

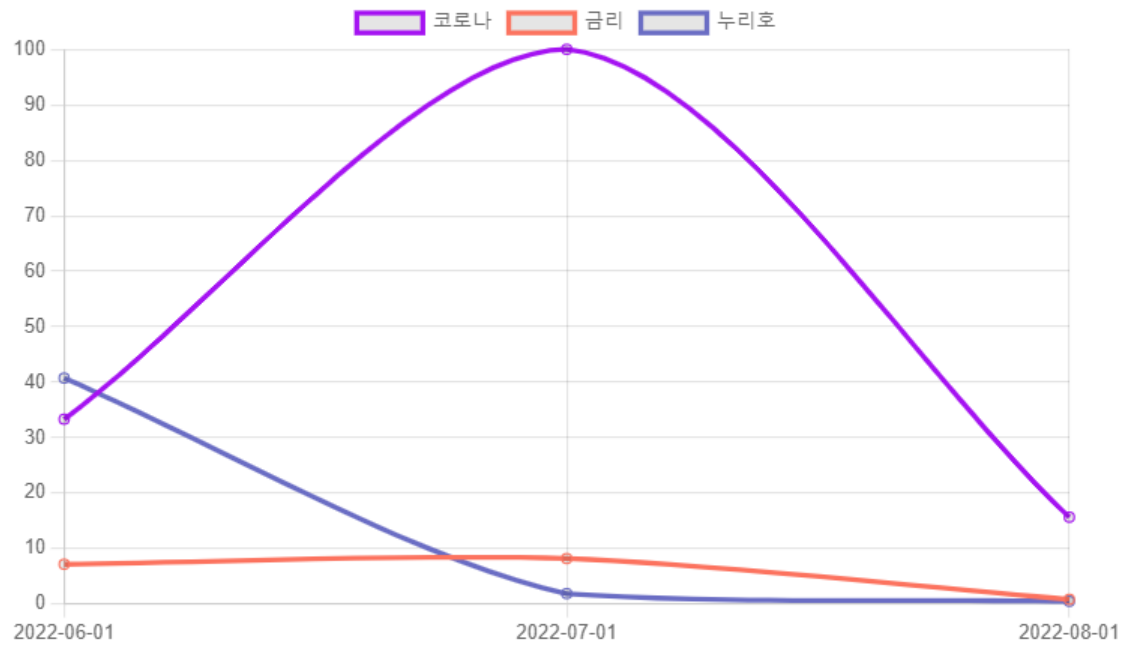


# [project4] Data Visualizer

🕒 Created	@2022년 8월 4일 오후 4:12
🕒 Last Edited Time	@2023년 1월 11일 오후 2:09
📄 Type	
📄 Status	
👤 Created By	 자룡 이
👤 Last Edited By	 온유 이
👥 Stakeholders	
📅 날짜	

## Web PJT 관통 프로젝트

임베디드 뿐만 아니라 수 많은 분야에서 data들은 존재하기 마련이다. 이에 따라 data를 요구사항에 맞게 시각화 (visualization) 하는 연습이 필요하다. 따라서 해당 프로젝트를 통해 네이버 datalab API를 사용해 키워드 검색어 추이를 가져와 차트로 나타내는 Data Visualizing을 진행한다.



시작일

종료일

그룹명

추가

키워드

추가

그룹 확정

제출

## 1. 요구사항 정리

naver datalab api 를 활용해서 data 를 시각화한다.

### A. API 서버 구축하기

- express 서버에서 naver datalab api 를 가져온다.
- api 를 axios 로 가져와서 리턴해주는 프로젝트를 생성한다.

### B. Vue.js 로 visualizing

- axios 로 express 단에 요청을 보낸다.
- 응답 값을 chart 화 시킨다.

## 2. 기술 Stack

### A. express(Node.js)

B. Vue.js

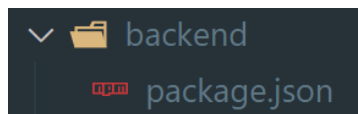
### 3. 개발 환경 구축

웹 개발 에디터인 Visual Studio Code 를 사용한다.

### 4. 기본 세팅하기

#### A. 백엔드 구축

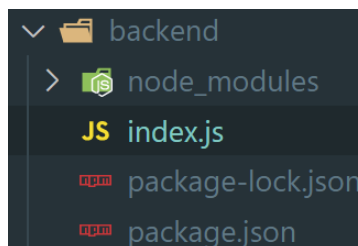
기본 폴더구조를 세팅해준다.



\$ npm init 한 후, package.json 을 생성했다.

```
$ npm i express morgan cors axios dotenv
```

Package 이름	내용
express	Node.js 서버 프레임워크
morgan	클라이언트 http 요청이 들어올 때 로그를 상세히 출력
cors	CORS 정책 허용
axios	서버 코드에서 네이버 API 에 접근해 ajax 비동기통신을 하기 위해 설치
dotenv	환경변수 분리를 위한 라이브러리



index.js 파일을 생성 후, 다음 코드를 기입한다.

```
const express = require("express");

const app = express();
const PORT = 8081;

const cors = require("cors");
app.use(cors());

const morgan = require("morgan");
app.use(morgan("dev"));

app.use(express.json());

app.get("/", async (req, res) => {
```

```

    try {
      return res.json({
        test: true,
      });
    } catch (error) {
      return res.json({
        test: false,
      });
    }
  });
});

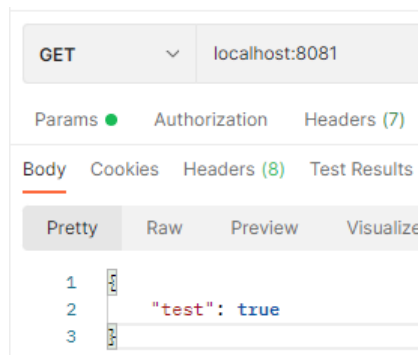
app.listen(PORT, () => console.log(`this server listening on ${PORT}`));

```

현재 8081 번 포트로 설정했는데, `vue-cli` 프로젝트에서 `$ npm run serve` 시, 기본적으로 8080 포트를 사용하므로 포트 충돌을 피하기 위함이다.

서버가 잘 동작하는지 테스트해보자.

```
$ nodemon index
```



POSTMAN 으로 테스트 시, 이상 없이 잘 작동된다.

## B. 프론트엔드 구축

`backend/` 와 별개로, `frontend/` 를 만들 것인데, `$ vue create` 명령어를 사용할 것이다.

```
$ vue create frontend
```

```

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
> Manually select features

```

Manually select features 선택 후 `enter`

```

? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection, and
<enter> to proceed)
>(*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  ( ) Router
  (*) Vuex
  ( ) CSS Pre-processors

```

```
( ) Linter / Formatter
( ) Unit Testing
( ) E2E Testing
```

다음과 같이 선택 후 **enter**

```
? Choose a version of Vue.js that you want to start the project with
  3.x
> 2.x
```

2.x 선택 후 **enter**

```
? Where do you prefer placing config for Babel, ESLint, etc.?
  In dedicated config files
> In package.json
```

In Package.json 선택 후 **enter**

```
? Save this as a preset for future projects? (y/N) N
```

현재 설정 저장할것인지 묻는다. **N** 입력 후 **enter**

이후, 템플릿 프로젝트 생성과정을 거친 후, 다음과 같이 명령어를 입력하자.

```
$ cd frontend
$ npm run serve
```

```
DONE Compiled successfully in 1618ms                                오후 5:11:59

App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.0.94:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

이제 브라우저를 켜고 **http://localhost:8080** 에 접속해보자.



## Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project, check out the [vue-cli documentation](#).

### Installed CLI Plugins

[babel](#) [vuex](#)

### Essential Links

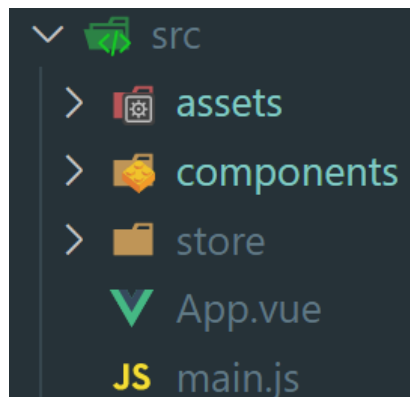
[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

### Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

다음과 같은 Vue.js 웰컴페이지가 나왔다.

우리가 당장 쓸 수 있도록 프로젝트 구조를 바꿔보자.



`assets/` , `components/` 를 삭제하고, `App.vue` 를 열어, 다음과 같이 바꾼다.

```
<template>
  <div id="app">
    <h1>Hello Vue</h1>
```

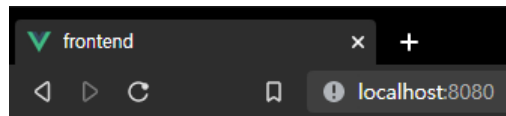
```

</div>
</template>

<script>
export default {
  name: "App",
};
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
</style>

```



# Hello Vue

다음과 같이 구성한 후, 프로젝트를 위해 필요한 패키지를 생성하자.

