

색 자동 추천기

Color Bound

팀명 : 솔플

팀장 : 김다솜

팀원 : 김다솜

2017-12-01

목 차

1	서론.....	5
1.1	프로젝트 개요.....	5
1.2	프로젝트 목표.....	5
1.3	프로젝트 일정.....	5
1.4	프로젝트 환경.....	6
2	본론.....	7
2.1	Color Bound 서비스 흐름.....	7
2.2	크롤링 개발.....	8
2.2.1	List of Color를 JSON파일로 크롤링.....	8
2.2.2	이미지 크롤링.....	11
2.3	이미지에서 색조합 추출 개발.....	14
2.4	서버 사이드 개발.....	26
2.4.1	MyBatis 연동.....	26
2.4.2	Ajax와 MyBatis간 연동.....	31
2.5	클라이언트 사이드 개발.....	35
2.5.1	아웃풋 기능 추가.....	35
2.5.2	인풋 기능 추가.....	41
3	결론.....	45
3.1	추후 과제.....	45
3.2	발전 방향.....	47

그 림 목 차

[그림 2 - 1] COLOR BOUND 실행 흐름	7
[그림 2 - 2] LIST OF COLOR 중 일부	8
[그림 2 - 3] MACKJSON CLASS의 메소드 MULT	9
[그림 2 - 4] MACKJSON CLASS의 메소드 ADDJSON	9
[그림 2 - 5] MACKJSON의 실행부분	10
[그림 2 - 6] 완성된 LIST OF COLORS의 JSON 파일 중 일부	11
[그림 2 - 7] 이미지 크롤러 소스코드 중 일부 1	12
[그림 2 - 8] 이미지 크롤러 소스코드 중 일부 2	12
[그림 2 - 9] 저장된 고흐 일러스트들	13
[그림 2 - 10] 저장된 STOCKSNAP의 사진들	13
[그림 2 - 11] 이미지에서 한 픽셀당 저장된 R, G, B의 배열화	14
[그림 2 - 12] R, G, B의 값을 배열로 저장	15
[그림 2 - 13] 배열화 된 RGB의 그래프 시각화	15
[그림 2 - 14] 세 개의 배열 -> 1차원 배열 -> 2차원 배열	16
[그림 2 - 15] KMEANS 모듈 IMPORT와 ELBOW 메소드	17
[그림 2 - 16] 프로젝트의 ELBOW 기법 그래프화	18
[그림 2 - 17] ELBOW 기법 예시	18
[그림 2 - 18] 이미지의 K 클러스터링	19
[그림 2 - 19] 3차원 거리계산을 이용한 공식적인 색 변환기	19
[그림 2 - 20] 색 변환 이후 중복 제거	19
[그림 2 - 21] 클러스터 계산 결과 시각화	20
[그림 2 - 22] 클러스터링에 쓰인 이미지	21
[그림 2 - 23] 도출된 이미지의 색 조합	21
[그림 2 - 24] ADOBE COLOR CC와 본 코드와의 결과값 비교	22
[그림 2 - 25] [그림 2-21] PYTHON과 MYSQL 연동 뒤 INSERT 작업 함수	23
[그림 2 - 26] 한 이미지당 작동하는 함수	23
[그림 2 - 27] 폴더 내의 모든 이미지에 작업을 실행하는 함수	24
[그림 2 - 28] CMD창에서 PYTHON 파일 실행	24
[그림 2 - 29] TEST 테이블과 GOGH 테이블의 정보	25
[그림 2 - 30] 입력된 데이터들 확인	25
[그림 2 - 31] 연동을 위해 프로젝트의 POM.XML 에 추가된 DEPENDENCY	26
[그림 2 - 32] ROOT-CONTEXT.XML의 네임스페이스 추가	27
[그림 2 - 33] DATASOURCE 설정, MYBATIS 연결, XML MAPPER 인식	27
[그림 2 - 34] COLORMAPPER.XML 중 일부	28

[그림 2 - 35] COLORDAOIMPL.JAVA 중 일부.....	29
[그림 2 - 36] COLORDAO 테스트 성공	30
[그림 2 - 37] 프론트 페이지와 스프링 컨트롤러간 정보교환 흐름.....	31
[그림 2 - 38] HOME.JSP 안의 COLORAJAX()의 일부분	32
[그림 2 - 39] HOME.JSP 안의 COLORAJAX()의 일부분2.....	33
[그림 2 - 40] HOME.JSP 안의 COLORAJAX()의 일부분2.....	33
[그림 2 - 41] COLORSERVICEIMPL.JAVA의 일부분2	34
[그림 2 - 42] HTML에서 데이터 확인 부분.....	34
[그림 2 - 43] MDN COLOR PICKER TOOL	35
[그림 2 - 44] CPT 에 포함되어 아웃풋을 만들기 위해 이용될 부분.....	36
[그림 2 - 45] 워드카운터 된 데이터들 중 일부.....	36
[그림 2 - 46] 아웃풋 부분을 만들기 위해 참고할 부분.....	37
[그림 2 - 47] CREATETHEMAPALETTE.....	38
[그림 2 - 48] 42CREATETHEMAPALETTE에 쓰인 함수 두 개	38
[그림 2 - 49] COLOR 객체 형식	39
[그림 2 - 50] 각자의 함수에서 쓰인 UPDATE 함수들	40
[그림 2 - 51] CREATETHEMAPALETTE에 쓰인 UPDATETHEMA 함수	40
[그림 2 - 52] AJAX로 받은 데이터로 표현된 아웃풋 결과	41
[그림 2 - 53] 47CPT 에 포함되어 인풋을 만들기 위해 이용될 부분	41
[그림 2 - 54] COLORPICKERSAMPLES_T()안의 PRINTRESULT()함수	42
[그림 2 - 55] PRINTRESULT()의 마지막 부분	43
[그림 2 - 56] 완성된 AJAX로 보낼 데이터를 정하는 인풋 부분	44
[그림 2 - 57] TEST DB에서 #A41313를 검색한 결과	44
[그림 2 - 58] #A41313와 #556B2F 검색 결과	44
[그림 3 - 1] 0개 선택시 고흐 테마 아웃풋	45
[그림 3 - 2] 0개 선택 시 TEST 테마 아웃풋.....	45
[그림 3 - 3] 테마를 선택하는 부분	46
[그림 3 - 4] 눌러야 색 검색이 동작이 되는 부분.....	46
[그림 3 - 5] 대중적이지 않은 색감의 고흐 그림들	47

표 목 차

[표 1 - 1] 프로젝트 일정.....	5
[표 1 - 2] 프로젝트 환경	6

1 서론

1.1 프로젝트 개요

색 자동 추천기 “Color Bound” 란 유저가 원하는 스타일에 맞는 테마에서 자신의 원하는 색에 대해서 그 스타일에 맞는 색들을 추천 받아 색 조합에 도움을 받는 서비스이다. 이번 프로젝트를 통해 이를 웹에서 서비스할 수 있는 도구로 구현해 보고자 한다.

