 설정
Spring:
  application:
    name: searchfood-api #app 이름
  mvc:
    view:
      prefix: /WEB-INF/views/
      suffix: .jsp
```

<https://developers.naver.com/main/> 로그인

<https://www.data.go.kr/>



```
application.yml ×
1 server:
2   port: 10002      #서버포트 10002
3   error:
4     whitelabel:
5       enabled: true
6
7 # 스프링 설정
8 Spring:
9   application:
10    name: searchfood-api #app 이름
11   mvc:
12     view:
13       prefix: /WEB-INF/views/
14       suffix: .jsp
```

애플리케이션 이름 ↗	<input type="text" value="search-food"/> ✓ <ul style="list-style-type: none"> <li>네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요.</li> <li>40자 이내의 영문, 한글, 숫자, 공백문자, 쉼표(,), "/" , "-" , "_" , 만 입력 가능합니다.</li> </ul>
사용 API ↗	<div> <div>선택하세요. ▼</div> <div>✓</div> </div> <div> <div>검색</div> <div>✕</div> </div>
비로그인 오픈 API 서비스 환경	<div> <div>환경 추가 ▼</div> <div> <div>WEB 설정 ✕ ^</div> <div> <div>웹 서비스 URL (최대 10개)</div> <div> <input type="text" value="http://localhost"/> + ✓ </div> <ul style="list-style-type: none"> <li>텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.</li> <li>http와 https는 구분하지 않습니다.</li> <li>www는 빼고 입력해 주세요. 예) http://naver.com</li> <li>서브 도메인이 있으면 대표 도메인명만 입력해 주세요. (예: http://naver.com)</li> <li>하이브리드 앱은 location.href 객체 출력 값을 입력하면 됩니다. (예: file://로컬 URI)</li> </ul> </div> </div> </div>

localhost... 아무거나 넣어도 된다 나중에 수정할 수 있다.

## 애플리케이션 정보

Client ID	LAcPTCPL_DIKVnoSBO50
Client Secret	A8mWZtCs4F
	재발급

절대 깃허브에 올리면 안된다.

### API 공통 가이드

#### 네이버 오픈API 종류

로그인 방식 오픈 API

비로그인 방식 오픈 API

사전 준비 사항

내 애플리케이션 관리

용어 정리

샘플 코드

오류 코드

### 비로그인 방식 오픈 API ↗

비로그인 방식 오픈 API는 HTTP 헤더에 클라이언트 아이디와 클라이언트 시크릿 값만 전송해 사용할 수 있는 오픈 API 인의 인증을 통한 접근 토큰을 획득할 필요가 없습니다.

다음과 같은 네이버 오픈API가 비로그인 방식 오픈 API입니다.

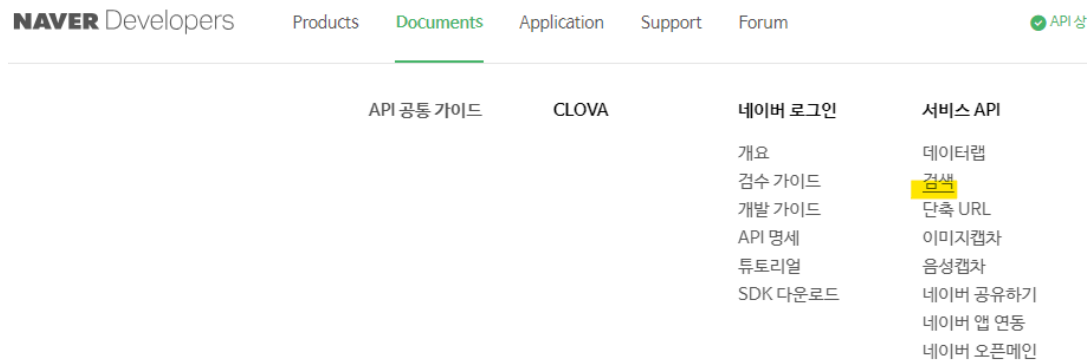
- 데이터랩: 네이버 데이터랩의 검색어 트렌드와 쇼핑인사이트**를 API로 실행할 수 있게 하는 API입니다.
- 검색:** 네이버 검색 결과를 뉴스, 백과사전, 블로그, 쇼핑, 웹 문서, 전문정보, 지식IN, 책, 카툰 등 분야별로 볼 수 있는 API 검색 결과와 성인 검색어 판별 기능, 오다 변환 기능을 제공합니다.
- 단축URL:** 원본 URL을 https://me2.do/example 과 같은 형태의 짧은 URL로 반환받을 수 있는 API입니다.
- 이미지 캡차:** 네이버 서비스에서 사용하는 이미지 캡차 기능을 외부 서비스에 사용할 수 있게 하는 API입니다.
- 음성 캡차:** 네이버 서비스에서 사용하는 음성 캡차 기능을 외부 서비스에 사용할 수 있게 하는 API입니다.
- 네이버 공유하기:** 콘텐츠를 네이버 블로그, 네이버 카페, PHOLAR에 공유할 수 있게 하는 API입니다.

네이버 오픈API는 웹 페이지를 네이버 앱에 추가할 수 있게 하는 푸시 알림 API



## Postman 로 테스트하기

<https://developers.naver.com/docs/serviceapi/search/local/local.md#%EC%A7%80%EC%97%AD>



## 지역과 이미지를 사용

### 1. 지역

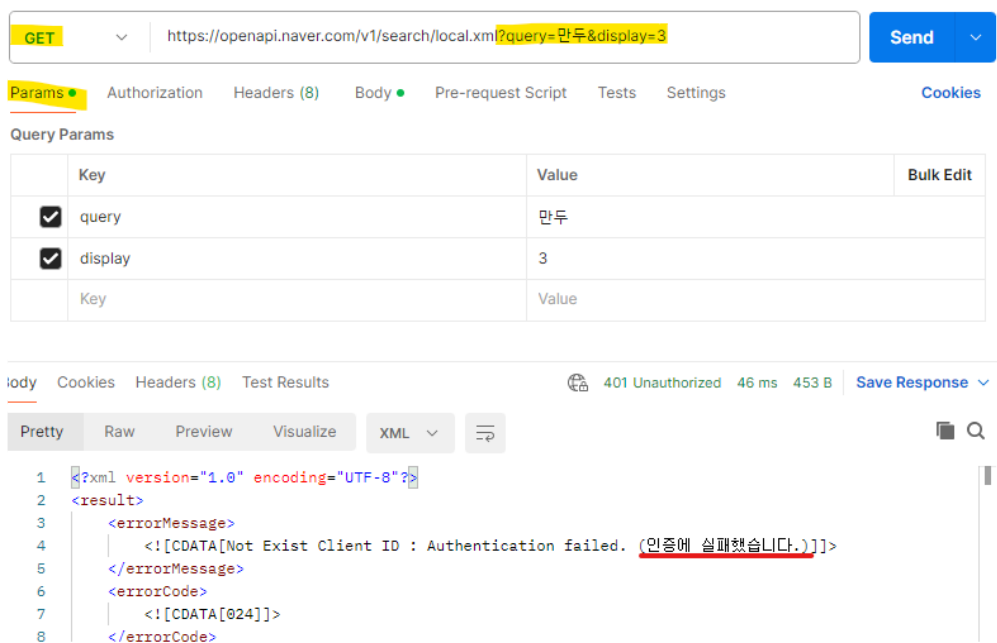
요청 예

```
curl "https://openapi.naver.com/v1/search/local.xml?query=%EC%A3%BC%EC%88%9D&display=10&start=1&sort=random" \
-H "X-Naver-Client-Id: {애플리케이션 등록 시 발급받은 클라이언트 아이디 값}" \
-H "X-Naver-Client-Secret: {애플리케이션 등록 시 발급받은 클라이언트 시크릿 값}" -v
```

## Postman

Get - Params 나 링크에 바로 ?를 사용해 넣을수 있다

Post - Params에서만 넣어야한다// 여기선 post 사용할 수 없다



send를 누르면 인증에 실패.

## 참고 사항 ↗

API를 요청할 때 다음 예와 같이 HTTP 요청 헤더에 **클라이언트 아이디**와 **클라이언트 시크릿**을 추가해야 합니다.

```
> GET /v1/search/local.xml?query=%EC%A3%BC%EC%88%9D&display=10&start=1&sort=random HTTP/1.1
Host: openapi.naver.com
User-Agent: curl/7.49.1
Accept: */*
X-Naver-Client-Id: {애플리케이션 등록 시 발급받은 클라이언트 아이디 값}
X-Naver-Client-Secret: {애플리케이션 등록 시 발급받은 클라이언트 시크릿 값}
```

위를 Postman - Headers에 추가한다

<xml 그리고 display=3>

The screenshot shows a Postman interface for an API request. The URL is `https://openapi.naver.com/v1/search/local.xml?query=만두&display=3`. The Headers tab is selected, showing the following headers:

Key	Value
Host	openapi.naver.com
User-Agent	curl/7.49.1
Accept	*/*
X-Naver-Client-Id	LAcPTCPI_DIKVnoSBO5O
X-Naver-Client-Secret	A8mWZtCs4F

The Body tab is selected, showing the XML response:

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Naver Open API - local ::&apos;만두&apos;</title>
<link>https://search.naver.com</link>
<description>Naver Search Result</description>
<lastBuildDate>Wed, 20 Mar 2024 12:25:24 +0900</lastBuildDate>
<total>3</total>
<start>1</start>
<display>3</display>
<item>
  <title>애플하우스</title>
  <link></link>
  <category>분식&gt;떡볶이</category>
  <description></description>
  <telephone></telephone>
  <address>서울특별시 동작구 사당동 138-4 2층</address>
  <roadAddress>서울특별시 동작구 동작대로27다길 29 2층</roadAddress>
  <mapx>1269806796</mapx>
  <mapy>374863013</mapy>
</item>
</xml>
```

<jason, display=1>

GET

https://openapi.naver.com/v1/search/local.json?query=만두&display=1

Send

Params

Authorization

Headers (13)

Body

Pre-request Script

Tests

Settings

Cook

Query Params

	Key	Value	Bulk E
<input checked="" type="checkbox"/>	query	만두	
<input checked="" type="checkbox"/>	display	1	
	Key	Value	

body

Cookies

Headers (13)