```
$ npm i axios vue-chartjs chart.js
```

Package 이름	내용
axios	비동기 요청 라이브러리
chart.js	JavaScript Chart 라이브러리
vue-chartjs	chart.js 를 Vue.js 에서 사용하기 위한 라이브러리

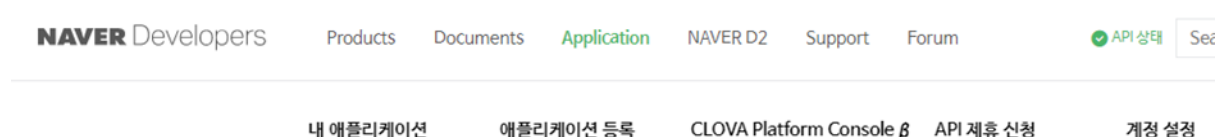
여기까지 진행하면 `backend/` 와 `frontend/` 의 기초 세팅은 마무리된다.

## C. API 사용하기

네이버의 datalab API 를 활용한다.

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

접속해서 로그인 후,



애플리케이션 등록을 진행해준다.

애플리케이션 이름 ↗	<input type="text" value="data-visualization"/> ✓ <ul style="list-style-type: none"> <li>네이버 아이디로 로그인할 때 사용자에게 표시되는 이름이므로 가급적 10자 이내의 간결한 이름을 사용해주세요.</li> <li>40자 이내의 영문, 한글, 숫자, 공백문자, "-", "_" 만 입력 가능합니다.</li> </ul>				
사용 API ↗	<div>선택하세요. ▼ ✓</div> <table border="1"> <tr> <td>데이터랩 (검색어트렌드)</td> <td>×</td> </tr> <tr> <td>데이터랩 (쇼핑인사이트)</td> <td>×</td> </tr> </table>	데이터랩 (검색어트렌드)	×	데이터랩 (쇼핑인사이트)	×
데이터랩 (검색어트렌드)	×				
데이터랩 (쇼핑인사이트)	×				
비로그인 오픈 API 서비스 환경	<div>환경 추가 ▼</div> <div> <div>WEB 설정 × ^</div> <div> <p>웹 서비스 URL (최대 10개)</p> <input type="text" value="http://127.0.0.1"/> + ✓  <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 URL)</li> </ul> </div> </div>				

등록하기

취소

사용 API - 데이터랩 (검색어트렌드), 데이터랩 (쇼핑인사이트)

환경 - WEB

URL - `http://127.0.0.1`

## 5. Backend 구축하기

`backend/index.js` 에 다음을 추가한다.

```
const axios = require("axios");
const dotenv = require("dotenv");
const fs = require("fs");
dotenv.config();
```

`fs` 를 사용했다. 즉, 파일을 작성할 계획이다.

`dotenv.config()` 는 `.env` 파일에 접근하기 위한 세팅이다. 아이디, 비밀번호, API 키 등 중요 정보를 `index.js` 파일이 아니라, `.env` 파일에서 따로 관리한다.



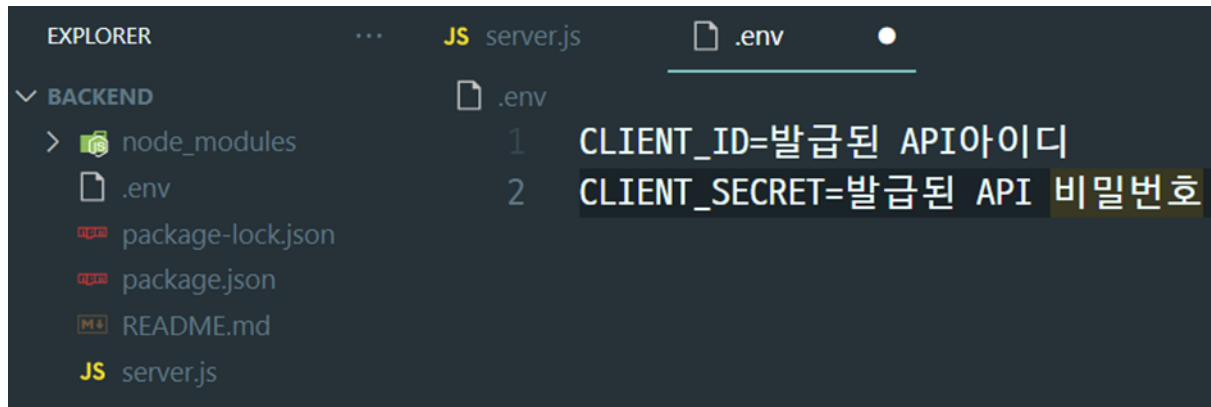
`.env` 파일을 생성해서 발급된 API ID와 비밀번호를 기입한다.

data-visualization

개요	API 설정	멤버관리	로그인 통계
----	--------	------	--------

애플리케이션 정보

Client ID	xhmCKelaqusMvjXN7Tge
Client Secret	..... <button>보기</button>



추가로, api 요청 시 에러를 방지하기 위해 `backend/` 에 `uploads/` 라는 디렉터리를 미리 만들어두자.

작성할 REST API 는 다음과 같다.

POST /data	네이버 검색 API 에, 시작일과 종료일, 검색어 등을 포함한 JSON 을 비동기통신으로 보낸 후, 네이버에서 보낸 요청에 대한 결과를 <code>uploads/chart.js</code> 파일로 저장
GET /data	<code>uploads/chart.json</code> 에 저장된 데이터를 JSON 포맷으로 출력
DELETE /data	<code>uploads/chart.js</code> 파일 삭제

이제 data 를 받아오는 `app.post` 를 작성해본다.

```
app.post("/data", async (req, res) => {
  try {
    const request_body = {
      startDate: "2022-06-01",
      endDate: "2022-08-01",
      // date, week, month 세 가지 제공
      timeUnit: "month",
      keywordGroups: [
        {
          groupName: "코로나",
          keywords: ["코로나", "covid", "백신", "거리두기"],
        },
      ],
    },
```

```

    {
      groupName: "금리",
      keywords: ["금리", "빅스텝", "파월"],
    },
    {
      groupName: "누리호",
      keywords: ["누리호", "항우연"],
    },
  ],
};
const url = "https://openapi.naver.com/v1/datalab/search";
const headers = {
  "Content-Type": "application/json",
  "X-Naver-Client-Id": process.env.CLIENT_ID,
  "X-Naver-Client-Secret": process.env.CLIENT_SECRET,
};
const response = await axios.post(url, request_body, {
  headers: headers,
});
fs.writeFile(
  `./uploads/chart.json`,
  JSON.stringify(response.data.results),
  (error) => {
    if (error) {
      throw error;
    }
  }
);
return res.json(response.data.results);
} catch (error) {
  return res.json(error);
}
});

```

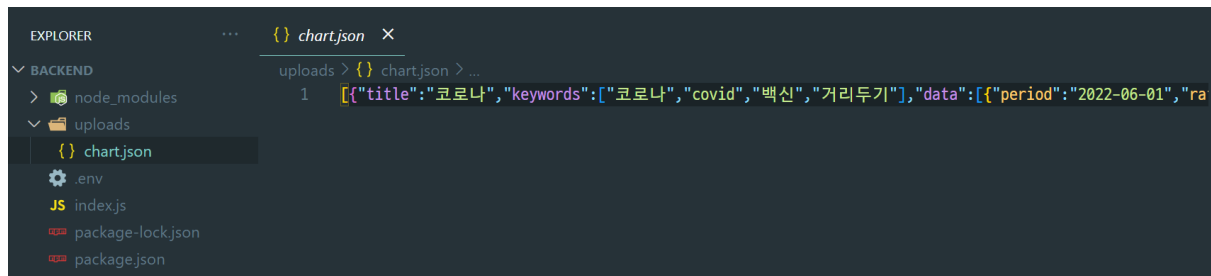
우선, Postman 으로 테스트해보면,

```

1  [
2    {
3      "title": "코로나",
4      "keywords": [
5        "코로나",
6        "covid",
7        "백신",
8        "거리두기"
9      ],
10     "data": [
11       {
12         "period": "2022-06-01",
13         "ratio": 33.32972
14       },
15       {
16         "period": "2022-07-01",
17         "ratio": 100
18       },
19       {
20         "period": "2022-08-01",
21         "ratio": 10.34851
22       }
23     ]
24   },
25   {
26     "title": "금리",
27     "keywords": [
28       "금리",
29       "빅스텝",
30       "파월"
31     ],

```

다음과 같은 결과가 잘 찍히고, 이 결과는



`uploads/chart.json` 에 파일로 잘 저장된것을 확인할 수 있다.

이제 각 코드가 무엇을 의미하는지 분석해보자.

```
app.post("/data", async (req, res) => {
  try {
    const request_body = {
      startDate: "2022-06-01",
      endDate: "2022-08-01",
      // date, week, month 세 가지 제공
      timeUnit: "month",
      keywordGroups: [
        {
          groupName: "코로나",
          keywords: ["코로나", "covid", "백신", "거리두기"],
        },
        {
          groupName: "금리",
          keywords: ["금리", "빅스텝", "파월"],
        },
        {
          groupName: "누리호",
          keywords: ["누리호", "항우연"],
        },
      ],
    };
  }
});
```

서버로 딸려보낼 `JSON` 을 `request_body` 라고 이름지었다.

우리가 원하는 건, 검색결과이다. 2022년 6월 1일부터, 2022년 8월 1일 세 달간,

`keywordGroups` 에 해당하는 검색어로 검색한 결과를 보고자 한다.

`groupName` 은 총 3개로서, 각각 검색 키워드들의 결과를 묶을 것이다.

즉, 3개월간 각 이슈들의 흐름을 분석하고자 한다.