1.2 프로젝트 목표






이 서비스는 객관적으로 호평 받은 색감의 이미지들을 모아 그 이미지의 색 조합을 뽑아내어 이미지의 테마대로 나누어 저장하고, 자신이 원하는 색을 자신이 원하는 스타일의 테마에서 가장 연관성이 높은 색들을 순서대로 추천 받을 수 있게 된다. 즉, 자신이 원하는 색에서 자신이 원하는 스타일의 테마에서 가장 보편적으로 같이 쓰인 품질 높은 색 조합들을 선택 할 수 있는 기회가 제공 된다.

1.3 프로젝트 일정

절차		1주			2주			3주		
자료 조사	벤치마킹 및 크롤링									
개발	서버 개발									
	클라이언트 개발									
디버깅	테스트 및 디버깅									
문서 작성	결과 보고서 및 발표 자료 작성									
결과 발표	최종 발표									

[표 1 - 1] 프로젝트 일정

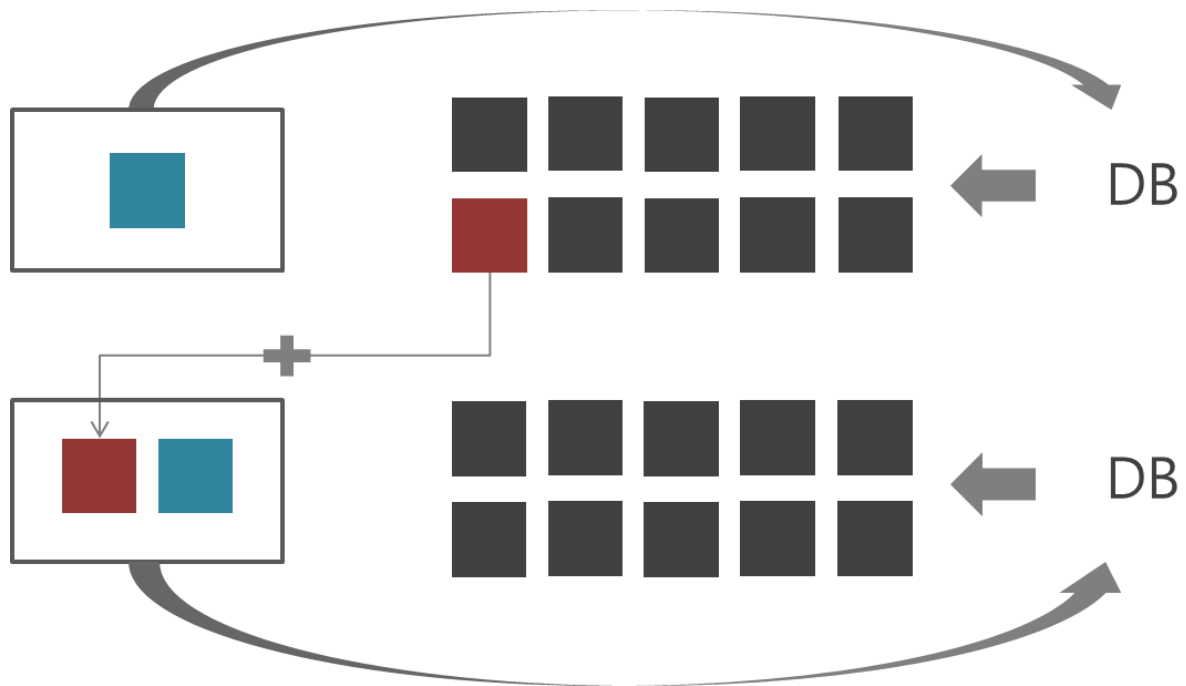
1.4 프로젝트 환경

크롤링		Java 8+
적합한 데이터 추출		Python 3+ (anaconda3, jupyter)
데이터 베이스		MySQL 5.7
서버 사이드 개발		Spring 3.9 MyBatis 3.4.1
클라이언트 사이드 개발		jQuery 2.2.4 jsp 2+

[표 1 - 2] 프로젝트 환경

2 본론

2.1 Color Bound 서비스 흐름















[그림 2 - 1] Color Bound 실행 흐름

Color Bound는 DB에서 한 색에 관련되어 연관성이 있는 데이터들을 모두 검색하여 뽑아내고 보여준다.

- 최초에는 선택된 테마의 데이터베이스에서 모든 색을 목록에 보여준다.
- 그 중에서 하나의 색을 뽑아서 검색하거나 자신이 원하는 색을 검색한다
- 그 색에 연관된 색들이 나열되고, 또 다른 색을 몇 개정도 까지 추가해서 검색이 가능하다.
- 사용자는 이러한 검색으로 자신이 만족스러운 색 조합을 찾고, 조정한다.