Test Results

200 OK 72 ms 700 B

Save Respons

Pretty

Raw

Preview

Visualize

JSON

```
1
2   "lastBuildDate": "Wed, 20 Mar 2024 12:30:32 +0900",
3   "total": 1,
4   "start": 1,
5   "display": 1,
6   "items": [
7     {
8       "title": "애플하우스",
9       "link": "",
10      "category": "분식>떡볶이",
11      "description": "",
12      "telephone": "",
13      "address": "서울특별시 동작구 사당동 138-4 2층",
14      "roadAddress": "서울특별시 동작구 동작대로27다길 29 2층",
15      "mapx": "1269806796",
16      "mapy": "374863013"
17    }
18  ]
19
```

## 2. 이미지

뉴스

책

성인 검색어 판별

백과사전

카페글

지식IN

지역

오타변환

웹문서

이미지

쇼핑

참고 사항

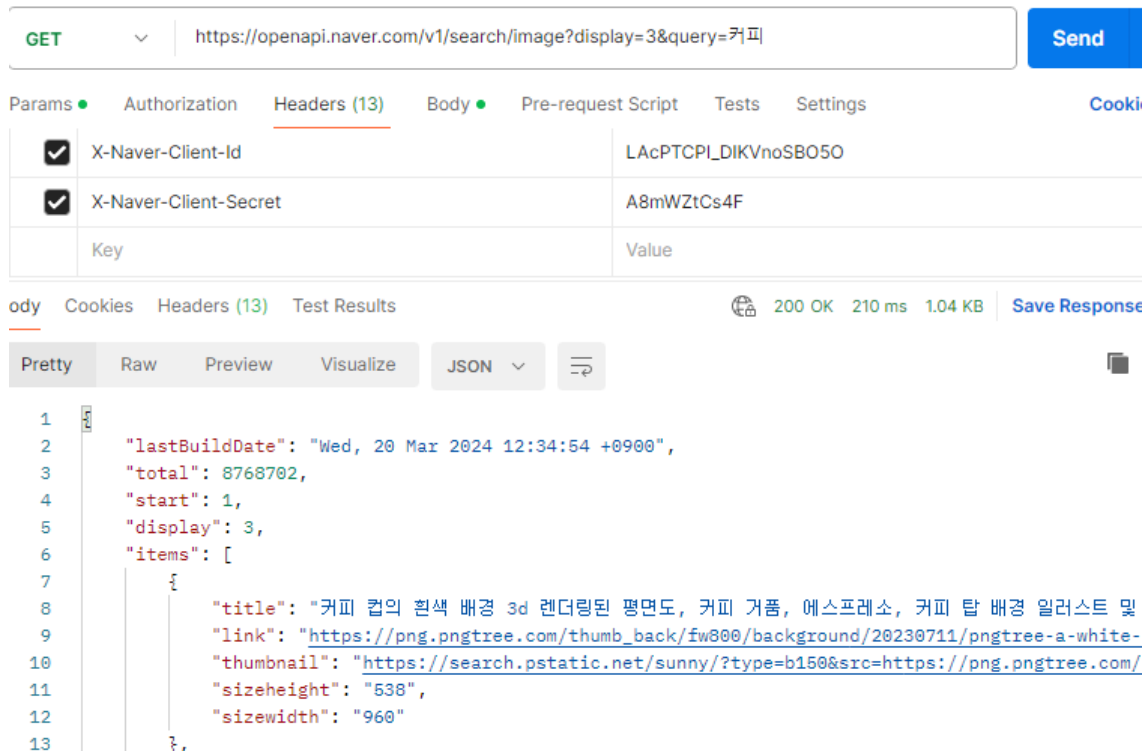
API를 요청할 때 다음 예와 같이 HTTP 요청 헤더에 클라이언트 아이디와 클라이언트 시크릿을 추가해야 합니다.

```
> GET /v1/search/image.xml?query=%EC%A3%BC%EC%88%90&display=10&start=1&sort=sim HTTP/1.1
> Host: openapi.naver.com
> User-Agent: curl/7.49.1
> Accept: */*
> X-Naver-Client-Id: {애플리케이션 등록 시 발급받은 클라이언트 아이디 값}
> X-Naver-Client-Secret: {애플리케이션 등록 시 발급받은 클라이언트 시크릿 값}
```

요청 예

```
curl "https://openapi.naver.com/v1/search/image.xml?query=%EC%A3%BC%EC%88%90&display=10&start=1&sort=sim" \
-H "X-Naver-Client-Id: {애플리케이션 등록 시 발급받은 클라이언트 아이디 값}" \
-H "X-Naver-Client-Secret: {애플리케이션 등록 시 발급받은 클라이언트 시크릿 값}" -v
```

결과값 image? 뒤에 xml 또는 json을 넣어주지 않고 출력해보았다



GET <https://openapi.naver.com/v1/search/image?display=3&query=커피> Send

Params Authorization Headers (13) Body Pre-request Script Tests Settings Cookies

Key	Value
X-Naver-Client-Id	LAcPTCPI_DIKVnoSBO5O
X-Naver-Client-Secret	A8mWZtCs4F