```
const url = "https://openapi.naver.com/v1/datalab/search";
const headers = {
  "Content-Type": "application/json",
  "X-Naver-Client-Id": process.env.CLIENT_ID,
  "X-Naver-Client-Secret": process.env.CLIENT_SECRET,
};
const response = await axios.post(url, request_body, {
  headers: headers,
});
```

보낼 `url` 과, `header` 라는 걸 같이 딸려보낼 것인데,

`header` 는 서버 코드에 들어갈 설정이라고 보면 된다.

api 사용할 때 어떤 식으로 적으라고 써있기 때문에 그것을 기반으로 적은 것이지만,

우리는 `JSON` 타입으로 받길 원하기 때문에 `Content-Type` 이 `JSON` 이고, 아이디와 비밀번호를 직접 입력하지 않고 `.env` 파일에서 가져와 쓴다.

`.env` 파일의 존재이유는, 코드에 중요 정보를 나타내지 않음으로서 해커가 공격할 수 있는 빌미를 사전 차단하기 위함이다.

그리고 `axios.post` 통신을 진행하는데, `url`, `request_body`, `header` 를 모두 딸려보낸다.

```
fs.writeFile(
  './uploads/chart.json',
  JSON.stringify(response.data.results),
  (error) => {
    if (error) {
      throw error;
    }
  }
);
return res.json(response.data.results);
```

그리고, 요청이 성공했을 때 차트를 그리기위한 배열이 주어질 것이다.

따라서, 우리는 서버에서 받아온 데이터로 `chart.json` 이라는 파일을 생성할 것인데, `JSON.stringify` 는 받아온 데이터를 `JSON` 양식으로 바꿀 때 쓴다.

여기서 주의 할 점은 `chart.json` 파일은 단순 데이터를 저장하고 보여주는 서버에 대한 부하를 덜어주기위한 캐싱의 목적으로만 사용하고 있다. 가져온 데이터의 내용을 변경하거나 추가하면 안되는 것에 유의한다.

지금까지 작성한 `HTTP` 요청은 `POST` 이다. 목적은 무엇인가?

`POST /data`

네이버 검색 API 에, 시작일과 종료일, 검색어 등을 포함한 `JSON` 을 비동기통신으로 보낸 후, 네이버에서 보낸 요청에 대한 결과를 `uploads/chart.js` 파일로 저장

즉, 내 백엔드 서버에 데이터 기록이다. 그래서 `POST` 인 것이다. 이제, `GET` 을 통해 파일을 읽어오는 로직을 작성해보자.

```
app.get("/data", async (req, res) => {
  try {
    res.set("Content-Type", "application/json; charset=utf-8");
    const tempFile = fs.createReadStream("uploads/chart.json");
    return tempFile.pipe(res);
  } catch (error) {
    return res.json(error);
  }
});
```

결론적으로 `POST` 요청시에 만들어진 `chart.json` 을 단순 리턴해주는 로직이다.

`res.set` 은 `res` 객체 설정인데, `JSON` 타입이고, 한글을 위해 `utf-8` 로 설정했다.

파일을 해당 경로에서 읽어오고, 담은 다음, `pipe()` 를 써주었다.

`pipe()` 행에 대해 좀 더 설명을 덧붙이자면, `tempFile` 은 readable stream 이며,

`res` 는 목적지이다. 즉, 응답, `res` 객체이다

`pipe` 는 둘을 연결할 때 쓴다.

그 실행결과를 리턴하게 되고, `chart.json` 이 `JSON` 포맷으로 보여지게 된다.

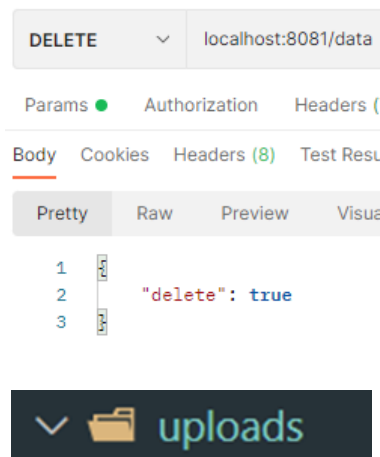
```
[
  {
    "title": "코로나",
    "keywords": [
      "코로나",
      "covid",
      "백신",
      "거리두기"
    ],
    "data": [
      {
        "period": "2022-06-01",
        "ratio": 33.32972
      },
      {
        "period": "2022-07-01",
        "ratio": 100
      },
      {
        "period": "2022-08-01",
        "ratio": 15.66919
      }
    ]
  },
  {
    "title": "금리",
    "keywords": [
      "금리",
      "빅스텝",
      "파월"
    ],
    "data": [
      {
        "period": "2022-06-01",
        "ratio": 7.17983
      },
      {
        "period": "2022-07-01",
        "ratio": 8.21018
      },
      {
        "period": "2022-08-01",
        "ratio": 0.85917
      }
    ]
  },
  {
    "title": "누리호",
    "keywords": [
      "누리호",
      "항우연"
    ],
    "data": [
      {
        "period": "2022-06-01",
        "ratio": 40.73772
      },
      {
        "period": "2022-07-01",
        "ratio": 1.89189
      },
      {
        "period": "2022-08-01",
        "ratio": 0.5017
      }
    ]
  }
]
```

단순히 `string` 일 경우, 컬러 하이라이팅이나 인덴팅이 적용되지 않아 가져오기 실패한 경우라고 보면 된다. 지금의 경우, `JSON` 포맷에 알맞게 잘 가져왔기 때문에 결과도 잘 나왔다.

이제 **DELETE** 요청 또한 만들어준다.

```
app.delete("/data", (req, res) => {
  try {
    fs.unlink("uploads/chart.json", (error) => {
      if (error) {
        return res.json(error);
      }
    });
    return res.json({
      delete: true,
    });
  } catch (error) {
    return res.json(error);
  }
});
```

`fs.unlink()` 는 해당 파일 삭제를 뜻한다. 삭제 성공시 확인 가능한 간단한 **JSON** 리턴한다.



파일이 사라진 것을 확인할 수 있다.

이제 요청에 따라 데이터를 맞게 요청 할 수 있게 `app.post('/data')` 를 수정해준다.

이미 `app.post('/data')` 가 만들어져 있지만, 클라이언트가 어떤 데이터를 입력한 게 아니라 단순히 하드코딩한 데이터로 네이브 API 를 사용했을 뿐이다.

지금부터, 테스트를 위한 하드코딩 부분을 빼버리고, 클라이언트에서 입력을 받아 결과를 확인해보도록 한다.

## 파라미터

파라미터를 JSON 형식으로 전달합니다.

파라미터	타입	필수 여부	설명
startDate	string	Y	조회기간 시작 날짜(yyyy-mm-dd 형식). 2017년 8월 1일부터 조회할 수 있습니다.
endDate	string	Y	조회기간 종료 날짜(yyyy-mm-dd 형식)
timeUnit	string	Y	구간 단위 - date: 일간 - week: 주간 - month: 월간
category	array(JSON)	Y	분야 이름과 분야 코드 쌍의 배열. 최대 3개의 쌍을 배열로 설정할 수 있습니다.
category.name	string	Y	쇼핑 분야 이름
category.param	array(string)	Y	쇼핑 분야 코드. <a href="#">네이버쇼핑</a> 에서 카테고리를 선택했을 때의 URL에 있는 cat_id 파라미터의 값으로 분야 코드를 확인할 수 있습니다.
device	string	N	기기. 검색 환경에 따른 조건입니다. - 설정 안 함: 모든 기기에서의 검색 클릭 추이 - pc: PC에서의 검색 클릭 추이 - mo: 모바일 기기에서의 검색 클릭 추이
gender	string	N	성별. 검색 사용자의 성별에 따른 조건입니다. - 설정 안 함: 모든 성별 - m: 남성 - f: 여성

이것은 우리가 사용할 API 공식문서의 일부분이다. 전체 공식문서는

[https:// developers.naver.com/docs/serviceapi/datalab/search/search.md](https://developers.naver.com/docs/serviceapi/datalab/search/search.md) 에서 확인 가능하다.

이를 기반으로 `POST /data` 를 수정해보면 다음과 같다.

```
app.post("/data", async (req, res) => {
  const { startDate, endDate, timeUnit, device, gender, keywordGroups } =
    req.body;
  try {
    const request_body = {
      startDate: startDate,
      endDate: endDate,
      timeUnit: timeUnit,
      device: device === "all" ? "" : device,
      gender: gender === "all" ? "" : gender,
      keywordGroups: keywordGroups,
    };
  };
```

이젠 하드코딩이 아니라, 클라이언트에서 준 데이터를 기반으로 통신할 것이므로 `req.body` 에서 떨어진 것들을 변수로 받아서, `request_body` 를 재작성한다.  
삼항연산자가 나오긴 했는데, 만약 `device` 가 `all` 이면 `device` 는 빈 `string` 이고, 특정 값이 있으면 해당 값을 팔려보낸다. `gender` 도 마찬가지다.