2.2 크롤링 개발

2.2.1 List of Color를 JSON파일로 크롤링

	American Bronze	#391802	22%	9%	1%	24°	93%	12%	96%	22%
	American Brown	#804040	50%	25%	25%	0°	33%	38%	50%	50%
	American Gold	#D3AF37	83%	69%	22%	46°	64%	52%	74%	83%
	American Green	#34B334	20%	70%	20%	120°	55%	45%	71%	70%
	American Orange	#FF8B00	100%	55%	0%	33°	100%	50%	100%	100%
	American Pink	#FF9899	100%	60%	60%	359°	100%	80%	40%	100%
	American Purple	#431C53	26%	11%	33%	283°	50%	22%	66%	33%
	American Red	#B32134	70%	13%	20%	352°	69%	42%	82%	70%
	American Rose	#FF033E	100%	1%	24%	346°	100%	51%	99%	100%
	American Silver	#CFCFCF	81%	81%	81%	0°	0%	81%	0%	81%
	American Violet	#551B8C	33%	11%	55%	271°	68%	33%	81%	55%
	American Yellow	#F2B400	95%	71%	0%	180°	100%	48%	100%	95%
	Amethyst	#9966CC	60%	40%	80%	270°	50%	60%	50%	80%
	Android Green	#A4C639	64%	78%	22%	74°	55%	50%	71%	78%
	Anti-Flash White	#F2F3F4	95%	95%	96%	210°	8%	95%	1%	96%
	Antique Brass	#C09575	80%	58%	46%	22°	47%	63%	43%	80%
	Antique Bronze	#665D1E	40%	36%	12%	53°	55%	26%	71%	40%
	Antique Fuchsia	#915C83	57%	36%	51%	316°	22%	46%	37%	57%
	Antique Ruby	#841B2D	52%	11%	18%	350°	66%	31%	80%	52%
	Antique White	#FAEBD7	98%	92%	84%	34°	78%	91%	14%	98%
	Ao (English)	#008000	0%	50%	0%	120°	100%	25%	100%	50%

[그림 2 - 2] List of Color 중 일부

컴퓨터 안에서 표현 될 수 있는 색들의 수는 255의 세제곱이다.

이 모든 수를 사용 한다면 조금이라도 수치가 바뀌어도 검색이 용이하지 않으니 검색해야 할 색이나 검색 되어야 할 색 모두 이 공식적인 색 목록에 있는 수치로 바꾸어 사용하려 한다.

우선 다음 작업에 사용이 용이하게 될 수 있도록 이 목록을 크롤링하여 JSON 파일로 작성하려 한다.

List of colors는 위키피디아(https://en.wikipedia.org/wiki/List_of_colors:_A%E2%80%93F)에서 정보가 제공되고 있다.


```

static void mult(JSONArray arr, String str, int i, int j) {
    String exePath = "C:\\KDS\\chromedriver.exe";
    System.setProperty("webdriver.chrome.driver", exePath);
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless");

    ChromeDriver driver = new ChromeDriver(options);

    driver.get(str);
    String html = driver.getPageSource();
    Document doc = Jsoup.parse(html);
    Elements ele1 = doc.select("table.wikitable.sortable.jquery-tablesorter>tbody>tr>th");
    Elements ele2 = doc.select("table.wikitable.sortable.jquery-tablesorter>tbody>tr>td:nth-child(2)");
    //System.out.print(ele1.text());
    addJSON(arr, ele1, ele2, i, j);

    driver.quit();
}

```

[그림 2 - 3] mackJSON class의 메소드 mult

우선 html상의 데이터를 불러오기 위하여 구글의 크롬 웹 드라이버를 사용하였다.

그리고 불러온 데이터 중에서 목록들 중의 색상 이름과 그에 맞는 색상의 hex값을 배열에 입력한다.

```

static void addJSON(JSONArray arr, Elements ele1, Elements ele2, int i, int j) {
    for(Element link : ele1) {
        arr.add(new JSONObject());
        ((JSONObject)arr.get(i)).put("name", link.text());
        //System.out.println(link.text());
        i++;
    }
    for(Element link : ele2) {
        ((JSONObject)arr.get(j)).put("R", Color.decode(link.text()).getRed());
        ((JSONObject)arr.get(j)).put("G", Color.decode(link.text()).getGreen());
        ((JSONObject)arr.get(j)).put("B", Color.decode(link.text()).getBlue());
        j++;
    }
}

```

[그림 2 - 4] mackJSON class의 메소드 addJSON

그리고 JSON 파일 형식으로 입력하기 위하여 org.json.simple.* 을 import 하였고, 그에 따라 JSON을 배열 형식으로 만들어 그 속의 한 오브젝트당 name, R, G, B 값을 집어넣었다.

```

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

        JSONArray arr = new JSONArray();

        mult(arr, "https://en.wikipedia.org/wiki/List_of_colors:_A-F", arr.size(), arr.size());
        System.out.println(arr.size());
        mult(arr, "https://en.wikipedia.org/wiki/List_of_colors:_G-M", arr.size(), arr.size());
        System.out.println(arr.size());
        mult(arr, "https://en.wikipedia.org/wiki/List_of_colors:_N-Z", arr.size(), arr.size());
        System.out.println(arr.size());
        System.out.println(arr);

        JSONParser parser = new JSONParser();
        try {

            FileWriter file = new FileWriter("c:\\test.json");
            file.write(arr.toJSONString());
            file.flush();
            file.close();

            Object read = parser.parse(new FileReader("c:\\test.json"));
            //System.out.print(read);

        } catch (IOException | ParseException e) {
            e.printStackTrace();
        }
    }
}

```

[그림 2 - 5] mackJSON의 실행부분

List of colors가 세 페이지에 나뉘져 정보가 제공되기 때문에 세 번의 과정 동안 JSONArray 안에 JSONObject를 집어넣는 작업을 실행시키고, 완성된 JSONArray를 java.io.FileWriter로 파일로 저장했다.

```

{} test.json
1 [
2   {
3     "R": 0,
4     "B": 186,
5     "G": 72,
6     "name": "Absolute zero"
7   },
8   {
9     "R": 76,
10    "B": 39,
11    "G": 47,
12    "name": "Acajou"
13  },
14  {
15    "R": 176,
16    "B": 26,
17    "G": 191,
18    "name": "Acid green"
19  },
20  {
21    "R": 124,
22    "B": 232,
23    "G": 185,
24    "name": "Aero"
25  },
26  {
27    "R": 201,
28    "B": 229,
29    "G": 255,
30    "name": "Aero blue"
31  },

```

[그림 2 - 6] 완성된 List of Colors의 JSON 파일 중 일부

2.2.2 이미지 크롤링

이미지에서 색 데이터를 추출하기 위해서 이미지를 크롤링 해야 한다.