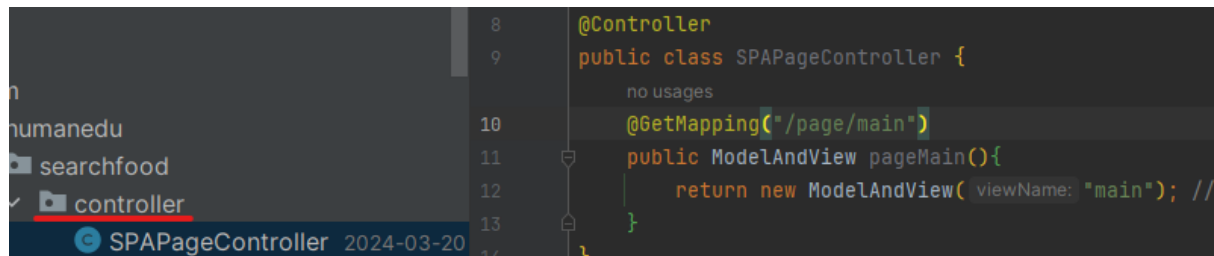
ody Cookies Headers (13) Test Results 200 OK 210 ms 1.04 KB Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "lastBuildDate": "Wed, 20 Mar 2024 12:34:54 +0900",
3    "total": 8768702,
4    "start": 1,
5    "display": 3,
6    "items": [
7      {
8        "title": "커피 컵의 흰색 배경 3d 렌더링된 평면도, 커피 거품, 에스프레소, 커피 탐 배경 일러스트 및
9        "link": "https://png.pngtree.com/thumb_back/fw800/background/20230711/pngtree-a-white-
10       "thumbnail": "https://search.pstatic.net/sunny/?type=b150&src=https://png.pngtree.com/
11       "sizeheight": "538",
12       "sizewidth": "960"
13     },
14   ]
15 }
```

naver는 post로 제공해주지 않기때문에 get으로만 send 할 수 있다.

intellij 들어가서 controller 생성 – String이 아닌 ModelAndView 를 사용했다



<http://localhost:10002/page/main> 들어가면 아래처럼 오류페이지가 나온다 (500 or 404)

## Whitelabel Error Page

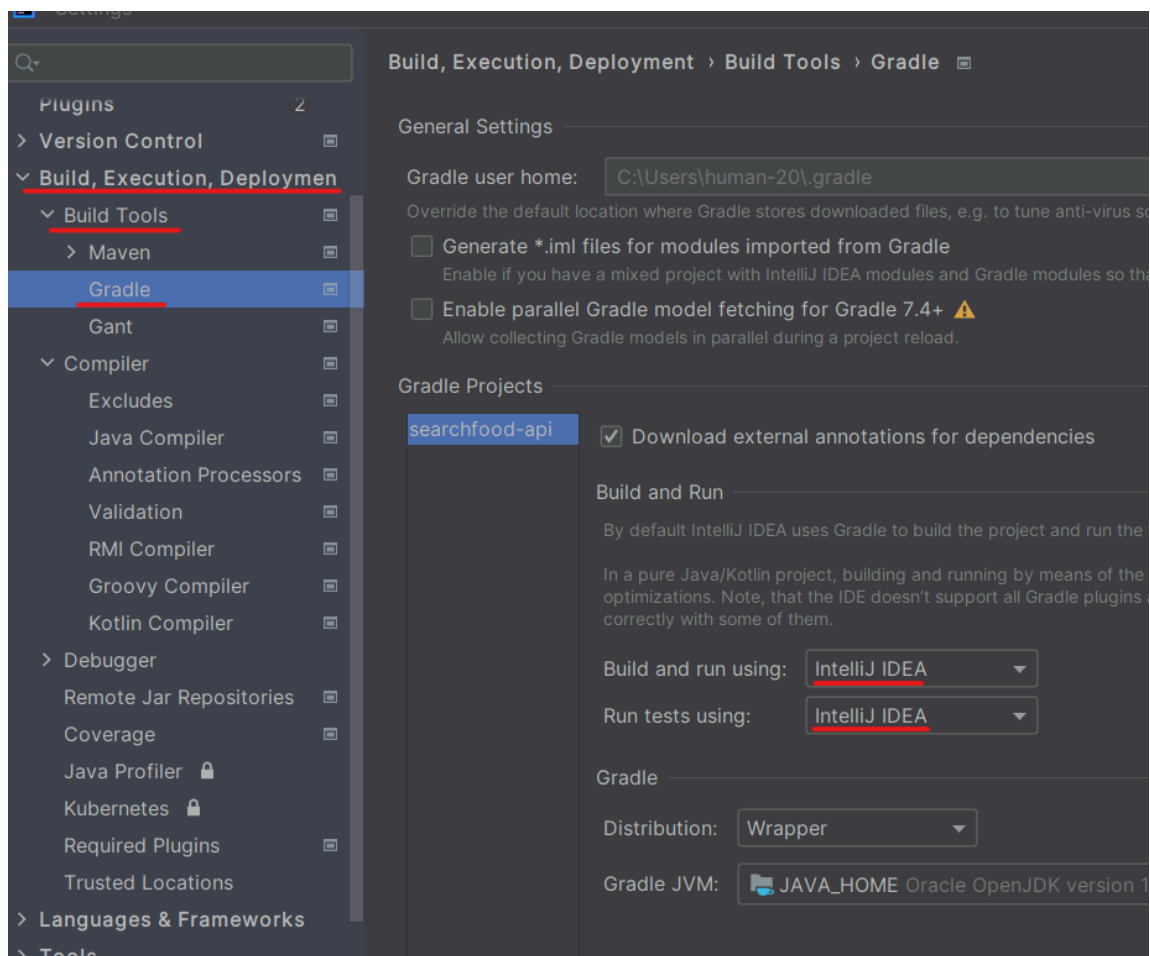
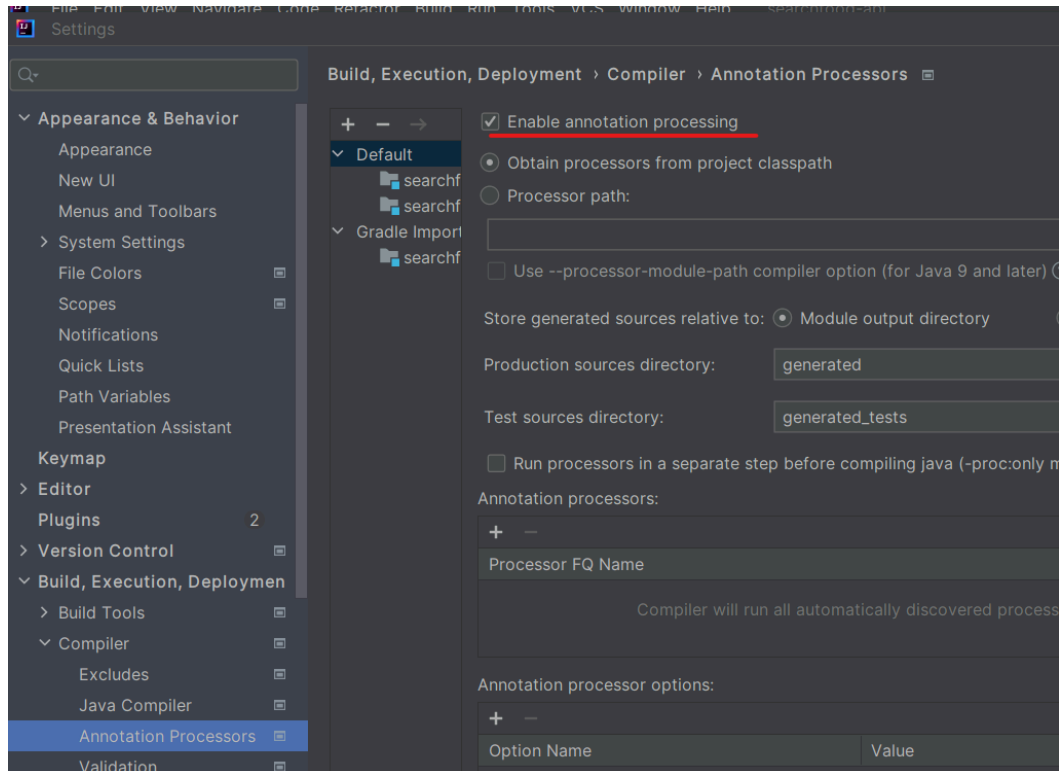
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Mar 20 14:03:44 KST 2024

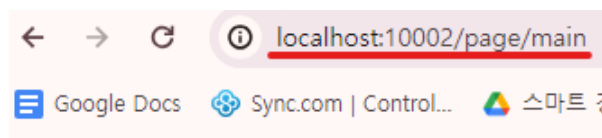
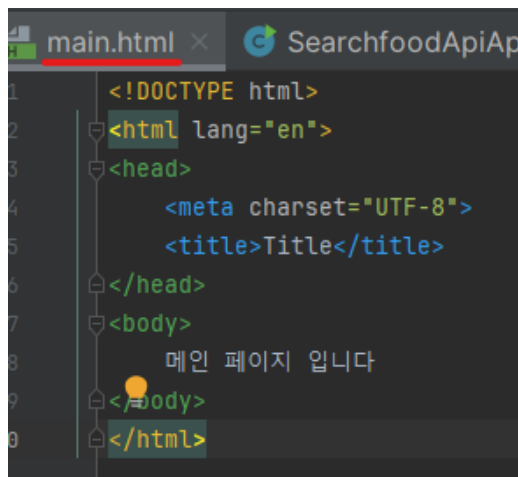
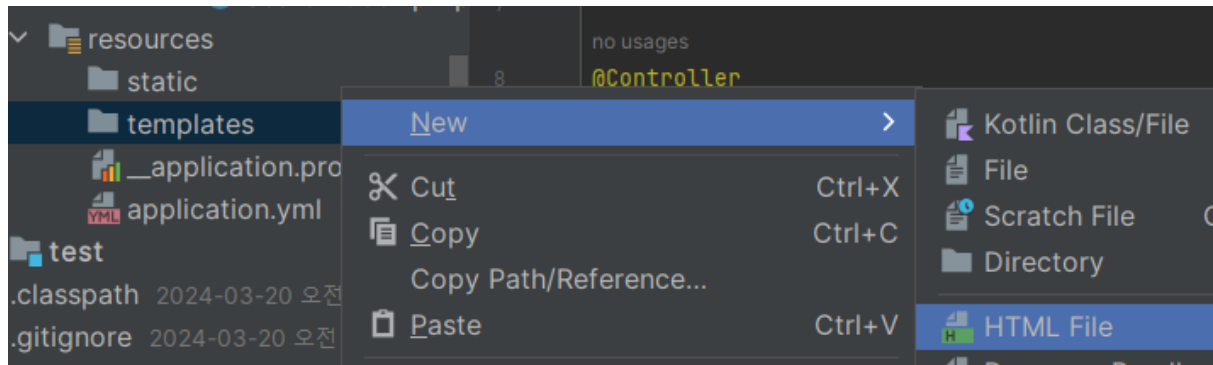
There was an unexpected error (type=Internal Server Error, status=500).

Error resolving template [main], template might not exist or might not be accessible by org.thymeleaf.exceptions.TemplateInputException: Error resolving template [main], template not found. Resolvers

## 설정해주기



HTML 파일 만들어주기

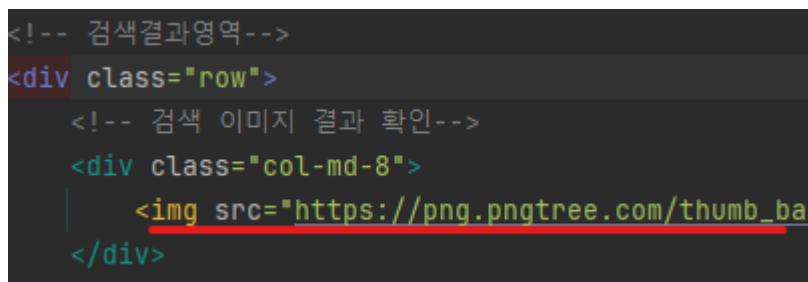


메인 페이지 입니다

한 줄 복사 Ctrl + D

주석 Ctrl + Shift + /

main.html에 아까 Get한 커피 이미지 링크 붙여넣기



GET ▼  Send ▼

Params ● Authorization Headers (13) Body ● Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	display	1	
<input checked="" type="checkbox"/>	query	커피	
	Key	Value	

Body Cookies Headers (13) Test Results 🔒 200 OK 232 ms 858 B Save Response

Pretty Raw Preview Visualize JSON ▼ 📄 C

```

4 1,
5 : 1,
6 [
7
8   title: "커피 컵의 흰색 배경 3d 렌더링된 평면도, 커피 거품, 에스프레소, 커피 탭 배경 일러스트 및 사진 무료 다운로드",
9   link: "https://png.pngtree.com/thumb_back/fw800/background/20230711/pngtree-a-white-background-3d-",
10  thumbnail: "https://search.pstatic.net/sunny/?type=b150&src=https://png.pngtree.com/thumb_back/fw800",
11  sizeheight: "538",
12  sizewidth: "960"
13 ]
14 }

```

js 에 사용한 backtick 백틱의 사용법 ``?? ajax 사용법

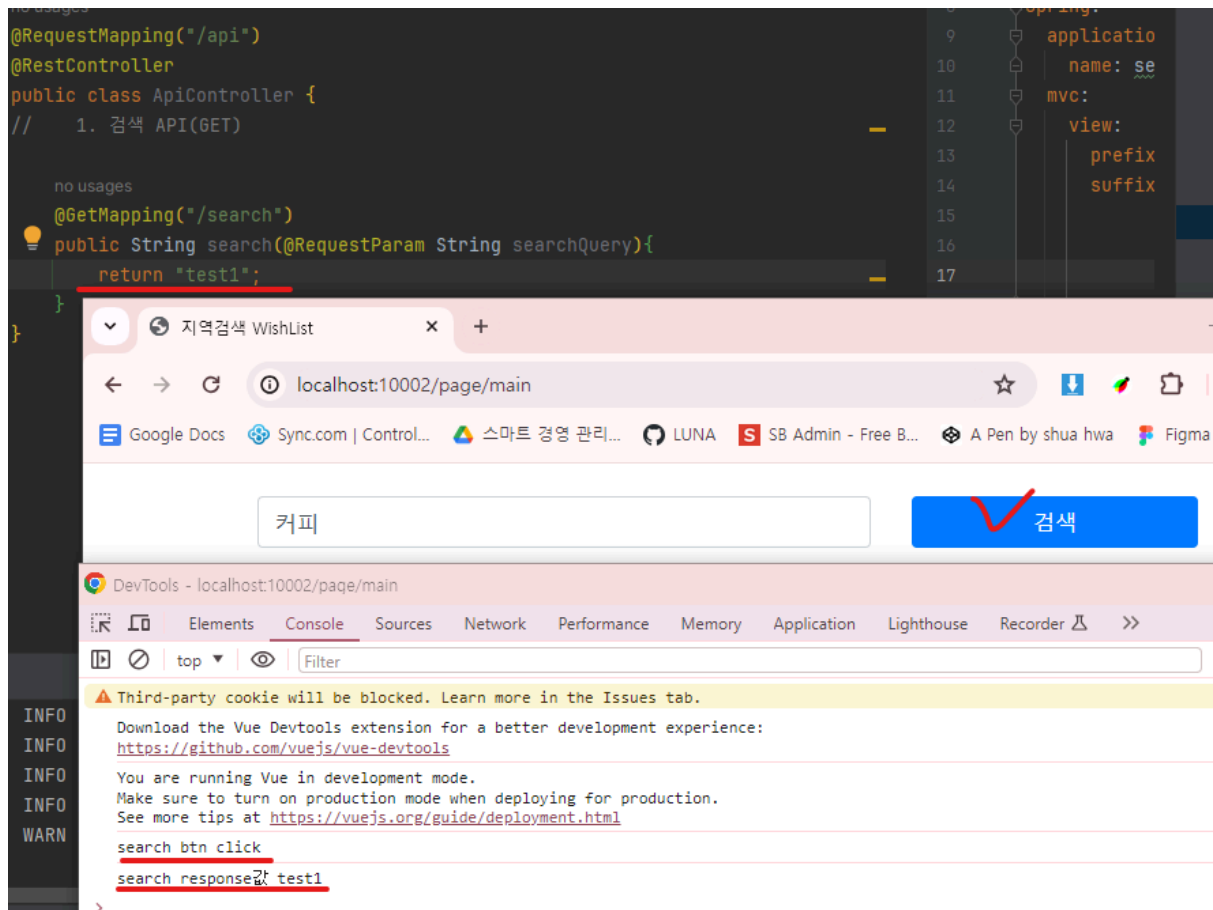
```

