REPORT

Lab2 – Earthquake / Volcano Chart

과목: 문제해결을 위한 자바활용

교수명: 박경신 교수님

학과: 컴퓨터공학과

학번: 32222174

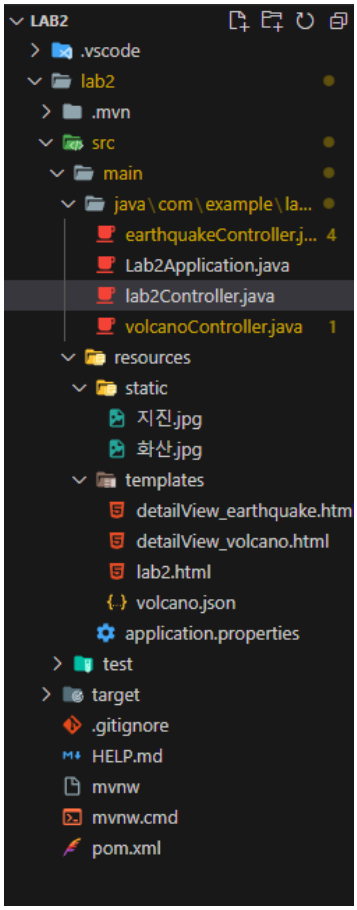
이름: 서영빈

제출일: 2024-04-07

목차

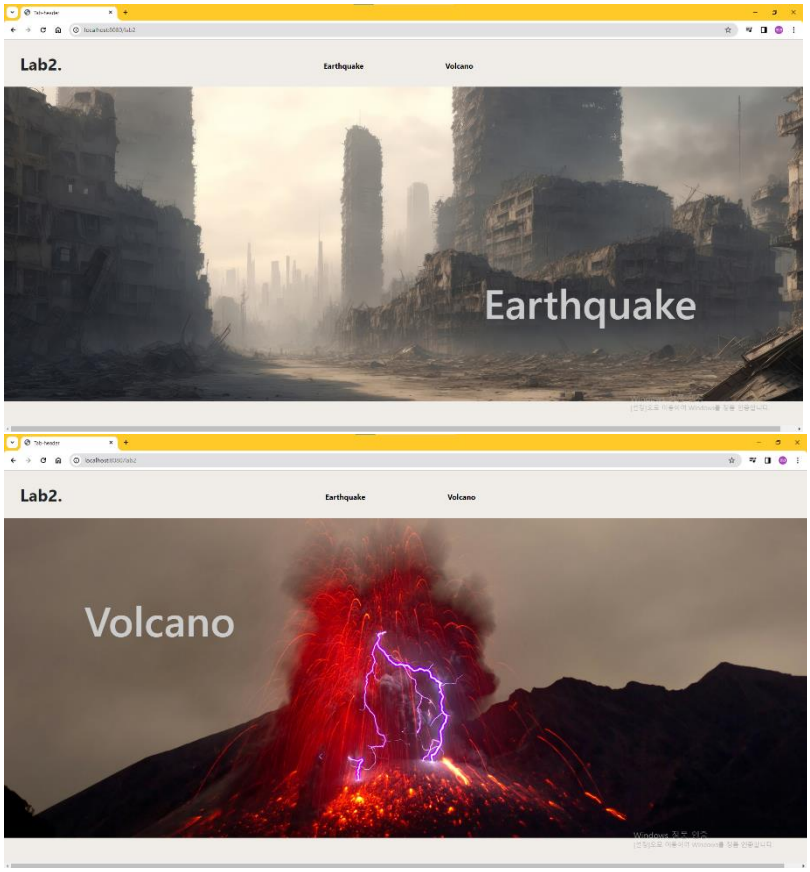
.