우선 객관적으로 색감이 좋은 평가의 사진이 모여있는 StockSnap(<http://stocksnap.io/>)의 사진들과 사진들이 아닌 일러스트도 실제 구현을 살펴보기 위해 빈센트 반 고흐의 이미지들을 크롤링 하기로 했다.

html상의 데이터를 불러오기 위해 이번에도 구글 크롬 웹드라이버를 사용하였다.

```

while(true) {
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    driver.executeScript("scroll(0,document.body.scrollHeight);");
    if(i==100) {
        break;
    }

    i++;
    System.out.println(i);
}

```

[그림 2 - 7] 이미지 크롤러 소스코드 중 일부 1

Stocksnap 같은 경우에는 스크롤을 내려야 계속해서 이미지가 로딩 되기 때문에 sleep로 딜레이를 주면서 드라이버의 자바스크립트 실행기능(driver.executeScript())으로 스크롤을 아래로 내려주어야 했다.

```

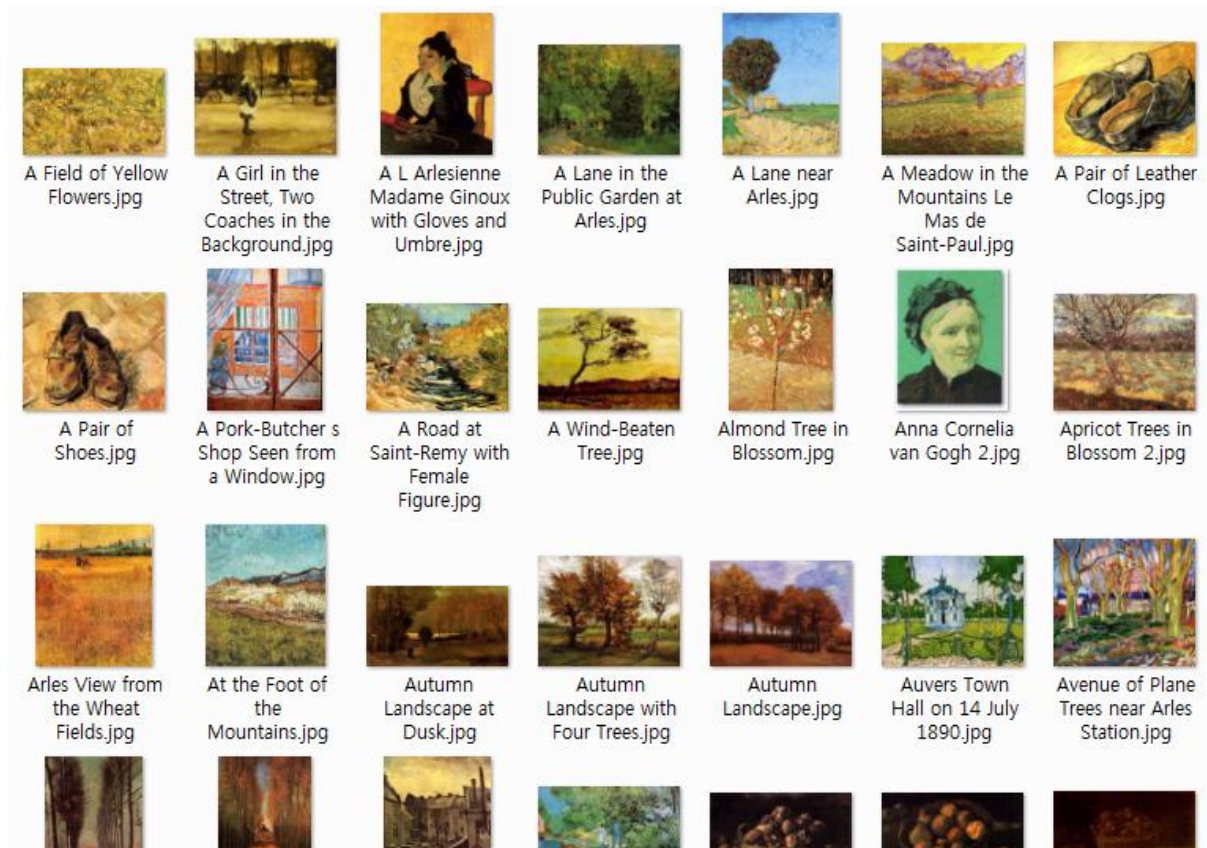
try {
    File file = new File("C:\\KDS\\imgs\\"+fileName);
    URL url = new URL(link.attr("src"));
    URLConnection conn = url.openConnection();
    conn.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows NT 6.1; WOW64
    conn.connect();
    FileUtils.copyInputStreamToFile(conn.getInputStream(), file);
} catch (Exception e) {
}