in.html x main.js x SPAPageController.java x ApiController.java
//console.log("hello");

//검색 버튼을 눌렀을 때 이벤트
$('#searchButton').click(function(){
    console.log('search btn click');

    const query = $('#searchBox').val(); //검색어
    // 실제 backend에 /api/search 요청해서 데이터 가지고오기 backtick``문자
    $.get(`/api/search?searchQuery=${query}`, function(response){
        console.log('search response', response);
    });
});

```



이 코드는 JavaScript와 jQuery를 사용하여 AJAX 요청을 보내고, 백엔드 서버에서 데이터를 가져오는 기능을 수행합니다. 아래는 코드의 각 부분에 대한 설명입니다.

1. `const query = $('#searchBox').val();`: 이 코드는 id가 "searchBox"인 HTML 요소의 값을 가져와서 `query` 변수에 할당합니다. 보통 검색 창에 입력된 검색어를 가져오기 위해 사용됩니다.
2. `$.get('/api/search?searchQuery=${query}', function(response){...});`: 이 코드는 jQuery의 `$.get()` 함수를 사용하여 GET 요청을 보냅니다. 이 요청은 `/api/search` 엔드포인트로 보내지며, URL 쿼리 문자열로 검색어를 전달합니다. `${query}`는 앞서 가져온 검색어를 사용합니다. 이 요청은 백엔드 서버에서 검색을 처리하는 데 사용됩니다.
3. `function(response){...}`: 이 부분은 AJAX 요청이 성공했을 때 실행할 콜백 함수를 정의합니다. 요청에 대한 응답이 `response` 매개변수로 전달됩니다.
4. `console.log('search response값', response);`: 이 코드는 AJAX 요청이 성공했을 때 서버에서 받은 응답을 콘솔에 출력합니다. 이 부분은 실제로는 서버에서 받은 응답을 처리하거나 화면에 표시하는 등의 로직으로 대체될 수 있습니다.

따라서 이 코드는 사용자가 입력한 검색어를 백엔드 서버로 보내고, 서버에서 처리한 결과를 콘솔에 출력하는 간단한 검색 기능을 구현합니다.



naver API @Component //bean으로 등록하기 위해서 넣어준다.

→ @Component 애노테이션은 스프링 프레임워크에서 빈(Beans)으로 등록하기 위해 사용됩니다. 스프링 애플리케이션 컨텍스트에 의해 관리되는 객체를 빈이라고 합니다.

@Component 애노테이션은 해당 클래스가 컴포넌트 스캔의 대상임을 나타내며, 스프링이 해당 클래스의 인스턴스를 생성하고 애플리케이션 컨텍스트에 등록하게 됩니다.

따라서 위 코드에서 @Component 애노테이션을 사용하면 해당 클래스가 빈으로 등록되어 스프링 애플리케이션 컨텍스트에서 사용할 수 있게 됩니다. 이 경우에는 해당 클래스가 어떤 역할을 수행하는지에 따라 적절한 이름을 지정하여 @Component 대신 @Service, @Repository 등의 어노테이션을 사용할 수도 있습니다.

request DTO 에 넣는것 <http://surl.li/rsxys>

#### 파라미터

파라미터를 쿼리 스트링 형식으로 전달합니다.

파라미터	타입
query	String
display	Integer
start	Integer
sort	String

```
package com.humanedu.searchfood.dto;

2 usages
public class SearchRegionRequestDto {

    no usages
    private String query;

    no usages
    private Integer display = 1;

    no usages
    private Integer start = 1;

    no usages
    private String sort = "random";
}
```

response DTO 에 넣어주는 것

GET <https://openapi.naver.com/v1/search/local.json?query=커피>

Params • Authorization Headers (13) Body • Pre-request Script Tests Settings

Query Params

Key	Value
<input type="checkbox"/> display	1
<input checked="" type="checkbox"/> query	커피

body Cookies Headers (13) Test Results 200 OK 81

Pretty Raw Preview Visualize JSON

```

1  {
2    "lastBuildDate": "Wed, 20 Mar 2024 15:15:14 +0900",
3    "total": 1,
4    "start": 1,
5    "display": 1,
6    "items": [
7      {
8        "title": "미철헤어<b>커피</b><b>커피</b> 명동1호점",
9        "link": "https://blog.naver.com/kerker2328",
10       "category": "생활,편의>마용실",
11       "description": "",
12       "telephone": "",
13       "address": "서울특별시 중구 충무로1가 24-6 5층",
14       "roadAddress": "서울특별시 중구 명동8나길 9 5층",
15       "mapx": "1269844364",
16       "mapy": "375613115"
17     }
18   ]
19 }

```

items 는 List 배열로 넣어줘야한다

```

@Data
@AllArgsConstructor
@NoArgsConstructor // get 과 set을 위해 3 어노테이션 넣기
public class SearchRegionResponseDto {

    private String lastBuildDate;

    private Integer total = 1;
    private Integer start = 1;
    private Integer display = 1;

    private List<SearchRegionItem> items;

    1 usage
    @Data
    @AllArgsConstructor
    @NoArgsConstructor
    public static class SearchRegionItem {
        private String title;
        private String link;
        private String category;
        private String description;
        private String telephone;
        private String address;
        private String roadAddress;
        private String mapx;
        private String mapy;
    }

```

네이버 비로그인-방식으로 하지만 수업에선 다른 방식으로 진행한다.

## 비로그인 방식 오픈 API 호출 예 [↗](http://surl.li/rsypp)

<http://surl.li/rsypp>

### NaverAPIClient

```
NaverAPIClient.java x
3  import com.humaneu.searchfood.naver.dto.SearchRegionRequestDto;
4  import com.humaneu.searchfood.naver.dto.SearchRegionResponseDto;
5  import org.springframework.http.MediaType;
6  import org.springframework.stereotype.Component;
7
8  import org.springframework.http.HttpHeaders;
9
10
11
12  @Component //bean으로 등록하기 위해서 넣어준다
13  public class NaverAPIClient {
14
15      //1. 지역검색 API call method(request dto, response dtd)
16
17      public SearchRegionResponseDto searchRegion(
18          SearchRegionRequestDto searchRegionRequestDto) {
19
20          //API Call Library -> HttpURLConnection, WebClient, RestTemplat
21          //(1) Header 설정
22          HttpHeaders headers = new HttpHeaders();
23          headers.set("X-Naver-Client-Id", "LAcpTCPL_DIKVnoSB050");
24          headers.set("X-Naver-Client-Secret", "A8mWZtCs4F");
25          headers.setContentType(MediaType.APPLICATION_JSON);
26
27          return null;
28      }
29  }
```

도메인 주소

```
GET  https://openapi.naver.com/v1/search/local.json?query=커피
```

```
NaverAPIClient.java x
// (2) Request 설정
URI uri = UriComponentsBuilder.fromUriString("https://openapi.naver.com")
    .path("/v1/search/local.json") UriComponentsBuilder
    .queryParams(name: "query", searchRegionRequestDto.getQuery() )
    //.queryParams("display", 1)
    .encode()
    .build() UriComponents
    .toUri();
```

## 지역검색 테스트

```
SearchRegionRequestDto.java x
5 usages
6 public class SearchRegionRequestDto {
7     @Getter
8     @Setter
9     private String query;
10    no usages
11    private Integer display = 1;
12    no usages
13    private Integer start = 1;
14    no usages
15    private String sort = "random";
16 }

NaverAPIClient.java x SearchfoodApiApplicationTests.java x main.js x
17
20 @Test //네이버 지역검색 OpenAPI 테스트
21 void naverSearchRegionAPITest(){
22     String paramQuery = "커피";
23
24     //네이버 지역검색 OpenAPI Call method 테스트
25     SearchRegionRequestDto searchRegionRequestDto = new SearchRegionRequestDto();
26     searchRegionRequestDto.setQuery(paramQuery);
27
28     SearchRegionResponseDto searchRegionResponseDto
29         = naverAPIClient.searchRegion(searchRegionRequestDto);
30     System.out.println("네이버 지역검색 OpenAPI response json" + searchRegionResponse);
31 }
```

```
//(2) Request 설정
URI uri = UriComponentsBuilder.fromUriString("https://openapi.naver.com/v1/search/image.json")
    .path("/v1/search/image.json")
    .queryParams("name", "query", "searchImage")
    //queryParams("display", 1)
    .encode()
    .build()
    .toUri();
```

궁금하면 ctrl + B / 사용법이니 그냥 외우자

<https://developers.naver.com/>

비번 재발급을 받을경우에 NaverAPIClient에 다 바꿔줘야하는 번거로움이 있다

## 애플리케이션 정보

Client ID	LAcPTCPI_DIKVnoSBO5O
Client Secret	A8mWZtCs4F
	<a href="#">재발급</a>

반복되는 코드만 리팩토링 해준다

반복전 코드: **NaverAPIClient**