1. 프로젝트 구조

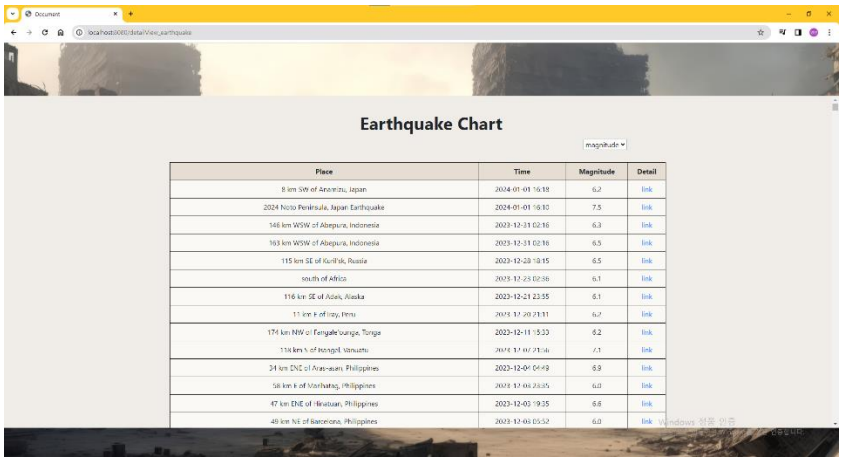


프로젝트는 크게 세 페이지로 이루어져 있다. 탭 헤더가 있는 lab2.html와 컨트롤러인 lab2Controller.java. 지진 정보가 있는 deatilView_earthquake.html와 이 html의 컨트롤러인 earthquakeController.java, detailView_volcano.html와 volcanoController.java가 있다. Static 디렉토리에는 배경과 표지로 쓸 지진, 화산 이미지가 있다. Volcano.json 파일은 volcano 데이터의 방대한 양 때문에 api로 받아올 수 없어서 직접 넣었다.

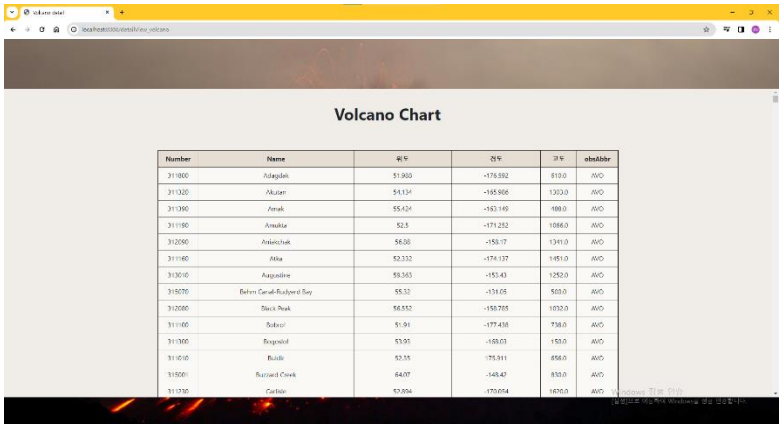
2. 실행 화면



상단에 탭 헤더가 있고 글자를 클릭하면 각 detailView로 이동하게 되어 있다. 이미지는 3초마다 바뀌는데 이미지를 클릭해도 각 detailView로 이동하도록 설정했다.



Earthquake chart는 지진의 위치, 시간, 규모, 세부사항이 적혀 는 차트를 만들었다. 각 지진마다 link를 누르면 그 지진에 대한 자세한 정보가 담겨있는 사이트로 이동한다. Magniutude를 누르면 지진의 규모가 6.0, 7.0, 8.0 이상의 자료만 출력되도록 했다.



Volcano chart는 화산의 번호, 이름, 위도, 경고, 고도, 관찰하는 기관의 정보를 담고 있다.

3. 코드 설명

lab2Controller.java

lab2 > src > main > java > com > example > lab2 > lab2Controller.java > ...

```java
package com.example.lab2;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;


@Controller
public class lab2Controller {
  @GetMapping("/lab2")
  public String showTable (Model model){
      return "lab2";
  }
}