```
{
  "startDate": "2022-06-01",
  "endDate": "2022-08-01",
  "timeUnit": "month",
  "keywordGroups": [
    {
      "groupName": "코로나",
      "keywords": ["코로나", "covid", "백신", "거리두기"]
    },
    {
      "groupName": "금리",
      "keywords": ["금리", "빅스텝", "파월"]
    },
    {
      "groupName": "누리호",
      "keywords": ["누리호", "항우연"]
    }
  ]
}
```

2022년 6월 1일부터 8월 1일까지 주요 이슈 세 가지는 코로나, 금리, 누리호였다.

다음과 같은 `JSON` 데이터를 POSTMAN 으로, `POST /data` 로 보냈을 때,

```
1  [
2    {
3      "title": "코로나",
4      "keywords": [
5        "코로나",
6        "covid",
7        "백신",
8        "거리두기"
9      ],
10     "data": [
11       {
12         "period": "2022-06-01",
13         "ratio": 33.32972
14       },
15       {
16         "period": "2022-07-01",
17         "ratio": 100
18       },
19       {
20         "period": "2022-08-01",
21         "ratio": 10.34851
22       }
23     ]
24   },
25   {
26     "title": "금리",
27     "keywords": [
28       "금리",
29       "빅스텝",
30       "파월"
31     ],
```

통신 성공 후 API 결과값이 리턴되었고,

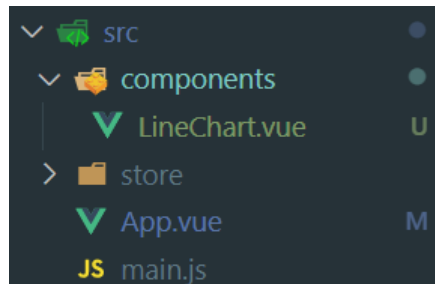
```
{ } chart.json X
uploads > { } chart.json > { } 2 > [ ] data > { } 1 > [ ] period
1 [{"title":"코로나","keywords":["코로나","covid","백신","거리두기"],"data":[{"period":"2022-06-01","ratio":33.32972},{
```



`chart.json` 에서 차트 데이터도 잘 받아왔다.

백엔드 구축은 이로서 마무리된다.

## 6. Frontend 구축하기



`src/components/LineChart.vue` 를 생성하고, 다음과 같이 기입한다. 아래 코드는 `vue-chartjs` 패키지의 Vue.js 2 버전 Line Chart 의 샘플코드를, 우리가 쓸 목적에 맞게 하드코딩한 코드다.

```
<template>
  <LineChartGenerator
    :chart-options="chartOptions"
    :chart-data="chartData"
    :chart-id="chartId"
    :dataset-id-key="datasetIdKey"
    :width="width"
    :height="height"
  />
</template>

<script>
import { Line as LineChartGenerator } from "vue-chartjs/legacy";

import {
  Chart as ChartJS,
  Title,
  Tooltip,
  Legend,
  LineElement,
  LinearScale,
  CategoryScale,
  PointElement,
} from "chart.js";

ChartJS.register(
  Title,
  Tooltip,
  Legend,
  LineElement,
  LinearScale,
  CategoryScale,
  PointElement
);

export default {
  name: "LineChart",
  components: {
    LineChartGenerator,
  },
  data() {
    return {
      chartId: "line-chart",
      datasetIdKey: "label",
      width: 400,
```

```

    height: 400,
    chartData: {
      labels: ["2022-06-01", "2022-07-01", "2022-08-01"],
      datasets: [
        {
          label: "금리",
          backgroundColor: "blue",
          data: [7.17983, 8.21018, 0.56848],
        },
        {
          label: "코로나",
          backgroundColor: "red",
          data: [33.32972, 100, 10.34851],
        },
        {
          label: "누리호",
          backgroundColor: "green",
          data: [40.73772, 1.89189, 0.13467],
        },
      ],
    },
    chartOptions: {
      responsive: true,
      maintainAspectRatio: false,
    },
  };
};
</script>

```

그리고, `App.vue` 에서 `LineChart.vue` 를 자식 컴포넌트로 사용한다.

```

<template>
  <div id="app">
    <h1>Hello Vue</h1>
    <LineChart />
  </div>
</template>

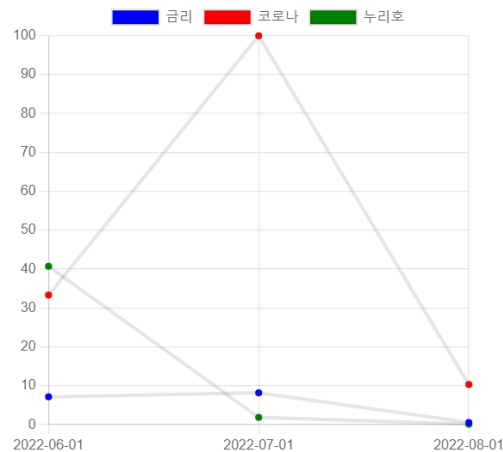
<script>
import LineChart from "../components/LineChart";
export default {
  name: "App",
  components: { LineChart },
};
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
</style>

```

다음과 같은 화면을 볼 수 있다.

## Hello Vue



이 결과를 기반으로, `LineChart.vue` 에서 필요한 부분만 분석해보자.

```
<template>
  <LineChartGenerator
    :chart-options="chartOptions"
    :chart-data="chartData"
    :chart-id="chartId"
    :dataset-id-key="datasetIdKey"
    :width="width"
    :height="height"
  />
</template>
```

`LineChartGenerator` 는 `vue-chartjs` 에서 Line Chart, 즉 꺾은선그래프를 사용하기 위한 컴포넌트다. 여기에 각종 `props` 를 내려보내 데이터를 등록하는것을 알 수 있다.

`<script>` 의 내용을 살펴보자.

```
import { Line as LineChartGenerator } from "vue-chartjs/legacy";

import {
  Chart as ChartJS,
  Title,
  Tooltip,
  Legend,
  LineElement,
  LinearScale,
  CategoryScale,
  PointElement,
} from "chart.js";

ChartJS.register(
  Title,
  Tooltip,
  Legend,
  LineElement,
  LinearScale,
  CategoryScale,
  PointElement
);
```

Vue.js 2 버전에서 사용하기 위해선, `vue-chartjs/legacy` 에서 가져와야한다.

그리고 `chart.js` 패키지에서 필요한 객체들을 가져온다. 차트 객체, 타이틀, 툴팁 등.

이것들을 가져와, `Chart` 객체의 별칭으로 지정한 `ChartJS` 객체에 등록한다.

```
export default {
  name: "LineChart",
  components: {
    LineChartGenerator,
  },
  data() {
    return {
      chartId: "line-chart",
      datasetIdKey: "label",
      width: 400,
      height: 400,
      chartData: {
        labels: ["2022-06-01", "2022-07-01", "2022-08-01"],
        datasets: [
          {
            label: "금리",
            backgroundColor: "blue",
            data: [7.17983, 8.21018, 0.56848],
          },
          {
            label: "코로나",
            backgroundColor: "red",
            data: [33.32972, 100, 10.34851],
          },
          {
            label: "누리호",
            backgroundColor: "green",
            data: [40.73772, 1.89189, 0.13467],
          },
        ],
      },
      chartOptions: {
        responsive: true,
        maintainAspectRatio: false,
      },
    };
  },
};
```

`props` 로 내려보낼 데이터들이다. `width` 와 `height` , 그리고 `chartData` 를 내려보내는데, `chartData` 내부엔 `labels` 는 배열이다. 표의 x축의 기준이 될 것이다.

`dataSets` 안에는 크기가 3인 배열이 들어갔는데, 각각의 라벨, 구분할 색깔, 들어갈 데이터가 있고, 제일 중요한 `data` 는 배열로 이루어져있다.

`chartOptions` 는 현재 차트의 옵션값 지정이다.

`chart.js` 공식문서를 참고해, 여러가지 설정을 바꿀 수 있다. 곡선을 부드럽게하려면 `datasets` 배열의 각각의 객체에 `tension` 을 추가할 수 있다.

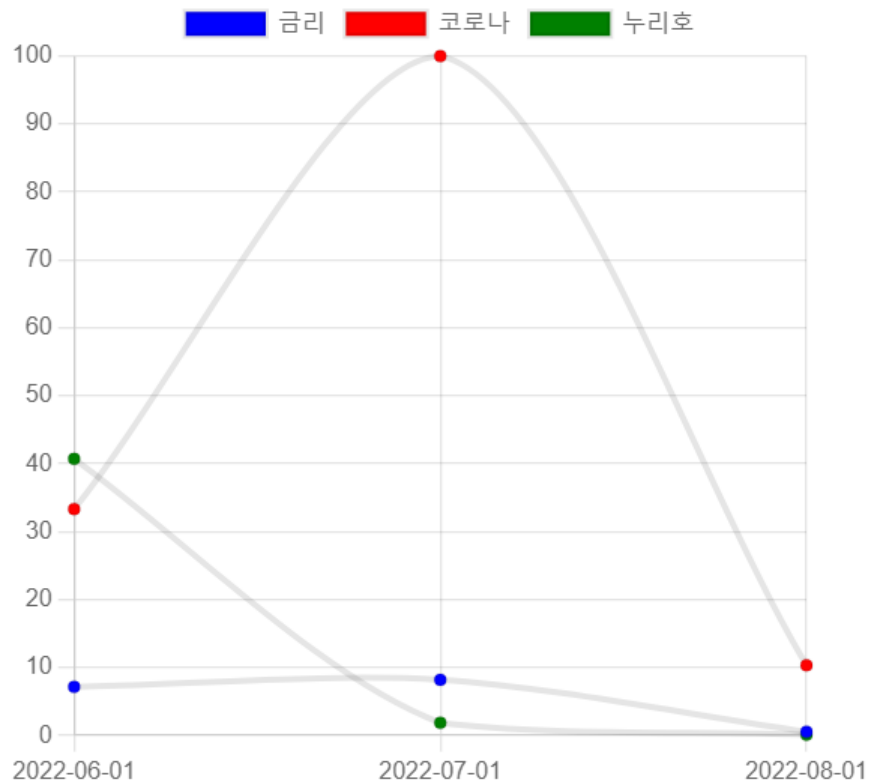
```
datasets: [
  {
    label: "금리",
    backgroundColor: "blue",
    data: [7.17983, 8.21018, 0.56848],
    tension: 0.3,
  },
  {
    label: "코로나",
    backgroundColor: "red",
    data: [33.32972, 100, 10.34851],
  },
]
```