```
package com.humaneu.searchfood.naver;

import com.humaneu.searchfood.naver.dto.SearchImageRequestDto;
import com.humaneu.searchfood.naver.dto.SearchImageResponseDto;
import com.humaneu.searchfood.naver.dto.SearchRegionRequestDto;
import com.humaneu.searchfood.naver.dto.SearchRegionResponseDto;
import org.springframework.http.*;
import org.springframework.stereotype.Component;

import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

import java.net.URI;

@Component //bean으로 등록하기 위해서 넣어준다
public class NaverAPIClient {

    //1. 지역검색 API call method(request dto, response dtd)
    public SearchRegionResponseDto searchRegion(
        SearchRegionRequestDto searchRegionRequestDto) {

        //API Call Library -> HttpURLConnection, WebClient, RestTemplate,
        Retry ect...(여기선 RestTemplate)
        //(1) Header 설정
        HttpHeaders headers = new HttpHeaders();
        headers.set("X-Naver-Client-Id", "LAcPTCP1_DIKVnoSB050");
        headers.set("X-Naver-Client-Secret", "A8mWZtCs4F");
        headers.setContentType(MediaType.APPLICATION_JSON);

        //(2) Request 설정
        URI uri = UriComponentsBuilder
            .fromUriString("https://openapi.naver.com") // 도메인 주소
            .path("/v1/search/local.json")
            .queryParams("query", searchRegionRequestDto.getQuery() )
            //.queryParams("display", 1)
            .encode()
            .build()
            .toUri();

        //(3, 4) Response 설정, 실제 API Call
        HttpEntity httpEntity = new HttpEntity(headers);
        ResponseEntity<SearchRegionResponseDto> responseRestTemplate
            = new RestTemplate().exchange(
                uri
                , HttpMethod.GET
                , httpEntity
                , SearchRegionResponseDto.class);

        return responseRestTemplate.getBody();
    }
}
```

```

    }

    public SearchImageResponseDto searchImage(
        SearchImageRequestDto searchImageRequestDto) {

        //API Call Library -> HttpURLConnection, WebClient, RestTemplate,
        Retry ect...(여기선 RestTemplate)
        //(1) Header 설정
        HttpHeaders headers = new HttpHeaders();
        headers.set("X-Naver-Client-Id", "LAcPTCPl_DIKVnoSB050");
        headers.set("X-Naver-Client-Secret", "A8mWZtCs4F");
        headers.setContentType(MediaType.APPLICATION_JSON);

        //(2) Request 설정
        URI uri =
        UriComponentsBuilder.fromUriString("https://openapi.naver.com") //
        도메인 주소

        .path("/v1/search/image.json")
        .queryParams("query", searchImageRequestDto.getQuery() )
        //.queryParams("display", 1)
        .encode()
        .build()
        .toUri();

        //(3, 4) Response 설정, 실제 API Call
        HttpEntity httpEntity = new HttpEntity(headers);
        ResponseEntity<SearchImageResponseDto> responseRestTemplate
            = new RestTemplate().exchange(
                uri, HttpMethod.GET, httpEntity,
                SearchImageResponseDto.class);

        return responseRestTemplate.getBody();
    }
}

```

## 리팩토링된 코드(반복삭제)

내가한 것

```
2 usages
private HttpEntity getHttpEntity(){
    HttpEntity httpEntity = new HttpEntity(getHttpHeaders());

    return httpEntity;
}
```

정답

```
2 usages
private HttpEntity getHttpEntity(){
    return new HttpEntity(getHttpHeaders());
}
```

```
@Component //bean으로 등록하기 위해서 넣어준다
public class NaverAPIClient {

    //1. 지역검색 API call method(request dto, response dtd)
    public SearchRegionResponseDto searchRegion(
        SearchRegionRequestDto searchRegionRequestDto) {

        //API Call Library -> HttpURLConnection, WebClient, RestTemplate,
        Retry ect...(여기선 RestTemplate)
        //(1) Header 설정
        //(2) Request 설정
        //(3, 4) Response 설정, 실제 API Call
        ResponseEntity<SearchRegionResponseDto> responseRestTemplate
            = new RestTemplate().exchange(getURI("/v1/search/image",
            searchRegionRequestDto.getQuery()),
            HttpMethod.GET,
            getHttpEntity(),
            SearchRegionResponseDto.class
        );

        return responseRestTemplate.getBody();
    }

    public SearchImageResponseDto searchImage(
        SearchImageRequestDto searchImageRequestDto) {

        //API Call Library -> HttpURLConnection, WebClient, RestTemplate,
        Retry ect...(여기선 RestTemplate)
        //(1) Header 설정
        //(2) Request 설정
        //(3, 4) Response 설정, 실제 API Call
```

```

        ResponseEntity<SearchImageResponseDto> responseRestTemplate
            = new RestTemplate().exchange(getURI("/v1/search/image",
searchImageRequestDto.getQuery())
            , HttpMethod.GET
            , getHttpEntity()
            , SearchImageResponseDto.class
            );

        return responseRestTemplate.getBody();
    }

    private HttpHeaders getHttpHeaders() {
        HttpHeaders headers = new HttpHeaders();
        headers.set("X-Naver-Client-Id", "LAcPTCPl_DIKVnoSB050");
        headers.set("X-Naver-Client-Secret", "A8mWZtCs4F");
        headers.setContentType(MediaType.APPLICATION_JSON);

        return headers;
    }

    private URI getURI(String path, String query) {
        return UriComponentsBuilder
            .fromUriString("https://openapi.naver.com") // 호출할 서버
            .path(path) // 호출할 서버
            .queryParams("query", query) // 호출할 query
            .encode() // 한글처리
            .build()
            .toUri();
    }

    private HttpEntity getHttpEntity(){
        return new HttpEntity(getHttpHeaders());
    }
}

```

<https://st-lab.tistory.com/153>

3월20일 2단계 9498 다시확인하기 <https://st-lab.tistory.com/22>



### 3월 21일 목요일

- 메소드화(논리적), DB대신 **List** 자바메모리에 **wish**정보, **Service**, **ApiController**
- **js** [검색시 출력값 **html**에 넣기, **Enter** 검색버튼 실행, 정규 표현식을 사용해 태그 없애주기

→ **Vue** 사용법

- 위시리스트 추가 **wishlist**, **Map** 자바메모리

```
//A
// ResponseEntity<SearchRegionResponseDto> responseRestTemplate

//B
// new RestTemplate().exchange(getURI("/v1/search/image", searchRegionRequestDto.getQuery())
// , HttpMethod.GET
// , getHttpEntity()
// , SearchRegionResponseDto.class

//C
// responseRestTemplate

// A = B, B = C -> A = C
// A = C, B = C -> A = B
```

	function a() // 로직1
int a = 1;	function b() // 로직2
// 로직1	int a = 1;
// 로직2	a() b()
// 로직 3	// 로직 3
a = C	a = c

----->>>> 메소드화 시킨다 로직을 밖으로 뺀다

문법과 논리 그리고 문법이 중요하다

## 어제 이용한 API 검색 수

내 애플리케이션

search-food

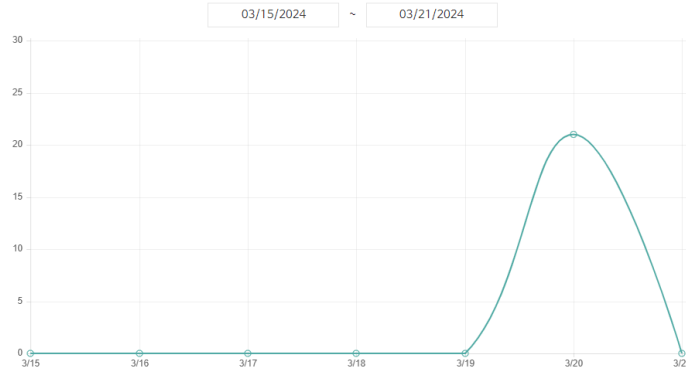
애플리케이션 등록

API 재휴 신청

계정 설정

search-food

개요	API 설정	멤버관리	로그인 통계	<u>API 통계</u>	Playground (Beta)
----	--------	------	--------	---------------	-------------------



모든 transaction은 돈이기 때문에 필요한 정보만 나오게 백엔드 설계를 해야한다  
네이버는 모든 정보가 나온다.

```
{
  "lastBuildDate": "Thu, 21 Mar 2024 09:39:32 +0900",
  "total": 3,
  "start": 1,
  "display": 3,
  "items": [
    {
      "title": "애플하우스",
      "link": "",
      "category": "분식>떡볶이",
      "description": "",
      "telephone": "",
      "address": "서울특별시 동작구 사당동 138-4 2층",
      "roadAddress": "서울특별시 동작구 동작대로27다길 29 2층",
      "mapx": "1269806796",
      "mapy": "374863013"
    }
  ]
}
```

☆ 화면에 출력될 6개의 값만이 필요하다

타이틀 ✓

카테고리 a

지번주소 3

도로명주소 4

홈페이지 3

```

@Data
public class WishListDto {

    private String title;        //검색결과 제목
    private String category;     //검색결과 카테고리
    private String jibunAddress; //검색결과 지번주소
    private String roadAddress;  //검색결과 도로명주소
    private String homepageLink; //검색결과 홈페이지링크
    private String imageLink;    //검색결과 이미지링크
}

```

6개 값만 DTO에 넣어준다

NaverAPIClient 와 연결되는 Service를 만든다 이 Service는 @Test 코드를 가져다 쓸 수 있다

```

@Test //네이버 지역검색 OpenAPI 테스트
void naverSearchRegionAPITest(){
    String paramQuery = "커피";

    //네이버 지역검색 OpenAPI Call method 테스트
    SearchRegionRequestDto searchRegionRequestDto = new SearchRegionRequestDto();
    searchRegionRequestDto.setQuery(paramQuery);