```

Lab2Controller.java

단순히 lab.html을 templates 디렉토리 안에 만들기 위한 컨트롤러이다.

```
<script>
let currentIndex = 0;
const images = document.querySelectorAll('.slideshow img');
const inImage1 = document.querySelector('.in-image1');
const inImage2 = document.querySelector('.in-image2');

function showImage(index) {
  images.forEach((image, i) => {
    image.style.display = i === index ? 'block' : 'none';
  });
  inImage1.style.display = index === 0 ? 'block' : 'none';
  inImage2.style.display = index !== 0 ? 'block' : 'none';
}

function nextImage() {
  currentIndex = (currentIndex + 1) % images.length;
  showImage(currentIndex);
}

showImage(currentIndex);
setInterval(nextImage, 3000);
</script>
```

Lab2.html

Lab2.html에서 시간에 따라 이미지를 변경시키기 위해 자바 스크립트를 사용했다.

currentIndex 변수는 현재 표시되고 있는 이미지의 인덱스를 나타내는 변수로 처음에는 0으로 설정된다. Images 변수는 querySelectorAll() 함수를 사용해 slideshow 클래스 내의 모든 이미지를 가진다.

showImage 함수는 이미지를 표시하는 함수이다. 인덱스를 받아 그 인덱스에 해당하는 이미지를 화면에 표시한다. 여기서 in-image1, in-image2 이미지 위에 써져야 하는 텍스트가 다르기 때문에 if문을 사용했다.

nextImage 함수는 다음 이미지를 표시하는 함수이다. 현재 인덱스를 증가시키고 이미지 배열의 길이로 나눈 나머지를 새로운 인덱스로 설정해 이미지가 끝까지 순환되도록 했다

setInterval() 함수를 3초마다 호출해 인덱스에 맞는 이미지로 전환된다.

```java
@GetMapping("/detailView_earthquake")
public String EarthquakeData(Model model) throws IOException{
  String url1 = "https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=1900-01-01&endtime=2024-01-02&minmagnitude=6.0";

  RestTemplate restTemplate = new RestTemplate();
  Map<String, Object> response = restTemplate.getForObject(url1, responseType:Map.class);

  List<Map<String, Object>> features = (List<Map<String, Object>>) response.get(key:"features");

  List<Map<String, Object>> earthquakedata = new ArrayList<>();

  for (Map<String, Object> feature : features) {
    Map<String, Object> properties = (Map<String, Object>) feature.get(key:"properties");
    String place = (String) properties.get(key:"place");
    long Unixtime = ((Number) properties.get(key:"time")).longValue();
    Double mag =((Number) properties.get(key:"mag")).doubleValue();
    String url1 = (String) properties.get(key:"url");

    String time=convertUnixTime(Unixtime);

    Map<String, Object> earthquake=Map.of(k1:"place",place,k2:"time",time,k3:"mag",mag,k4:"url",url1);
    earthquakedata.add(earthquake);
  }

  model.addAttribute(attributeName:"earthquakedata", earthquakedata);

  return "detailView_earthquake";
}
```

earthquakeController.java

https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=1900-01-01&endtime=2024-01-02&minmagnitude=6.

지진 데이터는 위 url에서 api 데이터를 json 형식으로 받아왔다. 5.0 이상의 데이터는 이 url에서 찾을 수 없어서 6.0 이상의 데이터를 사용했다. restTemplate를 사용해 url로 GET 요청을 보내고 응답을 map<string, object> 형태로 받아오고, 응답에서 features 키에 대한 하는 데이터 목록을 가져온다. For-each 문을 통해 가져온 데이터 중 필요한 데이터만 추출해 map을 만들고 earthquakedata 리스트에 저장한다. 리스트에 저장한 데이터를 earthquakedata라는 이름으로 모델에 추가한다.

```java
public String convertUnixTime(long unixTime) {
    Instant instant = Instant.ofEpochMilli(unixTime);
    ZoneId zoneId = ZoneId.systemDefault();
    LocalDateTime dateTime = LocalDateTime.ofInstant(instant, zoneId);

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd HH:mm");

    String formattedDateTime = dateTime.format(formatter);

    return formattedDateTime;
}
```