```

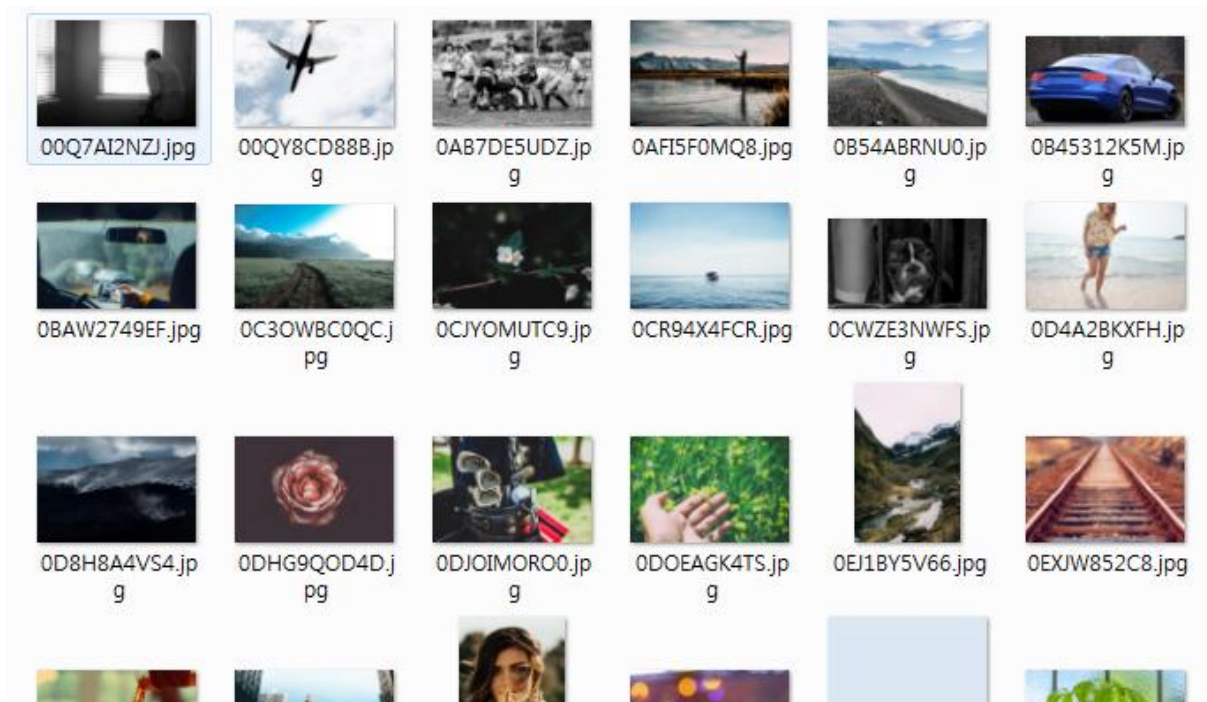
[그림 2 - 8] 이미지 크롤러 소스코드 중 일부 2

그리고 stocksnap에서는 로봇을 허가하지 않는 것인지 URLconnection으로 연결했을 때 접속 허가가 뜨지 않았다.

User-Agent 프로퍼티를 "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0"으로 바꾸어야만 연결이 허락되었다.



[그림 2 - 9] 저장된 고흐 일러스트들



[그림 2 - 10] 저장된 StockSnap의 사진들

2.3 이미지에서 색조합 추출 개발

이미지에서 뽑아낸 한 그림당 색들의 조합을 베이스로 서비스가 완성 되기 때문에 제일 중요한 단계이다.

3차원으로 표현된 R, G, B의 값으로 표현된 포인트들을 K-means를 사용하여 클러스터 군집을 나누어 그 클러스터의 평균값을 대표적인 색으로 취급하여 뽑아 낼 것이기 때문에 그래프 그리기에 용이하면서 K-means 모듈이 있는 Python과 그래프를 자주 확인하기에 편리한 Jupyter(Anaconda3)로 작업을 했다.



[160	136,	36]
[181	159,	60]
[177	158,	56]
[193	186,	95]
[209	203,	127]
[226	225,	145]
[241	246,	143]

[그림 2 - 11] 이미지에서 한 픽셀당 저장된 R, G, B의 배열화

우선 이미지를 수치 데이터화 해서 클러스터링 하기 위해서 위의 그림과 같이 한 픽셀당 저장되어있는 R, G, B의 값을 배열로 저장했다.


```
In [2]: r = []
g = []
b = []
for i in gh :
    if im.size[0]>im.size[1]:
        for j in range(0, im.size[1]) :
            r.append(i[j, 0])
            g.append(i[j, 1])
            b.append(i[j, 2])
    else :
        for j in range(0, im.size[0]) :
            r.append(i[j, 0])
            g.append(i[j, 1])
            b.append(i[j, 2])
```

[그림 2 - 12] R, G, B의 값을 배열로 저장

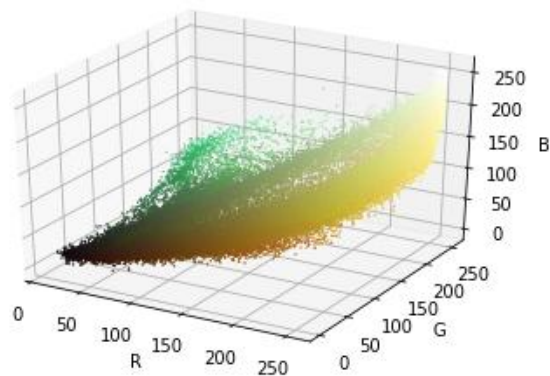
우선 위의 그림과는 다르게 배열을 세 개로 나누어 저장했는데, 그래프를 그리는 함수에서 편하게 쓰기 위해 세 개로 나누어 저장했다.

```
In [5]: fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = r
    ys = g
    zs = b
    ax.scatter(xs, ys, zs, s=0.1, c=color, marker=m)

ax.set_xlabel('R')
ax.set_ylabel('G')
ax.set_zlabel('B')

plt.show()
```



[그림 2 - 13] 배열화 된 RGB의 그래프 시각화

위의 이미지는 수치화된 배열을 R, G, B의 포인트들로 뿌려 그려낸 3차원 스캐터 그래프이다.

클러스터를 시작하기에 충분한 이미지의 수치화가 완성되었다.

```
In [6]: kluer = []  
        for i in range(len(r)) :  
            kluer.append(str(r[i])+" "+str(g[i])+" "+str(b[i]))
```

```
In [7]: kluer = list(set(kluer))
```

```
In [8]: realColor = []  
        for i in kluer :  
            realColor.append([i.split(' ')[0], i.split(' ')[1], i.split(' ')[2]])
```

[그림 2 - 14] 세 개의 배열 -> 1차원 배열 -> 2차원 배열

그리고 이미지에서 적게 쓰였지만 포인트가 될 수도 있는 색이 묻히는 현상을 막기 위해 수치화된 배열을 중복 제거하기 위해 set으로 만든 뒤 list로 만드는 방법을 사용하기 위하여 RGB를 공백으로 나눈 하나의 문자열로 입력하고 만들어 1차원의 배열로 만든 뒤 중복제거를 하고, 중복제거를 하고 난 결과물을 sklearn.cluster.k_means에서 구동하기 위하여 split() 함수를 사용하여 R, G, B를 2차원 배열로 만들었다


```
In [9]: from sklearn.cluster import KMeans
import sklearn.cluster.k_means_
```