    SearchRegionResponseDto searchRegionResponseDto
        = naverAPIClient.searchRegion(searchRegionRequestDto);
    System.out.println("네이버 지역검색 OpenAPI response json" + searchRegionResponseDto);
}

```

```

@Service
public class WishListService {

    1 usage
    @Autowired
    private NaverAPIClient naverAPIClient;

    no usages
    public WishListDto search(String paramQuery){
        WishListDto wishListDto = new WishListDto();

        //1. NaverAPI 지역검색 호출해 dto 값 매핑
        SearchRegionRequestDto searchRegionRequestDto = new SearchRegionRequestDto();
        searchRegionRequestDto.setQuery(paramQuery);

        SearchRegionResponseDto searchRegionResponseDto
            = naverAPIClient.searchRegion(searchRegionRequestDto);
        List<SearchRegionResponseDto.SearchRegionItem> searchRegionItemList
            = searchRegionResponseDto.getItems();

        return wishListDto;
    }
}

```

List는 SearchRegionResponseDto에서 SearchRegionItem을 가지고 온다

```

"title": "애플하우스",
"link": "",
"category": "분식>떡볶이",
"description": "",
"telephone": "",
"address": "서울특별시 동작구 사당동 138-4 2층",
"roadAddress": "서울특별시 동작구 동작대로27다길 29 2층",
"mapx": "1269806796",
"mapy": "374863013"

```

```

public WishListDto search(String paramQuery){
    WishListDto wishListDto = new WishListDto();

    //1. NaverAPI 지역검색 호출해 dto 값 매핑
    SearchRegionRequestDto searchRegionRequestDto = new SearchRegionRequestDto();
    searchRegionRequestDto.setQuery(paramQuery);

    SearchRegionResponseDto searchRegionResponseDto
        = naverAPIClient.searchRegion(searchRegionRequestDto);
    List<SearchRegionResponseDto.SearchRegionItem> searchRegionItemList
        = searchRegionResponseDto.getItems();
    if(searchRegionItemList != null && searchRegionItemList.size() > 0){
        SearchRegionResponseDto.SearchRegionItem searchRegionItem = searchRegionItemList.get(0);

        wishListDto.setTitle(searchRegionItem.getTitle());
        wishListDto.setCategory(searchRegionItem.getCategory());
        wishListDto.setJibunAddress(searchRegionItem.getAddress());
        wishListDto.setRoadAddress(searchRegionItem.getRoadAddress());
        wishListDto.setHomepageLink(searchRegionItem.getLink());
    }

    return wishListDto;
}

```

☆ 이번엔 이미지 넣기

```

//2. NaverAPI 이미지검색 호출해 dto 값 매핑
SearchImageRequestDto searchImageRequestDto = new SearchImageRequestDto();
searchImageRequestDto.setQuery(paramQuery);

SearchImageResponseDto searchImageResponseDto
    = naverAPIClient.searchImage(searchImageRequestDto);
List<SearchImageResponseDto.SearchImageItem> searchImageItemList
    = searchImageResponseDto.getItems();
if(searchImageItemList != null && searchImageItemList.size() > 0){
    SearchImageResponseDto.SearchImageItem searchImageItem =
searchImageItemList.get(0);

    wishListDto.setImageLink(searchImageItem.getLink());
}
return wishListDto;

```

## ☆ ApiController

```
@RequestMapping("/api")
@RestController
public class ApiController {

    1 usage
    @Autowired
    private WishListService wishListService;


    // 1. 검색 API(GET)
    no usages
    @GetMapping("/search")
    public WishListDto search(@RequestParam String searchQuery){
        WishListDto wishListDto = wishListService.search(searchQuery);
        return wishListDto;
    }
}
```

## 콘솔창에 나온다

```
search btn click
search response값
{title: '커피 컵의 흰색 배경 3d 렌더링된 평면!',
  ess: null, roadAddress: null, homepageLink:
  g', ...}
category: null
homepageLink: "https://png.pngtree.com/thu
imageLink: "https://png.pngtree.com/thumb_
jibunAddress: null
roadAddress: null
title: "커피 컵의 흰색 배경 3d 렌더링된 평면
[[Prototype]]: Object
```

### 1. 검색시 출력값 html에 넣기

**오류:** 검색시 해당 값이 오른쪽에 나와야하는데 오류가 났다.

커피	검색
	<p>커피 컵의 흰색 배경 3d 렌더링된 평면도, 커피 거품, 에스프레소, 커피 탭 배경 일러스트 및 사진 무료 다운로드 - Pngtree</p> <p>null</p> <p>null</p> <p>null</p> <p>홈페이지</p> <p>위시리스트 추가</p>

**이유:** local 로 넣어야한다(NaverAPIClient)

```

public SearchRegionResponseDto searchRegion(
    SearchRegionRequestDto searchRegionRequestDto) {

    //API Call Library -> HttpURLConnection, WebClient, RestTemplate, Ret
    //(1) Header 설정
    //(2) Request 설정
    //(3, 4) Response 설정, 실제 API Call
    ResponseEntity<SearchRegionResponseDto> responseRestTemplate
        = new RestTemplate().exchange(getURI("path: "/v1/search/image",
            , HttpMethod.GET
            , getHttpEntity()
            , SearchRegionResponseDto.class
            );

```

2. Enter 검색버튼 실행하기 <https://jh-tr.tistory.com/70>  
 key코드리스트 <https://docs-kr.zep.us/creator/reference/keycode>

```

$(function() {
    $("#searchBox").keypress(function(e) {
        //검색어 입력 후 엔터키 입력하면 조회버튼 클릭
        console.log('엔터');
        if(e.keyCode && e.keyCode == 13){
            $("#searchButton").trigger("click");
            console.log('엔터성공');
            return false;
        }
        //엔터키 막기
        if(e.keyCode && e.keyCode == 13){
            e.preventDefault();
        }
    });
});

```

3. 자바스크립트 정규 표현식을 이용해 이상한 태그 없애주기

```

"title": "개나리<b>아구짬</b> 본점",
//$('#title').replace(/(<([>])+/ig, "");???

```

VUE 사용해서 js를 쓸 수 있다 <https://vuejs.org/>

사용전 js / html - id 값으로 출력

```

$.get('/api/search?searchQuery=${query}', function(response){
    console.log('search response', response);

    $('#title').text(response.title);
    $('#category').text(response.category);
    $('#jibunAddress').text(response.jibunAddress);
    $('#roadAddress').text(response.roadAddress);
    $('#homepageLink').attr("href", response.homepageLink);
    $('#imageLink').attr("src", response.imageLink);

});

```

```

<div class="col-md-4">
  <ul class="list-group list-group-flush">
    <li class="list-group-item" id="title">타이틀</li>
    <li class="list-group-item" id="category">카테고리</li>
    <li class="list-group-item" id="jibunAddress">지번주소</li>
    <li class="list-group-item" id="roadAddress">도로명주소</li>
    <li class="list-group-item" id="homepageLink"><a href="#">홈페이지</a></li>
  </ul>

```

## ★★VUE 사용

### 1-1 검색시 출력값 html에 넣기

```

main.js x
1 //vue를 사용한 코드
2 var searchResult = new Vue({
3   el: '#search-result',
4   data: {
5     search_result: {},
6   },
7 });
8
9 //검색 버튼을 눌렀을 때 이벤트
10 $('#searchButton').click(function(){
11   console.log('search btn click');
12
13   const query = $('#searchBox').val(); //검색어
14   // 실제 backend에 /api/search 요청해서 데이터 가지고오기 backtick``문자
15   $.get(`/api/search?searchQuery=${query}`, function(response){
16     console.log('search response값', response);
17
18     searchResult.search_result = response;
19   });
20 });

```

```

28 <br>
29 <!-- 검색 결과영역 -->
30 <div class="row" id="search-result">
31 <!-- 검색 이미지 결과 화면 -->
32 <div class="col-md-8">
33 
34 </div>
35 <!-- 검색 텍스트 결과 버튼 -->
36 <div class="col-md-4">
37 <ul class="list-group list-group-flush">
38 <li class="list-group-item">{{ search_result.title }}</li>
39 <li class="list-group-item">{{ search_result.category }}</li>
40 <li class="list-group-item">{{ search_result.jibunAddress }}</li>
41 <li class="list-group-item">{{ search_result.roadAddress }}</li>
42 <li class="list-group-item"><a v-bind:href="search_result.homepageLink" target="_blank">홈페이지</a>
43 </ul>
44 <button type="button" class="btn btn-primary" style="position: absolute; bottom: 0px; ...">위시리스트</button>
45 </div>

```

## 2-1 검색 → Enter 실행

```

//검색장에서 enter 했을때 이벤트 처리
$('#searchBox').on('keyup', function(e) {
    if(e.keyCode == 13) {
        $('#searchButton').click();
    }
});

```

## 3-1 정규 표현식을 이용해 title 태그 삭제

```

const title = document.getElementById('wish-title');
title.innerHTML = searchResult.search_result.title.replace(/(<([^\>]+)>)/ig, '');

```

먼저 **title**의 **id** 값을 가변과 선언해준다

선언한 값을 다시 **HTML**에 넣어줄때 **replace** 를 적용하고 정규표현식을 넣으면 된다

☆위시리스트 추가하기

**DB**대신 **Map** 자바메모리에 **wish**정보 >> **WishListRepository** >> **WishListVO**

**WishListRepository** 에 위시리스트 저장, **create** 후 **select all**을 해줘야한다

**WishListRepository** - **@Repository** 어노테이션이 부여된 클래스는 **Spring**에 의해 자동으로 빈으로 관리되며, 데이터 액세스 레이어의 구현을 정의합니다. 이 클래스는 주로 **JDBC, JPA, Hibernate** 등을 사용하여 데이터베이스와의 상호작용을 처리하는데 사용됩니다.

**WishListService** : 같은 코드지만 **void, return**값 **X** > **WishListVO** return



```
1 usage
public void addWish(WishListDto wishListDto){
    // Storage(DB, Memory, etc)에 wish정보 저장
    wishListRepository.wishSave(wishListDto);
}
```

```
usage
public WishListVO addWish(WishListDto wishListDto){
    // Storage(DB, Memory, etc)에 wish정보 저장
    return wishListRepository.wishSave(wishListDto);
}
```

**ApiController: void**를 사용해 **return** 값이 없다