위 url에서는 시간이 Unix 시간으로 저장되어 있다. 유닉스 시간을 일반적인 날짜로 바꾸기 위해 converUnixTime() 함수를 만들었다.

```
<table>
  <thead>
    <tr>
      <th>Place</th>
      <th>Time</th>
      <th>Magnitude</th>
      <th>Detail</th>
    <tr>
  </thead>
  <tbody id="earthquake-table-body">
    <tr th:each="earthquake : ${earthquakedata}">
      <td th:text="${earthquake.place}">Unknown</td>
      <td th:text="${earthquake.time}">Unknown</td>
      <td th:text="${earthquake.mag}">Unknown</td>
      <td>
        <a th:href="${earthquake.url}">link</a>
      </td>
    </tr>
  </tbody>
</table>
```

detailView_earthquake .html

Earthquakedata 리스트에 있는 지진 데이터에 대한 테이블의 행을 만들고 Thymeleaf를 통해 earthquakedata 리스트 각 요소의 값을 가져온다. 값이 없으면 unknown을 표시한다. Detail 열에는 리스트의 url에 해당하는 링크를 연결시켰다.

```
<script>
  const magnitudeFilter = document.getElementById('magnitude-filter');
  const earthquakeRows = document.querySelectorAll('#earthquake-table-body tr');

  magnitudeFilter.addEventListener('change', function() {
    const selectedMagnitude = parseFloat(this.value);
    earthquakeRows.forEach(row => {
      const magnitude = parseFloat(row.cells[2].textContent);
      row.style.display = (selectedMagnitude === 0 || magnitude >= selectedMagnitude) ? '' : 'none';
    });
  });
</script>
```

사용자가 선택한 지진 규모에 따라 해당하는 행만 나타나게 하는 자바 스크립트 코드이다. 클래스가 earthquake-table-body인 요소 내 모든 tr(행)를 선택해 earthquakeRows 변수에 할당한다. 사용자가 선택한 값을 가져와서 숫자로 변환해 selectedMagnitude 변수에 할당한다. 사용자가 선택한 규모보다 작으면 행을 숨기고, 같거나 크면 표시한다.

```java
public class volcanoController {
  @GetMapping("detailView_volcano")
  public String VolcanoData(Model model) throws IOException {
    ObjectMapper objectMapper = new ObjectMapper();

    ClassPathResource resource = new ClassPathResource(path:"templates/volcano.json");

    Map<String, Object>[] data = objectMapper.readValue(resource.getInputStream(), Map[].class);

    List<Map<String, Object>> volcanodata = new ArrayList<>();

    for (Map<String, Object> item : data) {
      String vnum=(String) item.get(key:"vnum");
      String vn = (String) item.get(key:"vn");
      Double lat = ((Number)item.get(key:"lat")).doubleValue();
      Double lng = ((Number)item.get(key:"lng")).doubleValue();
      Double elevM = ((Number)item.get(key:"elevM")).doubleValue();
      String obsAbbr = (String) item.get(key:"obsAbbr");

      Map<String, Object> volcano = Map.of(k1:"vnum",vnum,k2:"vn",vn,k3:"lat",lat,k4:"lng",lng,k5:"elevM",elevM, k6:"obsAbbr", obsAbbr);
        volcanodata.add(volcano);
    }

    model.addAttribute(attributeName:"volcanodata", volcanodata);
    return "detailView_volcano";
  }
}
```