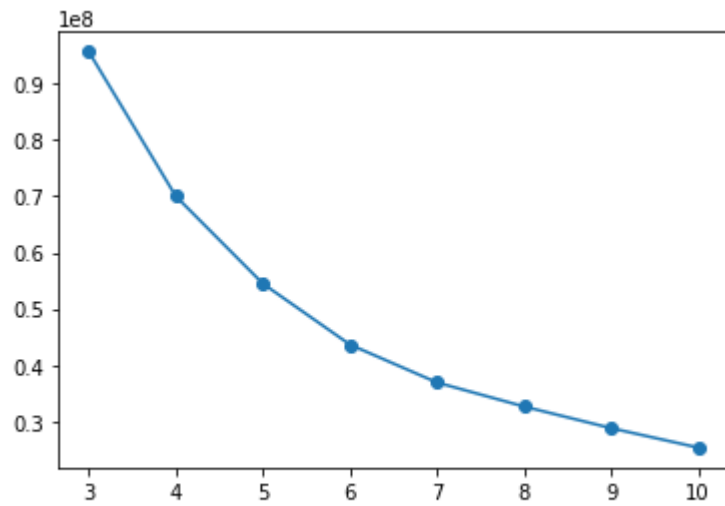
```
In [10]: def elbow(X):
sse = []
a = []
for i in range(3, 11):
    km = KMeans(n_clusters=i, init='k-means++', random_state=0)
    km.fit(X)
    sse.append(km.inertia_)
plt.plot(range(3, 11), sse, marker='o')
plt.show()
b=0

for j in range(len(sse)):
    a.append((sse[j]-sse[j-1]))
    c = abs(a[j-1])-abs(a[j])
    if a[j]>a[j-1] :
        if b<c :
            b=c
            global d
            d=j+3

elbow(realColor)
print (d)
```

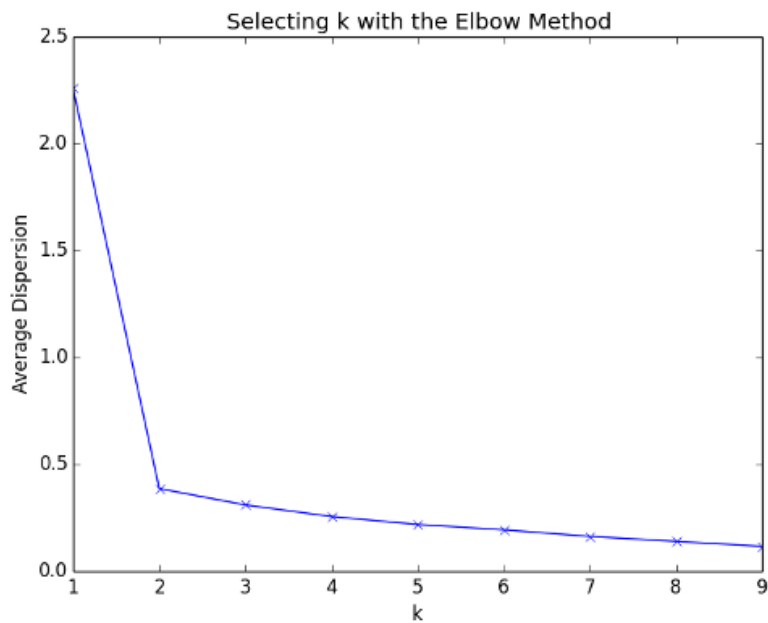
[그림 2 - 15] kmeans 모듈 import와 elbow 메소드

sklearn.cluster.k_means 모듈을 임포트시키고, 이미지에서 최적의 클러스터가 몇 개인지 세기 위해 inertia 라는 attribute를 이용한다. 이 값은 가장 가까운 클러스터 센터에 샘플의 제곱 거리의 합이다. 이 값으로 그래프를 그려 팔꿈치처럼 꺾이는 부분이 가장 최적의 클러스터의 수이다.



5

[그림 2 - 16] 프로젝트의 elbow 기법 그래프화



[그림 2 - 17] elbow 기법 예시

원래는 elbow그래프의 예시처럼 육안으로도 보이게 꺾이는 부분이 보여야하지만 육안으로 보이지 않기에 그래프 기울기차이를 계산하여 제일 팔꿈치에 가까운 부분을 뽑아냈다. 이 이미지의 결과는 5이다.

```
d = d
model = KMeans(n_clusters=d, init="k-means++", random_state=1).fit(realColor)

e = model.cluster_centers_
```

[그림 2 - 18] 이미지의 k 클러스터링

다음으로 최적의 클러스터의 결과로 클러스터의 수를 설정하고 k 클러스터링을 실시했다.

```
def nameConverter(r, g, b):
    r = int(r)
    g = int(g)
    b = int(b)
    minest = 500
    crrectColor = {}
    for i in colorList :
        R = i.get("R")
        G = i.get("G")
        B = i.get("B")

        distance = (r-R)*(r-R) + (g-G)*(g-G) + (b-B)*(b-B)
        distance = math.sqrt(distance)
        if minest > distance :
            minest = distance
            crrectColor = i

    return crrectColor
```

[그림 2 - 19] 3차원 거리계산을 이용한 공식적인 색 변환기

```
In [13]: e_2 = []
        for i in e:
            a = nameConverter(i[0], i[1], i[2])
            e_2.append(str(a.get("R"))+" "+str(a.get("G"))+" "+str(a.get("B")))

        print(len(e_2))
        e_2 = list(set(e_2))
        print(len(e_2))
        e = []
        for i in e_2:
            e.append([int(i.split(' ')[0]), int(i.split(' ')[1]), int(i.split(' ')[2])])

5
5
```

```
In [14]: d = len(e)
        e = np.asarray(e)
```

[그림 2 - 20] 색 변환 이후 중복 제거

한 자리수의 클러스터 개수라면 색 변환 이후의 중복 제거가 필요하지 않지만, 고흐 테마에서는 선택 할 수 있는 색들을 억지로 늘려보고 구현 시 어떻게 되는지 실험해 보기 위해 elbow 에서 도출된 최적의 클러스터 수에서 10을 곱해 클러스터를 계산하였기 때문에 이 과정을 추가하였다.