```

    tension: 0.3,
  },
  {
    label: "누리호",
    backgroundColor: "green",
    data: [40.73772, 1.89189, 0.13467],
    tension: 0.3,
  },
];

```

다음과 같이, 각각 `tension: 0.3` 으로 지정했을 때, 그래프는 다음과 같이 좀 더 부드럽게 바뀐다.



이제 무엇을 하면 되는가? 하드코딩으로 다음과 같은 그래프를 구현할 수 있다는 뜻은, 프론트엔드에서 백엔드에 적절한 요청을 보냈을 때, 그래프의 데이터를 적절히 갱신하여 원하는 그래프로 바꿀 수 있음을 뜻한다.

이에, 다음의 작업을 필요로 한다.

1. 프론트엔드 `axios` API
2. 입력 창
3. 적절한 데이터 파싱

우선, `axios` API 제작을 위해 `utils/` 디렉터리를 만들고, `axios.js` 파일을 생성해 다음 코드를 기입한다.

```

const axios = require("axios");

const api = axios.create({
  baseURL: "http://localhost:8081",

```

```
});

export const dataLab = {
  get: () => {
    return api.get("/data");
  },
  post: (data) => {
    return api.post("/data", data);
  },
  delete: () => {
    return api.delete("/data");
  },
};
```

다음, vuex 작업을 해보자. `store/index.js` 파일을 다음과 같이 수정한다.

```
import Vue from "vue";
import Vuex from "vuex";

Vue.use(Vuex);

export default new Vuex.Store({
  state: { chartData: {} },
  mutations: {
    CHANGE_CHART_DATA(state, data) {
      state.chartData = data;
    },
  },
});
```

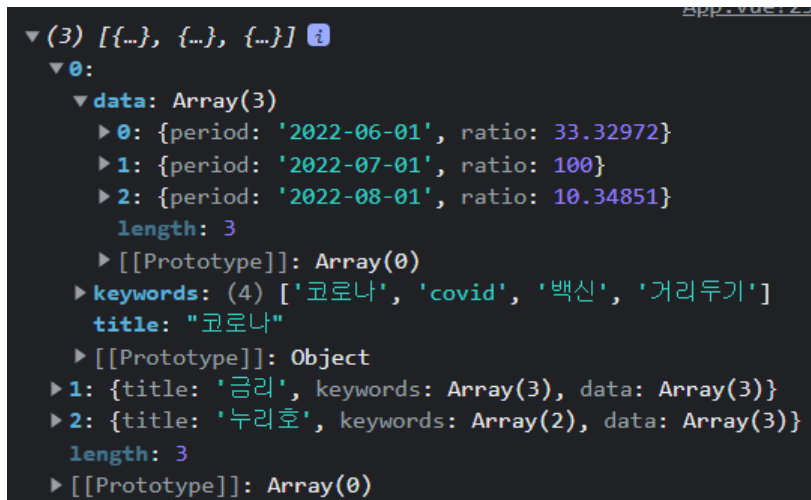
당장 `chartData` 는 빈 객체 `{}` 이지만, `GET /data` 명령 이후 채워질 것이며, 새로운 차트를 필요로 할 땐 `CHANGE_CHART_DATA` 를 사용해 새로운 차트 데이터를 요청받아 차트를 다시 그릴 것이다.

`App.vue` 에서, `axios` API 와 `mapState`, `mapMutations` 를 사용할 수 있도록 준비하자.

```
import { dataLab } from "../utils/axios";
import { mapState, mapMutations } from "vuex";
import LineChart from "../components/LineChart";
export default {
  name: "App",
  components: { LineChart },
  computed: {
    ...mapState(["chartData"]),
  },
  methods: {
    ...mapMutations(["CHANGE_CHART_DATA"]),
  },
};
```

그리고, `created()` 를 다음과 같이 작성한다.

```
async created() {
  const response = await dataLab.get();
  console.log(response.data);
},
```



데이터를 분석해보니, 총 세 개의 결과가 나왔는데 각각 코로나, 금리, 누리호 묶음이다.

그리고, 각각은 `data` 를 가지는데, 총 3개의 배열이다. 왜냐면, 6월 1일부터 8월 1일까지 한 달 간격이면

6월 1일      7월 1일      8월 1일

총 세 번 찍히기 때문이다.

여기서, `ratio` 는 상대적인 기준이다. 각각의 데이터들을 비교해 해당 날짜의 검색어 빈도를 나타낸다. 반드시, 전체 데이터 중 기준이 되는 `ratio` 100 과 0은 존재한다.

우리 데이터의 경우, 7월 1일에 코로나, covid, 백신, 거리두기 네 가지 검색어, 즉 “코로나” 검색어묶음이 전체 `ratio` 중 가장 빈도값이 크기 때문에 100을 가진다.

확인한 데이터 형태를 기반으로, `created()` 에서 직접 파싱해보자.

```
async created() {
  const response = await dataLab.get();
  const chartData = {
    labels: response.data[0].data.map((li) => li.period),
    datasets: response.data.reduce((acc, cur) => {
      const label = cur.title;
      const data = cur.data.map((li) => li.ratio);
      acc.push({
        label: label,
        data: data,
        backgroundColor: "red",
        tension: 0.3,
      });
      return acc;
    }, []),
  };
  this.CHANGE_CHART_DATA(chartData);
  console.log(this.chartData.labels);
  console.log(this.chartData.datasets);
},
```

먼저, `labels` 의 경우, 날짜를 의미한다. 배열이 몇 개가 되든 날짜의 기준은 동일하기에, 0번 인덱스를 반복하면서 날짜만 모은 것이다.

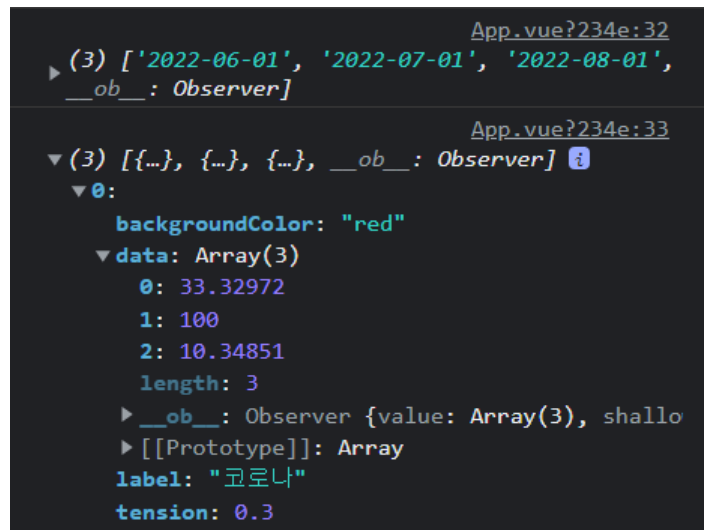
`datasets` 에 `ratio` 를 가져와야한다. 오랜만에 `reduce()` 사용해보자.

빈 배열 `[]` 이 초깃값이고, 지금의 경우 한글 영어 총 두 개이기 때문에 두 번 반복할 것이다.

`label` 은 `title`, `data` 는 `ratio` 가 될 것이고,

`acc` 에 `push` 하는 식으로 반복한다.

그리고 맨 마지막으로, `vuex` 의 전역 상태 `chartData` 를 바꿔주자.



현재 콘솔에 찍힌 데이터로 판단해보았을 때, 파싱은 잘 되었다. 이제 그래프를 갱신해야한다.

`LineChart.vue` 의 `chartData` 는 우리가 다음과 같이 하드코딩해두었다.

```

chartData: {
  labels: ["2022-06-01", "2022-07-01", "2022-08-01"],
  datasets: [
    {
      label: "금리",
      backgroundColor: "blue",
      data: [7.17983, 8.21018, 0.56848],
      tension: 0.3,
    },
    {
      label: "코로나",
      backgroundColor: "red",
      data: [33.32972, 100, 10.34851],
      tension: 0.3,
    },
    {
      label: "누리호",
      backgroundColor: "green",
      data: [40.73772, 1.89189, 0.13467],
      tension: 0.3,
    },
  ],
},
},

```

이것을 `vuex` 의 데이터로 바꾸면 될 것이다. `LineChart.vue` 를 다음과 같이 수정하자.