volcanoController.java

volcview.wr.usgs.gov/vv-api/volcanoApi/wwvolcanoes

화산 데이터는 위 url의 데이터를 사용했다. 지진 데이터와는 비슷하게 volcano.json 파일을 읽어온 후 ObjectMapper를 사용해 json 데이터를 map 배열로 변환한다. For-each 문을 통해 가져온 데이터 중 필요한 데이터만 추출해 map을 만들고 volcanodata 리스트에 저장한다. 화산 데이터 리스트를 volcanodata 라는 이름으로 모델에 추가한다.


```
[{"vnum":"311800","vn":"Adagdak","lat":51.988,"lng":-176.592,"elevM":610,"color":null,"obsAbbr":"AVO"},{"vnum":"311320","vn":"Akutan","lat":54.134,"lng":-165.986,"elevM":1303,"color":null,
"obsAbbr":"AVO"},{"vnum":"311390","vn":"Amak","lat":55.424,"lng":-163.149,"elevM":488,"color":null,"obsAbbr":"AVO"},{"vnum":"311190","vn":"Amukta","lat":52.5,"lng":-171.252,"elevM":1066,
"color":null,"obsAbbr":"AVO"},{"vnum":"312090","vn":"Aniakchak","lat":56.88,"lng":-158.17,"elevM":1341,"color":null,"obsAbbr":"AVO"},{"vnum":"311160","vn":"Atka","lat":52.332,"lng":-174.137,
"elevM":1451,"color":null,"obsAbbr":"AVO"},{"vnum":"313010","vn":"Augustine","lat":59.363,"lng":-153.43,"elevM":1252,"color":null,"obsAbbr":"AVO"},{"vnum":"315070","vn":"Behm Canal-Rudyerd Bay",
"lat":55.32,"lng":-131.05,"elevM":500,"color":null,"obsAbbr":"AVO"},{"vnum":"312080","vn":"Black Peak","lat":56.552,"lng":-158.785,"elevM":1032,"color":null,"obsAbbr":"AVO"},{"vnum":"311100",
"vn":"Bobrof","lat":51.91,"lng":-177.438,"elevM":738,"color":null,"obsAbbr":"AVO"},{"vnum":"311300","vn":"Bogoslof","lat":53.93,"lng":-168.03,"elevM":150,"color":null,"obsAbbr":"AVO"},
{"vnum":"311010","vn":"Buldir","lat":52.35,"lng":175.911,"elevM":656,"color":null,"obsAbbr":"AVO"},{"vnum":"315001","vn":"Buzzard Creek","lat":64.07,"lng":-148.42,"elevM":830,"color":null,
"obsAbbr":"AVO"},{"vnum":"311230","vn":"Carlisle","lat":52.894,"lng":-170.054,"elevM":1620,"color":null,"obsAbbr":"AVO"},{"vnum":"311200","vn":"Chagulak","lat":52.577,"lng":-171.13,"elevM":1142,
"color":null,"obsAbbr":"AVO"},{"vnum":"312110","vn":"Chiginagak","lat":57.135,"lng":-156.99,"elevM":2221,"color":null,"obsAbbr":"AVO"},{"vnum":"315030","vn":"Churchill","lat":61.38,"lng":-141.75,
"elevM":5005,"color":null,"obsAbbr":"AVO"},{"vnum":"311240","vn":"Cleveland","lat":52.825,"lng":-169.944,"elevM":1730,"color":null,"obsAbbr":"AVO"},{"vnum":"312050","vn":"Dana","lat":55.641,
```


Volcano.json 파일을 간단하게 보면 이렇게 이루어져 있다.

```html
<tbody>
  <tr th:each="vn : ${volcanodata}">
    <td th:text="${vn.vnum}">Unknown</td>
    <td th:text="${vn.vn}">Unknown</td>
    <td th:text="${vn.lat}">Unknown</td>
    <td th:text="${vn.lng}">Unknown</td>
    <td th:text="${vn.elevM}">Unknown</td>
    <td th:text="${vn.obsAbbr}">Unknown</td>
  </tr>
</tbody>
```

detailView_volcano.html

지진 테이블과 같은 방식으로 화산 데이터에 대한 테이블 행을 만들고 thymeleaf를 통해 요소의 값을 가져온다.

4. 평가 및 결과

지난 프로젝트에서는 데이터를 수기로 직접 입력하는 방법을 사용했는데 너무 번거롭고 많은 시간을 요구했다. 그래서 이번 프로젝트는 API를 통해 데이터를 받아오는 방법을 사용했다. API를 통해 데이터를 자동으로 받아오고 필요한 부분만 활용할 수 있었다. 이를 통해 작업을 더욱 빠르고 효율적이게 할 수 있었고, 실시간으로 업데이트되는 데이터를 제공할 수 있게 되었다. 또한, Thymeleaf를 활용해 동적 데이터를 포함하는 웹 페이지를 만드는 방법을 습득할 수 있게 되었다.