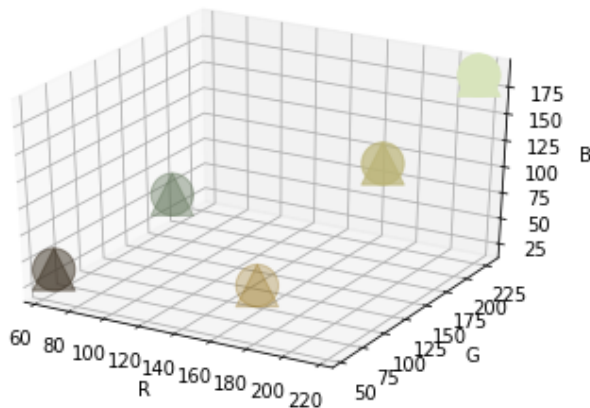
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
color = []
print (e[0,0])
for a in range(len(e)):
    color.append('#%02x%02x%02x' % (int(e[a, 0]), int(e[a, 1]), int(e[a, 2])))

print (color)
for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = e[range(d), 0]
    ys = e[range(d), 1]
    zs = e[range(d), 2]
    ax.scatter(xs, ys, zs, s=500, c=color, marker=m)

ax.set_xlabel('R')
ax.set_ylabel('G')
ax.set_zlabel('B')

plt.show()
```

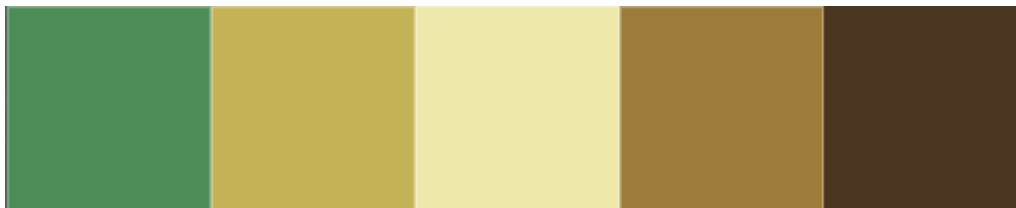
```
65
['#413628', '#bbb477', '#63775b', '#967117', '#d8e4bc']
```



[그림 2 - 21] 클러스터 계산 결과 시각화

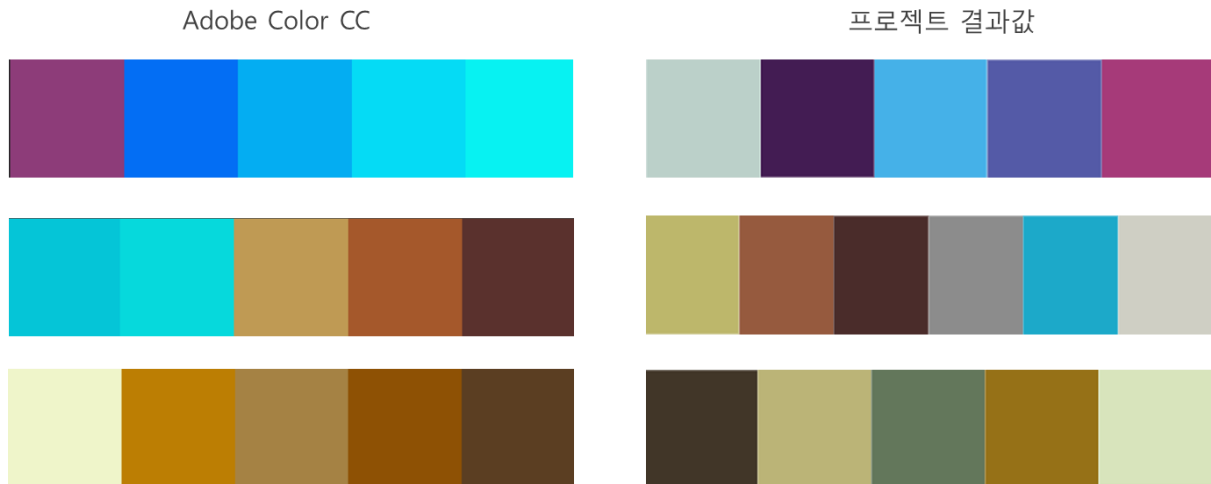


[그림 2 - 22] 클러스터링에 쓰인 이미지



[그림 2 - 23] 도출된 이미지의 색 조합

이에 따라 이미지의 색 조합이 뽑혔지만 이것이 알맞은지 또는 정확도가 높은지 판단 할 수 있는 기준이 없어 이미 색감을 뽑아내는 서비스가 실행되고 있는 Adobe Color CC와 비교작업에 들어갔다.



[그림 2 - 24] Adobe Color CC와 본 코드와의 결과값 비교

비교결과 Adobe Color CC보다 비교적 색상이 다양하게 나오고, 클러스터 수를 이미지에 맞도록 조정 할 수 있다는 장점이 있었지만 프로젝트 결과값은 색들이 공통적으로 채도가 살짝 낮다는 단점이 있었다.

이 단점은 이미지의 RGB 클러스터 군집들의 '평균값'이 결과물이므로 클러스터 군집 안에서 채도가 높은 색은 무시되고 그보다 조금 적은 채도의 색이 선택 되어 나온 현상임으로 추리하고, 클러스터 군집에서 클러스터의 평균 값과 색상이 가까우면서 R, G, B의 값 차이가 극명하게 나는 것 (R, G, B의 수치가 비슷할수록 채도가 낮다)을 선택해야 하는 방법을 구현할 방법을 강구하였지만 현재 파이썬의 kmeans 모듈로는 구현 가능한 방법을 찾지 못했다

다음은 위의 작업의 코딩을 간소화 시킨 뒤 도출된 색상들의 헥사값을 데이터베이스에 넣는 작업을 실행했다.

다음은 코드의 일부이다.

```
In [7]: def mysql(name, colors):
        conn = pymysql.connect(host='localhost', user='root', password='248624', db='project', charset='utf8', autoc
        try :
            curs = conn.cursor()
            sql = "INSERT INTO `gogh`(`name`, `colors`) VALUES (%s,%s)"
            curs.execute(sql, (name, colors))
        except MySQLError as err:
            print(err)
        finally :
            conn.close()
```

[그림 2 - 25] [그림 2-21] Python과 MySQL 연동 뒤 insert 작업 함수

위 그림은 project DB안에서 gogh 테이블의 name와 colors 안에 각각 파일 이름과 도출된 색 조합들을 하나의 스트링 형식으로 만든 데이터를 넣는 작업이다.