```
// 2. 위시리스트 추가 API(POST)
no usages
@PostMapping("/wishadd")
public void wishAdd(@RequestBody WishListDto wishListDto){
    wishListService.addWish(wishListDto);
}
```

**@RequestBody** 클라이언트로부터 전송된 JSON 또는 XML과 같은 데이터를 자바 객체로 변환할 때 주로 사용. 클라이언트가 JSON 형식으로 다음과 같은 데이터를 POST 요청

```
search btn click
search response값
{title: '개나리<b>아구짬</b> 본점', category: '
7-8 1층', homepageLink: 'https://app.catchtabl
with btn click
```

위시리스트 추가 완료

위시리스트 추가하면 아래 목록에 보이고 다시 홈페이지를 재실행 할 경우 추가된 위시리스트의 메모리는 자동 삭제 된다.

Create

Select ALL

구현해 보기

Update -> 방문추가 -> 기존 방문횟수 +1

Delete -> 위시리스트 삭제 -> wishlist에서 1개 삭제 + 화면에서도 삭제

위시리스트 삭제 - 풀이오답(X)

- **WishListRepository**

```
public WishListVO wishDelete(Integer id){
    WishListVO wishListVO = new WishListVO();
    wishListVO.setId(1);
    wishListVOList.remove(wishListVO);
    return wishListVO;
}
```

- WishListService

```
1 usage
public WishListVO deleteWish (Integer id){
    return wishListRepository.wishDelete(id);
}
```

- ApiController

```
no usages
@DeleteMapping("/wishdelete")
public WishListVO wishdelete(
    @RequestBody(Integer id){
    return wishListService.deleteWish(id);
}
```

- Main.html

```
- 위시리스트 테스트 화면 -->
<div class="col-md-4">
    <ul class="list-group list-group-flush">
        <li class="list-group-item" style="display:none;">id: {{ wish.id }}</li>
        <li class="list-group-item">제목: {{ wish.title }}</li>
        <li class="list-group-item">카테고리: {{ wish.category }}</li>
        <li class="list-group-item">지번주소: {{ wish.jibunAddress }}</li>
        <li class="list-group-item">도로명주소: {{ wish.roadAddress }}</li>
        <li class="list-group-item">방문여부: {{ wish.visitIs }}</li>
        <li class="list-group-item">방문횟수: {{ wish.visitCount }}</li>
        <li class="list-group-item">방문날짜: {{ wish.lastVisitDate }}</li>
        <li class="list-group-item"><a v-bind:href="wish.homepageLink" target="_blank">홈페이지</a></li>
    </ul>
    <button type="button" class="btn btn-primary" style="width: 100%;">방문추가</button>
    <br><br>
    <button type="button" onclick="wishDelete();" class="btn btn-primary" style="width: 100%;">리스트삭제</button>
</div>
```

- JS

```

// 위시리스트 삭제 버튼 눌렀을 때
function wishDelete(id){
    console.log('삭제클릭ㅋㅋ');
    $.ajax({
        type: 'delete',
        url: '/api/wishdelete',
        data: JSON.stringify(id),
        contentType: 'application/json',
        success: function(response, status, xhr) {
            console.log('위시리스트 삭제 완료', response);

            getWishList();
        },
        error: function(request, status, error) {
            alert('위시리스트 삭제 실패하였습니다');
        }
    });
}

```

>> 삭제 버튼 클릭시, JS의 **error alert**가 뜬다. 삭제안됨

**3월22일 금요일** - 위시리스트 삭제, 방문추가폴이, **SCM @EnableScheduling**  
어노테이션

★ 리스트 삭제폴이

1. Vue에 **methods**와 함수이름을 넣어준다

```

class="btn btn-primary" style="width: 100%;">
@Click="deleteWishList();" class="btn btn-pri

```

```

10 var wishListResult = new Vue({
11   el: '#wish-list-result',
12   data: {
13     wish_list: {},
14   },
15   methods: {
16     deleteWishList(id) {
17
18     },
19   },
20 });

```

## 2. 콘솔로 찍어서 실행되는지 본다

```
var wishListResult = new Vue({
  el: '#wish-list-result',
  data: {
    wish_list: {},
  },
  methods: {
    deleteWishList(id) {
      console.log('wishlist delete id', id);
    },
  },
});
```

- Repository = mapper

```
1 usage
public boolean wishDelete(Integer id){
    boolean isDelete = wishListVOList.re(id);
    return isDelete;
}
```

remove(Object o)  
remove(int index)

**Integer** = 클래스값 **objective type**

**int** = 원시적인 값 **primitive type**

이므로 순수한 값인 **int**로 가져오고 **object**가 아닌 **index**값으로 가져와야한다

>> 실행시 오류 다시 아래로 고쳐준다

```
1 usage
public boolean wishDelete(int id){
    Integer a = null;
    int b = null;

    WishListVO wishListVO = wishListVOList.remove(id);
    return wishListVO != null ? true : false;
}
```

> **Integer & int** 의 차이점 **null** 사용유무

- Repository = mapper 수정

```
1 usage
public boolean wishDelete(int id){
    WishListVO wishListVO = wishListVOList.remove(id);
    return wishListVO != null ? true : false;
}
```

⚠ Third-party cookie will be blocked. Learn more in the Issues tab.

wishlist delete id 0

위시리스트 삭제 완료 true

wishall response ▶ `{{...}}`

wishlist delete id 1

❌ ▶ DELETE `http://localhost:10002/api/wishdelete` 500 (Internal Server Error)

위처럼 **return**값을 수정했지만, 첫 위시리스트는 삭제되고 다음부터는 삭제가 안된다.

이유는 첫 **id** 값을 삭제해주면 **JS**에서 실행되는 **deleteWishList(id)** 함수에서

**getWishList()**; 를 작동해 다시 정리된 위시리스트를 가지고온다, 즉 삭제되지 않았던

리스트 인덱스값이 **1**이 아닌 **0**이 된다. **.remove(id)**를 넣어주면 **id**값과 저장된 **id**의 인덱스

값이 일치하지 않기 때문에 에러가난다

wishlist delete id 0

❌ ▶ DELETE `http://localhost:10002/api/wishdelete/` 404 (Not Found)

- **Repository = mapper** 다시 수정한다

```
public boolean wishDelete(int id){ //false 와 true를 반환, int로 설정
    boolean isDelete = false; // true를 막아준다
    for(WishListVO wishListVO : wishListVOList) { ///List 자바메모리에 wish정보
        if(wishListVO.getId() == id) {
            wishListVOList.remove(wishListVO); //객체를 삭제해준다 object
            isDelete = true; //if가 맞다면 true 설정

            break; //for문 정지시켜주기
        }
    }
    return isDelete;
}
```

- **Controller** 와 **Ajax**의 **url**이 같아야된다, 같지 않으면 오류발생

```
@DeleteMapping("/wishdelete")
public boolean wishdelete(@RequestBody Integer id) {
    boolean wishListDto = wishListService.deleteWishList(id);
    return wishListDto;
}
```

//실제 DB에서 삭제

```
$.ajax({
    type: 'delete',
    url: '/api/wishdelete',
```

★★위시리스트 삭제 전체 코드 내코드

1. **main.html** `{{wish.id}}` >> `(wish.id)`

```
<button type="button" @Click="deleteWishList(wish.id);" class="btn btn-primary" style="width: 100%;">리스트삭제</button>
```

## 2. WishListDto

```
public Integer id;
public int visitCount;
```

## 3. WishListService (int로 선언해도된다)

```
public boolean deleteWish (Integer id){
    boolean Id = wishListRepository.wishDelete(id);
    return Id;
}
```

## 4. WishListRepository

```
public boolean wishDelete(int id){ //false 와 true를 반환, int로 설정
    boolean isDelete = false;    // true를 막아준다
    for(WishListVO wishListVO : wishListVOList) { ///List 자바메모리에 wish정보
        if(wishListVO.getId() == id) {
            wishListVOList.remove(wishListVO); //객체를 삭제해준다 object
            isDelete = true;    //if가 맞다면 true 설정

            break;    //for문 정지시켜주기
        }
    }
    return isDelete;
}
```

## 5. ApiController

```
@DeleteMapping("/wishdelete")
public boolean wishdelete(@RequestBody Integer id){
    boolean wishListDto = wishListService.deleteWish(id);
    return wishListDto;
}
```

## 6. JS

```
var wishListResult = new Vue({
    el: '#wish-list-result',
    data: {
        wish_list: {},
    },
    methods: {
        deleteWishList(id) {
            console.log('wishlist delete id', id);

            //실제 DB에서 삭제
            $.ajax({
                type: 'delete',
                url: '/api/wishdelete' ,
                data: JSON.stringify(id),
```

```

        contentType: 'application/json',
        success: function(response, status, xhr) {
            console.log('위시리스트 삭제 완료', response);    //
WishListVO

            getWishList();
        },
        error: function(request, status, error) {
            alert('위시리스트 삭제 실패하였습니다');
        }
    });

    },
    },
});

```

## ★★방문추가 전체 코드 - 수하코드

### 1. main.html

```

<button type="button" @Click="addVisit(wish.id);" class="btn btn-primary"
style="width: 100%;">방문추가</button>

```

### 2. WishListDto

```

private boolean visitIs;    //방문여부
private int visitCount;    //방문 횟수
private LocalDateTime lastVisitDate;    //방문시간

```

### 3. WishListService

```

public boolean visitCount(Integer id){
    boolean addId = wishListRepository.addVisit(id);
    return addId;
}

```

### 4. WishListRepository

```

public boolean addVisit(Integer id){
    boolean isAdd = false;    // true를 막아준다

    for(WishListVO wishListVO : wishListVOList) {    ///List 자바메모리에 wish정보

```

```

        if(wishListVO.getId() == id) {
            wishListVO.setVisitIs(true);
            wishListVO.setVisitCount(wishListVO.getVisitCount() + 1);
            wishListVO.setLastVisitDate(LocalDateTime.now());

            isAdd = true;    //if가 맞다면 true 설정

            break;    //for문 정지시켜주기
        }
    }
    return isAdd;
}

```

## 5. ApiController