```
import { mapState } from "vuex";
```

```

export default {
  name: "LineChart",
  components: {
    LineChartGenerator,
  },
  computed: {
    ...mapState(["chartData"]),
  },
  data() {
    return {

```

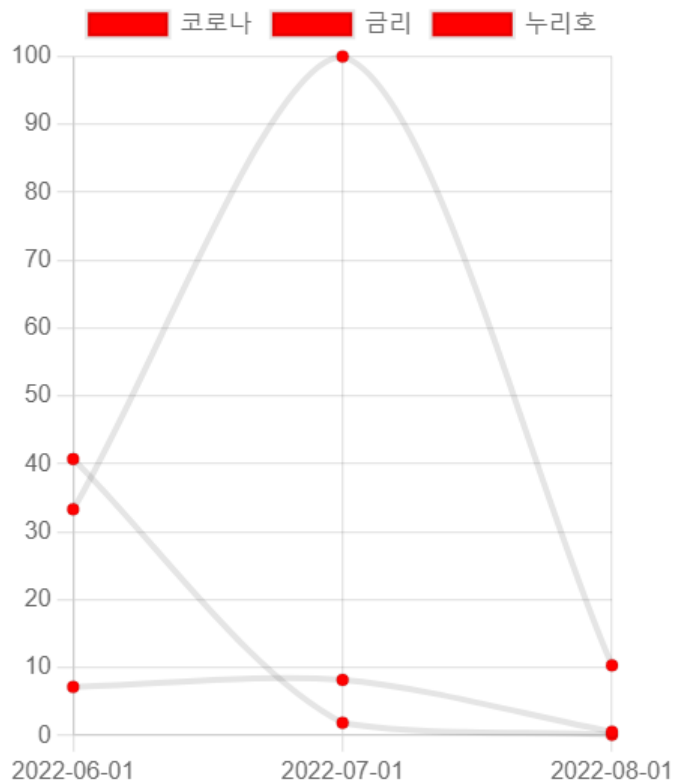


```

    chartId: "line-chart",
    datasetIdKey: "label",
    width: 400,
    height: 400,
    // chartData: {...},
    chartOptions: {
      responsive: true,
      maintainAspectRatio: false,
    },
  },
},
},
};

```

기존의 하드코딩되어있던 `chartData` 를 삭제해버리고, `vuex` 에서 가져온 `chartData` 로 대체했다.



이 그래프는 무엇을 의미하는가? 방금 전과 동일해보이지만, 우리가 하드코딩한 데이터로 그린 그래프가 아니라, 서버의 `chart.json` 을 받아와서, 해당 데이터를 클라이언트에서 파싱한 후, 파싱한 데이터를 받아와 그래프를 그린 것이다. 즉, 성공이다.

이제 테스트를 위해 `App.vue` 에서 사용된 `console.log` 와, `mapState` 는 필요치 않으므로, `App.vue` 전체를 다음과 같이 수정해준다.

```

<template>
  <div id="app">
    <LineChart />
  </div>
</template>

<script>
import { dataLab } from "../utils/axios";
import { mapMutations } from "vuex";
import LineChart from "../components/LineChart";
export default {

```

```

name: "App",
components: { LineChart },
async created() {
  const response = await dataLab.get();
  const chartData = {
    labels: response.data[0].data.map((li) => li.period),
    datasets: response.data.reduce((acc, cur) => {
      const label = cur.title;
      const data = cur.data.map((li) => li.ratio);
      acc.push({
        label: label,
        data: data,
        backgroundColor: "red",
        tension: 0.3,
      });
      return acc;
    }, []),
  };
  this.CHANGE_CHART_DATA(chartData);
},
methods: {
  ...mapMutations(["CHANGE_CHART_DATA"]),
},
};
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
</style>

```

선 색깔을 지정하고, 색깔을 랜덤으로 다르게 만들어보자.

App.vue

```

acc.push({
  label: label,
  data: data,
  borderColor: this.makeColor(),
  tension: 0.3,
});

```

`backgroundColor` 를 빼버리고, `borderColor` 라는 걸 쓰겠다. `chart.js` 패키지는 선 색깔도 지정할 수 있는데, 이는 `borderColor` 를 사용하며, `this.makeColor()` 로 지정한다.

`makeColor()` 메서드는 다음과 같이 작성하는데,

```

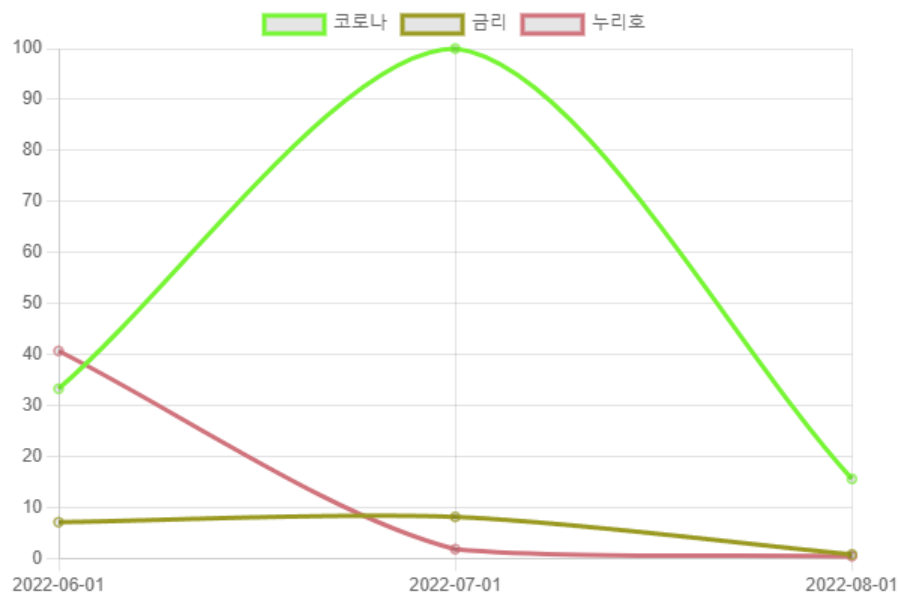
methods: {
  ...mapMutations(["CHANGE_CHART_DATA"]),
  makeColor() {
    return "#" + Math.round(Math.random() * 0xffffff).toString(16);
  },
},

```

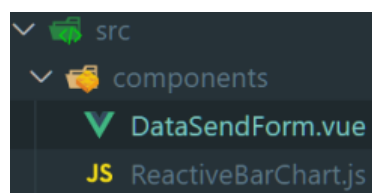
이 코드는 외우고 다니는 게 아니라, js 랜덤 색 만들기 등으로 검색했을 때 나온 예제다.

<https://tranquilotter.tistory.com/5>

16진수와 관련있던데, 설명은 생략.



사용자 입력을 위한 자식컴포넌트 하나 만들어주겠다.



`DataSendForm.vue` 만들고,

```
<template>
  <div>DataSendForm</div>
</template>

<script>
export default {
  name: "DataSendForm",
};
</script>

<style scoped></style>
```

다음과 같이 기본 템플릿을 만들고,

`App` 에 임포트하고, 등록하고, 붙인다.


여기서 할 일은, 다음과 같은 `JSON` 을 만드는 것이다.


```

{
  "startDate": "2022-06-01",
  "endDate": "2022-08-01",
  "timeUnit": "month",
  "keywordGroups": [
    {
      "groupName": "코로나",
      "keywords": ["코로나", "covid", "백신", "거리두기"]
    },
    {
      "groupName": "금리",
      "keywords": ["금리", "빅스텝", "파월"]
    },
    {
      "groupName": "누리호",
      "keywords": ["누리호", "항우연"]
    }
  ]
}

```

그래서 우선 다음과 같은 화면을 만들 것이다.

시작일  

종료일  

월간

그룹명:

키워드:

사용자 입력 그룹별 키워드

그룹 이름: 코로나

그룹 키워드: ["코로나", "covid", "백신", "거리두기"]

그룹 이름: 금리

그룹 키워드: ["금리", "빅스텝"]

그룹 이름: 누리호

그룹 키워드: ["누리호", "항우연"]

안 예쁘지만, 적어도 동작은 할 것이다.

콘솔로그로 다음과 같이 찍히는 게 목표다.

```

▼ {__ob__: Observer} ⓘ
  endDate: "2022-08-01"
  ▼ keywordGroups: Array(3)
    ▼ 0:
      groupName: "코로나"
      ▼ keywords: Array(4)
        0: "코로나"
        1: "covid"
        2: "백신"
        3: "거리두기"
        length: 4
        ▶ __ob__: Observer {value: Array(4), shallow: false}
        ▶ [[Prototype]]: Array
        ▶ __ob__: Observer {value: {...}, shallow: false}
        ▶ get groupName: f reactiveGetter()
        ▶ set groupName: f reactiveSetter(newVal)
        ▶ get keywords: f reactiveGetter()
        ▶ set keywords: f reactiveSetter(newVal)
        ▶ [[Prototype]]: Object
        ▶ 1: {__ob__: Observer}
        ▶ 2: {__ob__: Observer}
        length: 3
        ▶ __ob__: Observer {value: Array(3), shallow: false}
        ▶ [[Prototype]]: Array
      startDate: "2022-06-01"
      timeUnit: "month"

```

즉, 객체만 제대로 만들어 API 를 사용하기만 하면,  
데이터를 받아와서 그래프를 바꿀 것이다.

`DataSendForm.vue` 의 `<template>` 을 다음과 같이 작성한다.

```

<template>
  <div>
    <div>시작일 <input type="date" v-model="startDate" /></div>
    <div>종료일 <input type="date" v-model="endDate" /></div>
    <select v-model="timeUnit">
      <option value="date">일간</option>
      <option value="week">주간</option>
      <option value="month">월간</option>
    </select>
    <div>
      그룹명: <input type="text" v-model="userInputGroupName" />
      <button v-on:click="tempGroupAdd">추가</button>
      {{ tempGroupName }}
    </div>
    <div>
      키워드: <input type="text" v-model="userInputKeyword" />
      <button v-on:click="tempKeywordAdd">추가</button>
      {{ tempKeywords }}
    </div>
    <div>
      <button v-on:click="makeGroup">그룹 확정</button>
    </div>
    <div>
      <div>사용자 입력 그룹별 키워드</div>
      <div v-if="keywordGroups.length">
        <div v-for="(keywordGroup, index) in keywordGroups" v-bind:key="index">

```

```

        <div>그룹 이름: {{ keywordGroup.groupName }}</div>
        <div>그룹 키워드: {{ keywordGroup.keywords }}</div>
      </div>
    </div>
  </div>
  <div>
    <input type="submit" v-on:click="sendResultToApi" />
  </div>
</div>
</template>

```

Bootstrap 도 쓰지 않고, `<input>`, `<select>`, `<button>` 등 HTML 에서 기본 제공하는 태그들로 작성했다.

각각의 데이터를 `v-model` 로 연결해두고, `<button>` 에 이벤트를 달아두었다.

화면에 붙일 것은 콧수염 `{{ }}` 으로 작성했다.

`v-for` 도 사용했는데, 그룹이 여러 개일 수 있기 때문이다.

`v-if` 처리를 한 이유는, `keywordGroup` 이 아직 아무것도 없을 때는 보여주지 않기 위해서이고,

`v-for` 에서 `index` 를 키로 설정했다.

submit 을 클릭할 때, `sendResultToApi()` 메서드가 작동하도록 설정했는데, 이 버튼을 누르면 최종 데이터가 완성되고 서버로 전송된다.

`<script>` 의 `data()` 부분은 다음과 같다.

```

export default {
  name: "DataSendForm",
  data() {
    return {
      startDate: "",
      endDate: "",
      timeUnit: "month",
      keywordGroups: [],
      userInputGroupName: "",
      userInputKeyword: "",
      tempGroupName: "",
      tempKeywords: [],
      toApiData: {},
    };
  },
},

```

필요한 데이터를 기입하고, 이 데이터를 최종적으로 모아서 `toApiData()` 를 사용해 서버로 보낼 것이다.

`methods` 객체는 다음과 같다.

```

methods: {
  tempGroupAdd() {
    this.tempGroupName = this.userInputGroupName;
  },
  tempKeywordAdd() {
    this.tempKeywords.push(this.userInputKeyword);
    this.userInputKeyword = "";
  },
  makeGroup() {
    this.keywordGroups.push({
      groupName: this.tempGroupName,
      keywords: this.tempKeywords,
    });
    this.tempGroupName = "";
    this.tempKeywords = [];
    this.userInputGroupName = "";
  },
  sendResultToApi() {
    this.toApiData = {
      startDate: this.startDate,
      endDate: this.endDate,
      timeUnit: this.timeUnit,
    };
  },
}

```

```

        keywordGroups: this.keywordGroups,
      };
      console.log(this.toApiData);
    },
  },
},

```

작성한 메서드는 총 4개이다.

`tempGroupAdd()`

그룹명 옆에 추가 버튼을 누를 시에 작동된다. 사용자 입력 그룹 이름을 `tempGroupName` 에 임시로 보관한다.

`tempKeywordAdd()`

키워드 옆에 추가 버튼을 누를 시에 작동된다. 사용자 입력 키워드를 `tempKeywords` 배열에 `push()` 하고, 사용자 편의를 위해 입력창에서 키워드를 지워준다.


`makeGroup()`


그룹 확정 버튼을 누를 때 동작한다. `keywordGroups` 에 `push()` 하는데, `groupName`, `keywords` 객체 형태로 `push()` 하며, 추가 버튼 옆에 표시된 데이터들과 그룹명 입력창을 초기화한다.


`sendResultToApi()`

최종적으로 데이터 형태를 만들어 서버로 보낼 준비한다.


다음과 같은 형태로 만들어질 것이다.

시작일 2022-06-01 

종료일 2022-08-01 

월간 

그룹명:

키워드:   

사용자 입력 그룹별 키워드

그룹 이름: 코로나

그룹 키워드: ["코로나", "covid", "백신", "거리두기"]

그룹 이름: 금리

그룹 키워드: ["금리", "빅스텝"]

그룹 이름: 누리호

그룹 키워드: ["누리호", "항우연"]

제출 버튼 누를 시,

```

▼ {__ob__: Observer} ⓘ
  endDate: "2022-08-01"
  ▼ keywordGroups: Array(3)
    ▼ 0:
      groupName: "코로나"
      ▼ keywords: Array(4)
        0: "코로나"
        1: "covid"
        2: "백신"
        3: "거리두기"
        length: 4
        ▶ __ob__: Observer {value: Array(4), shallow: false, mock: false, dep: Dep, vmCount: 0}
        ▶ [[Prototype]]: Array
        ▶ __ob__: Observer {value: {...}, shallow: false, mock: false, dep: Dep, vmCount: 0}
        ▶ get groupName: f reactiveGetter()
        ▶ set groupName: f reactiveSetter(newVal)
        ▶ get keywords: f reactiveGetter()
        ▶ set keywords: f reactiveSetter(newVal)
        ▶ [[Prototype]]: Object
      ▶ 1: {__ob__: Observer}
      ▶ 2: {__ob__: Observer}
      length: 3
      ▶ __ob__: Observer {value: Array(3), shallow: false, mock: false, dep: Dep, vmCount: 0}
      ▶ [[Prototype]]: Array
    startDate: "2022-06-01"
    timeUnit: "month"

```

다음과 같이, 브라우저에 콘솔로 잘 찍힌다.

중요한것은, 콘솔에 찍힌 데이터가 우리가 원하는 형태로 준비되었는지도이다. 즉, 이와 같이 콘솔에 찍힌 데이터는 다음 **JSON** 포맷과 일치해야한다.

```

{
  "startDate": "2022-06-01",
  "endDate": "2022-08-01",
  "timeUnit": "month",
  "keywordGroups": [
    {
      "groupName": "코로나",
      "keywords": ["코로나", "covid", "백신", "거리두기"]
    },
    {
      "groupName": "금리",
      "keywords": ["금리", "빅스텝", "파월"]
    },
    {
      "groupName": "누리호",
      "keywords": ["누리호", "항우연"]
    }
  ]
}

```

이제 서버로 보내보자.

**DataSendForm.vue**

```
import { dataLab } from "../utils/axios";
```

Api 임포트 후 **sendResultToApi()** 앞에 **async** 를 붙이고, 다음 두 줄을 추가한다.



```

async sendResultToApi() {
  this.toApiData = {
    startDate: this.startDate,
    endDate: this.endDate,
    timeUnit: this.timeUnit,
    keywordGroups: this.keywordGroups,
  };
  await dataLab.post(this.toApiData);
  location.reload();
},

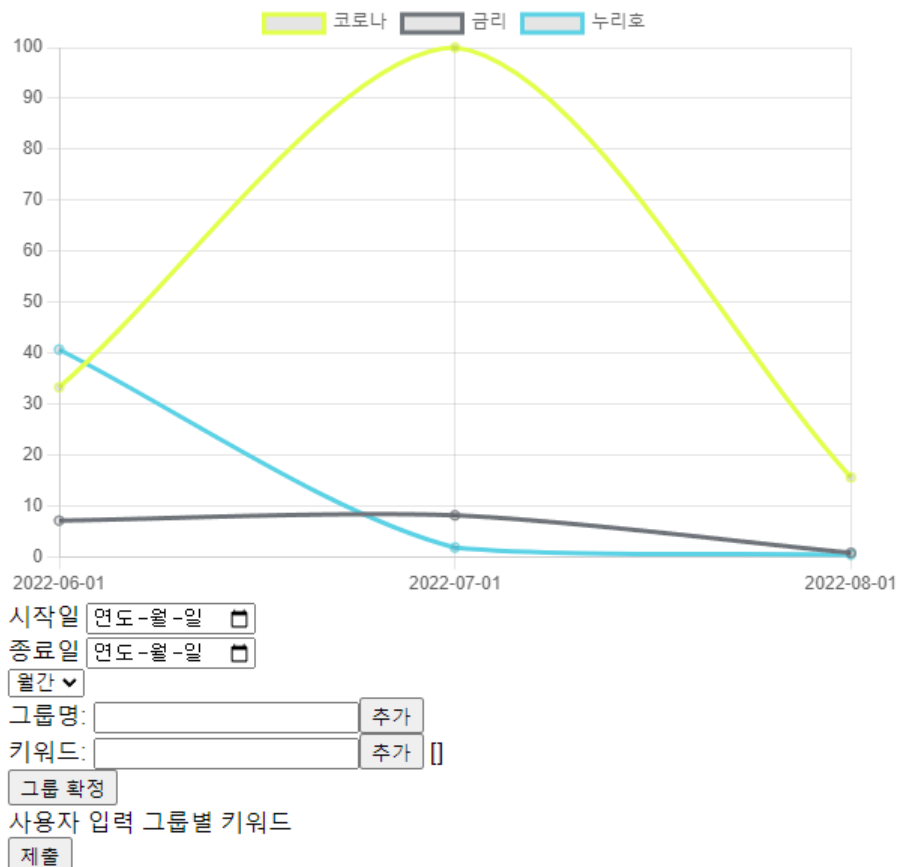
```

`post` 실행해서, 사용자가 입력한 데이터로, 서버의 `chart.json` 을 갱신한다.

그리고, `location.reload()` 를 사용했는데 이것은 새로고침이다. 왜 썼는가? 새로고침을 사용했을 경우 장점을 생각해보자.

`post` 가 실행되었기 때문에 서버의 `chart.json` 은 이미 바뀌어 있을 것이고, 새로고침이 되면서 모든 컴포넌트는 새로 그려지는데, `App.vue` 안에 `created()` 구문이 실행되어 서버에 저장된 `chart.json` 으로, 그래프를 바꿀 것이다.

만약 새로고침을 쓰지 않는다면? 받아온 데이터를 파싱하는 `reduce()` 와, 색깔을 랜덤으로 지정하는 `makeColor()` 를 `DataSendForm.vue` 에 똑같은 내용으로 다시 적거나, 두 개의 함수를 쓰기 위해 모듈 분리가 필요하고, 사용자가 적어둔 그룹 내역을 새로 지우는 과정까지 추가해야한다. 여러모로보나, 지금 새로고침 한 번 하면 해결될 문제가 길어지는 것이다.



Bootstrap 을 설치하자.