```

@PostMapping("/addVisit/{id}")
public boolean addVisit(@PathVariable("id") Integer id){
    boolean isAddVisitSuccess = wishListService.visitCount(id);
    return isAddVisitSuccess ;
}

```

## 6. JS

```

var wishListResult = new Vue({
    el: '#wish-list-result',
    data: {
        wish_list: {},
    },
    methods: {

        addVisit(id){
            console.log('addvisit id', id);
            addVisit(id)
        },
    },
});
// 방문추가 버튼 눌렀을 때
function addVisit(id){
    console.log('visited++' + id);
    $.ajax({
        type: 'post',
        url: '/api/addVisit/' + id,
        /* @PathVariable 사용하면 아래의 data, contentType 사용하면 필요 x
           만약 @PathVariable, @RequestBody 둘 다 사용할거면 아래의 2줄
           있어야 함. */
        /*data: JSON.stringify(id),
        contentType: 'application/json',*/

        success: function(response, status, xhr) {
            console.log('방문추가 완료', response);    // WishListVO

            getWishList();
        },
        error: function(request, status, error) {

```



```

        alert('방문추가 실패하였습니다');
    }
});
}

```

선생님폴이

★★위시리스트 삭제 & 방문횟수 추가 > 깃허브 > **Springboot API**

[ApiController 코드]

View에서 넘어오는 파라미터 값을 적어줘야 함.

wishList에서 출력된 **title**의 불필요한 태그들을 삭제하기

```

public boolean addVisit(@PathVariable("id") Integer id) {
    boolean isAddVisitSuccess = wishListService.addVisit(id);
    isAddVisitSuccess;
}

// 5. 위시리스트 삭제 API(DELETE)
no usages new *
@DeleteMapping("/wishdelete")
public boolean wishdelete(@RequestBody Integer id){
    boolean idDeleteWishSuccess = wishListService.deleteWish(id);
    return idDeleteWishSuccess;
}

```

이건 필요한 url과 어노테이션만 적으면 된다.

[main.js 코드 - 수정]

```

// 수정(방문 추가)
function addVisit(id) {
    console.log('add visit count id', id);

    $.ajax({
        type: 'post',
        url: '/api/addVisit/' + id,
        /* url: @PathVariable 사용하면 아래의 data, contentType 사용하면 필요 X */
    });
}

```

url 뒤에 View에서 받아올 값을 추가해야 함.

제목: 개 나리 <b>아구찜</b> 본점

```

getWishList();
},
error: function(request, status, error) {
    alert('방문횟수 추가 실패', error);
}
});

```

<Data, contentType>

만약, @PathVariable만 사용할 것이면 굳이 적을 필요 없다.  
그러나 @PathVariable, @RequestBody를 동시에 사용한다면, 적어야 한다.

[main.js 코드 - 삭제]

```

// 위시리스트 삭제
function deleteWishList(id) {
    console.log('wishList delete id', id);

    //실제 DB에서 삭제
    $.ajax({
        type: 'delete',
        url: '/api/wishdelete',
        data: JSON.stringify(id),
        contentType: 'application/json',
        success: function(response, status, xhr) {
            console.log('위시리스트 삭제 완료', response); // WishListVO
        },
        error: function(request, status, error) {
            alert('위시리스트 삭제 실패하였습니다');
        }
    });
}

```

필요한 url만 적으면 된다.

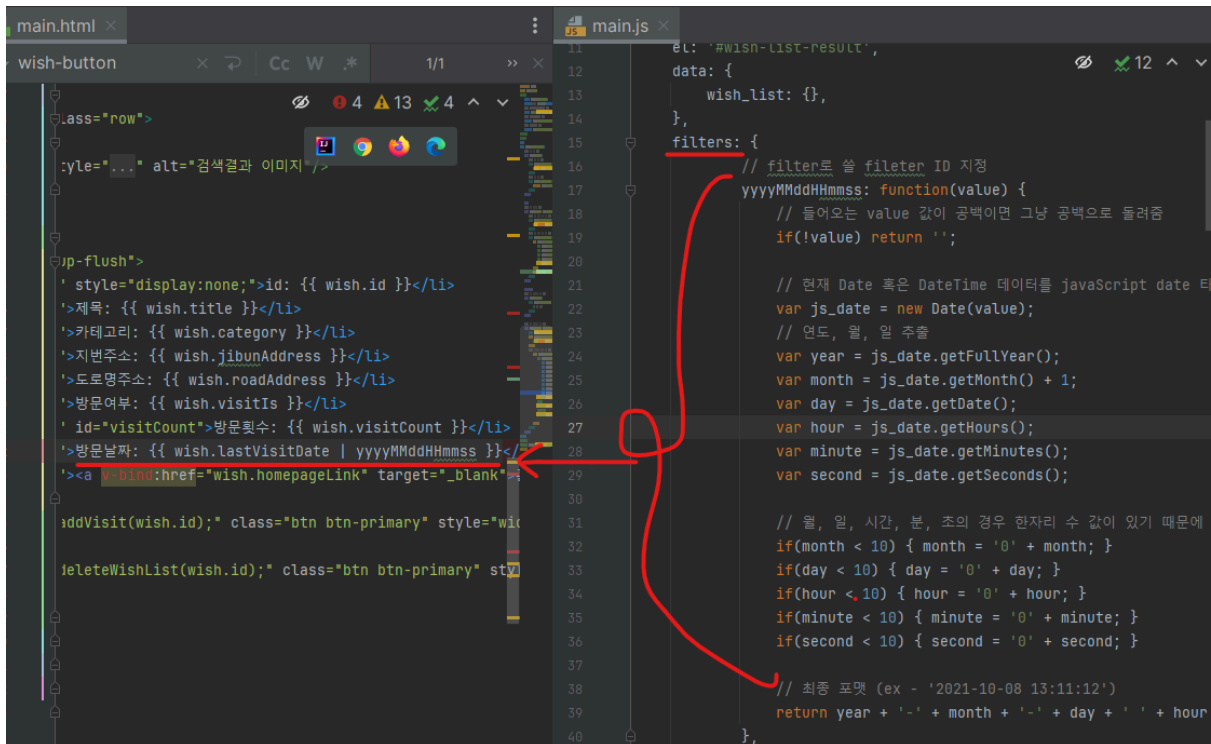
<Data, contentType>  
@RequestBody는 적어야 한다.

```
1 package com.humanedu.searchfood.utils;
2 import org.apache.logging.log4j.util.Strings;
3 import java.util.regex.Pattern;
4
5 public class StringUtils {
6     1 usage
7     public static String removeTags(String str) {
8         if(Strings.isEmpty(str)) {
9             return str;
10        }
11
12        return Pattern.compile(regex: "<.+?>").matcher(str)
13            .replaceAll(replacement: "");
14    }
15 }
```

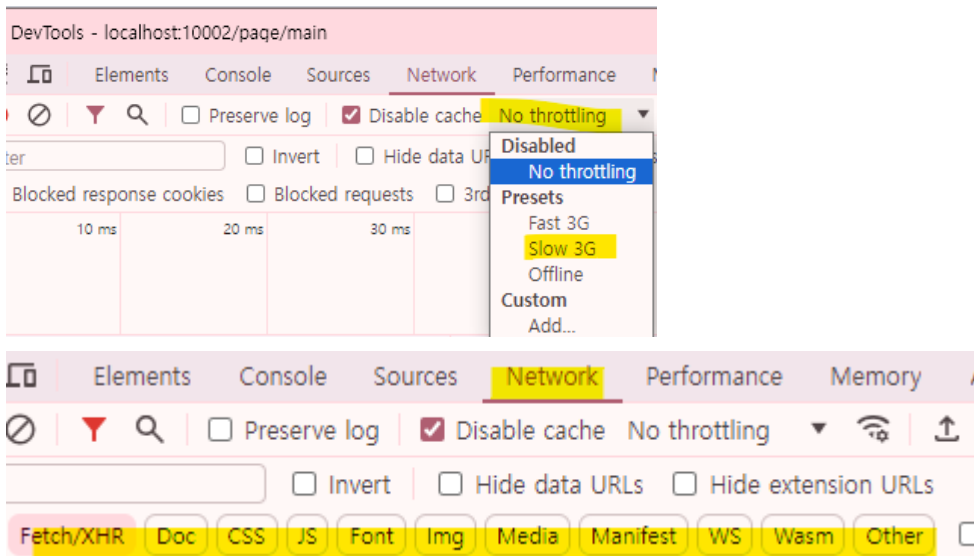
2.

```
18 @ 1 usage
19 public WishListVO wishSave(WishListDto wishListDto) {
20     WishListVO wishListVO = new WishListVO();
21     wishListVO.setId(generateNewId());
22     wishListVO.setTitle(StringUtils.removeTags(wishListDto.getTitle()));
23     wishListVO.setCategory(wishListDto.getCategory());
24     wishListVO.setJibunAddress(wishListDto.getJibunAddress());
25     wishListVO.setRoadAddress(wishListDto.getRoadAddress());
26     wishListVO.setHomepageLink(wishListDto.getHomepageLink());
27 }
```

방문날짜: 2024-03-22T14:51:07.556447  
wishList에서 출력된 방문날짜 수정



개발자도구 **F12**에서 속도감



SCM

@EnableScheduling 어노테이션  
Cron 표현식을 사용해 작업을 예약

```

@EnableScheduling
@SpringBootApplication
public class SearchfoodApiApplication {

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

}

```

StringUtils.java

```

@Component
public class StringUtils {

    private int cnt = 0;

    private int cnt2 = 0;

    public static String removeTags(String str) {
        if (Strings.isEmpty(str)) {
            return str;
        }
        return Pattern.compile(regex: "<.+?>").matcher(str)
            .replaceAll(replacement: "");
    }

    @Scheduled(fixedDelay = 1000) // 1초마다 실행
    public void scheduleTest1() {
        System.out.println("1초마다 실행 하는 로그 " + (cnt++));
    }

    // 초(0-59) 분(0-59) 시간(0-23) 일(1-31) 월(1-12) 요일(0-6) (0:
    @Scheduled(cron = "0/2 * * * *", zone = "Asia/Seoul")
    public void scheduleTest2() {
        System.out.println("2초마다 실행 하는 로그 " + (cnt2++));
    }

}

```