```
$ npm i bootstrap@4.6.1 bootstrap-vue
```

**bootstrap** 은 반드시 버전 신경써서 깔아야한다.

**/src/main.js** 를 다음과 같이 수정한다.

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'

// BootstrapVue 임포트
// bootstrap.css, bootstrap-vue.css 도 함께 가져와야한다.
import { BootstrapVue } from "bootstrap-vue";
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

// Vue 에서 전역으로, BootstrapVue 사용
Vue.use(BootstrapVue)

Vue.config.productionTip = false

new Vue({
  store,
  render: h => h(App)
}).$mount('#app')
```

이제 **DataSendForm.vue** 에서, 적용해보자.

```
<template>
  <div>
    <b-card>
      <div class="date-container">
        <label for="startDate">시작일</label>
        <b-form-datepicker
          id="startDate"
          v-model="startDate"
        ></b-form-datepicker>
      </div>
      <div class="date-container">
        <label for="endDate">종료일</label>
        <b-form-datepicker id="endDate" v-model="endDate"></b-form-datepicker>
      </div>
      <div class="select-period-container">
        <b-form-select
          class="time-select"
          v-model="timeUnit"
          :options="timeUnitOptions"
        ></b-form-select>
      </div>
      <b-input-group class="group-name-container" prepend="그룹명">
        <b-form-input v-model="userInputGroupName"></b-form-input>
        <b-input-group-append>
          <b-button variant="outline-success" v-on:click="tempGroupAdd"
            >추가</b-button>
        </b-input-group-append>
      </b-input-group>
      <div class="printed-groupname-container" v-if="tempGroupName">
        {{ tempGroupName }}
      </div>
      <b-input-group class="keywords-container" prepend="키워드">
        <b-form-input v-model="userInputKeyword"></b-form-input>
        <b-input-group-append>
```

```

        <b-button variant="outline-success" v-on:click="tempKeywordAdd"
        >추가</b-button>
      >
    </b-input-group-append>
  </b-input-group>
  <div class="printed-keywords-container" v-if="tempKeywords.length">
    {{ tempKeywords }}
  </div>
  <b-button class="make-group-btn" variant="success" v-on:click="makeGroup"
  >그룹 확정</b-button>
  >
  <div v-if="keywordGroups.length">
    <h1>사용자 입력 그룹별 키워드</h1>
    <b-table striped hover :items="keywordGroups"></b-table>
  </div>
  <b-button class="submit-btn" variant="info" v-on:click="sendResultToApi"
  >제출</b-button>
  >
</b-card>
</div>
</template>

```

```

<script>
import { dataLab } from "../utils/axios";
export default {
  name: "DataSendForm",
  data() {
    return {
      startDate: "",
      endDate: "",
      timeUnit: null,
      timeUnitOptions: [
        { value: null, text: "검색기준을 선택해주세요" },
        { value: "month", text: "월간" },
        { value: "week", text: "주간" },
        { value: "date", text: "일간" },
      ],
      keywordGroups: [],
      userInputGroupName: "",
      userInputKeyword: "",
      tempGroupName: "",
      tempKeywords: [],
      toApiData: {},
    };
  },
  methods: {
    tempGroupAdd() {
      this.tempGroupName = this.userInputGroupName;
    },
    tempKeywordAdd() {
      this.tempKeywords.push(this.userInputKeyword);
      this.userInputKeyword = "";
    },
    makeGroup() {
      this.keywordGroups.push({
        groupName: this.tempGroupName,
        keywords: this.tempKeywords,
      });
      this.tempGroupName = "";
      this.tempKeywords = [];
      this.userInputGroupName = "";
    },
    async sendResultToApi() {
      this.toApiData = {
        startDate: this.startDate,
        endDate: this.endDate,
        timeUnit: this.timeUnit,
        keywordGroups: this.keywordGroups,
      };
      await dataLab.post(this.toApiData);
      location.reload();
    },
  },
};
</script>

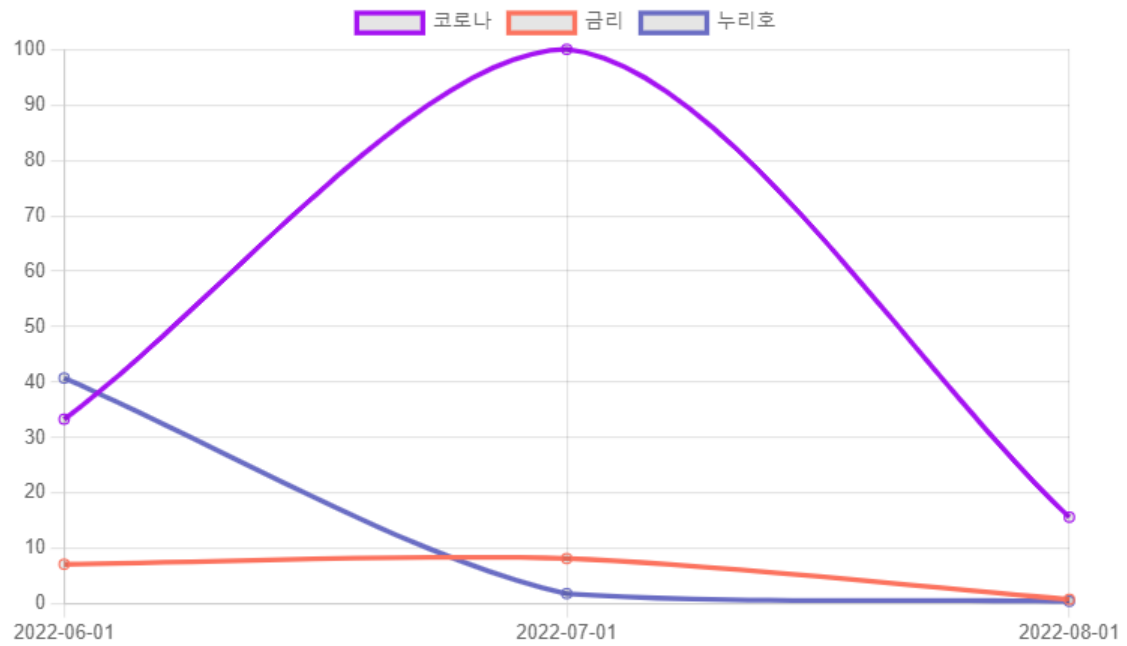
```

```
<style scoped>
.date-container,
.select-period-container,
.keywords-container,
.group-name-container,
.make-group-btn {
  margin-bottom: 10px;
}

.printed-groupname-container,
.printed-keywords-container {
  margin: 20px;
}

.time-select,
.make-group-btn,
.submit-btn {
  width: 100%;
}
</style>
```

결과는 다음과 같다.



시작일

종료일

그룹명

추가

키워드

추가

그룹 확정

제출

## 심화과제

1. 검색조건에 ages 추가하기

ages

array(JSON)

N

연령. 검색사용자의 연령에 따른 조건입니다.

- 설정 안 함: 모든 연령

- 10: 10~19세

- 20: 20~29세

- 30: 30~39세

- 40: 40~49세

- 50: 50~59세

- 60: 60세 이상

## 요청 예

```
curl https://openapi.naver.com/v1/datalab/shopping/categories \
--header "X-Naver-Client-Id: {애플리케이션 등록 시 발급받은 클라이언트 아이디 값}" \
--header "X-Naver-Client-Secret: {애플리케이션 등록 시 발급받은 클라이언트 시크릿 값}" \
--header "Content-Type: application/json" \
-d @<(cat <<EOF
{
  "startDate": "2017-08-01",
  "endDate": "2017-09-30",
  "timeUnit": "month",
  "category": [
    {"name": "패션의류", "param": [ "50000000" ]},
    {"name": "화장품/미용", "param": [ "50000002" ]}
  ],
  "device": "pc",
  "gender": "f",
  "ages": [ "20", "30" ]
}
EOF
)
```

1. API 를 분석해서, 네이버의 전체 사용자가 아니라 해당 연령대의 사용자가 검색한 내역만 가져와보자.
2. API 중 `delete` 는 사용하지 않았다. 어떻게 활용할 수 있는지 고민해보자.