```
In [11]: def input_cluster(filename):
        #filename = "C:\Users\KDS\Documents\1MBFDUHLZ4.jpg"
        im = Image.open("C:\Users\KDS\Documents\1MBFDUHLZ4.jpg")
        gh = np.array(im)
        kluer = []
        realColor = []
        for i in gh :
            if im.size[0]>im.size[1]:
                for j in range(0, im.size[1]) :
                    kluer.append(str(i[j, 0])+" "+str(i[j, 1])+" "+str(i[j, 2]))
            else :
                for j in range(0, im.size[0]) :
                    kluer.append(str(i[j, 0])+" "+str(i[j, 1])+" "+str(i[j, 2]))
        kluer = list(set(kluer))

        for i in kluer :
            realColor.append([i.split(' ')[0], i.split(' ')[1], i.split(' ')[2]])

        d = elbow(realColor)*10
        e = KMeans(n_clusters=d, init='k-means++', random_state=0).fit(realColor).cluster_centers_

        e_2 = []
        for i in e:
            a = nameConverter(i[0], i[1], i[2])
            e_2.append(str(a.get("R"))+" "+str(a.get("G"))+" "+str(a.get("B")))

        e_2 = list(set(e_2))
        e = []
        for i in e_2:
            e.append([int(i.split(' ')[0]), int(i.split(' ')[1]), int(i.split(' ')[2])])

        d = len(e)
        e = np.asarray(e)

        color = []
        for a in range(len(e)):
            color.append(RGBtoHEX(e[a, 0], e[a, 1], e[a, 2]))

        colors = ""

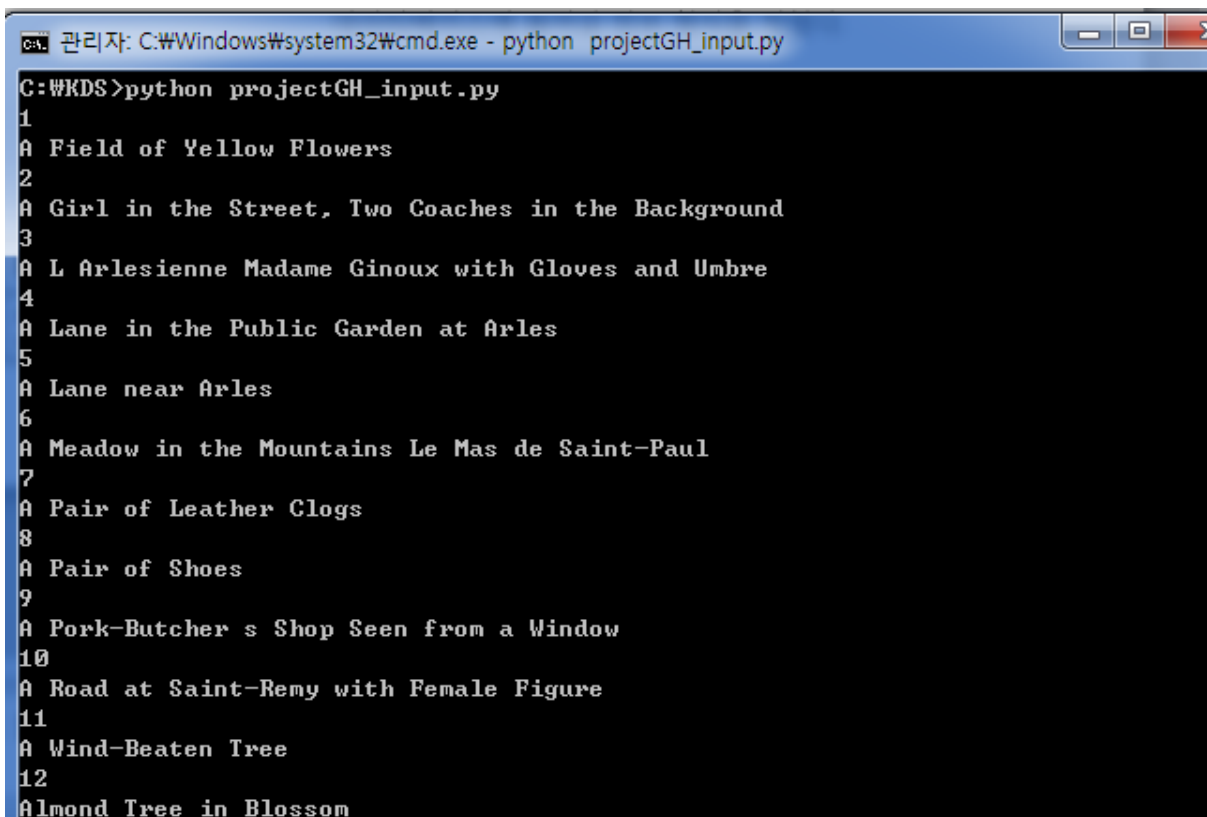
        for i in color :
            colors = colors+i+" "

        mysql(filename, colors)
```

[그림 2 - 26] 한 이미지당 작동하는 함수

```
In [12]: num = 1
         for i in file_list :
             input_cluster(i)
             print(num)
             print(i)
             num = num+1
```

[그림 2 - 27] 폴더 내의 모든 이미지에 작업을 실행하는 함수



```
C:\WKS>python projectGH_input.py
1
A Field of Yellow Flowers
2
A Girl in the Street, Two Coaches in the Background
3
A L Arlesienne Madame Ginoux with Gloves and Umbre
4
A Lane in the Public Garden at Arles
5
A Lane near Arles
6
A Meadow in the Mountains Le Mas de Saint-Paul
7
A Pair of Leather Clogs
8
A Pair of Shoes
9
A Pork-Butcher s Shop Seen from a Window
10
A Road at Saint-Remy with Female Figure
11
A Wind-Beaten Tree
12
Almond Tree in Blossom
```

[그림 2 - 28] cmd창에서 python 파일 실행

다음은 MySQL 안에 있는 파이선에서 연결했던 테이블들의 정보이다.

Gogh 테이블은 클러스터의 수가 많아 100개의 문자열로는 충분하지 않았기 때문에 varchar 형식으로 1000개 까지 입력될 수 있도록 했다.


```
mysql> desc test;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | char(100)     | NO   | PRI | NULL    |       |
| colors| char(100)     | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> desc gogh;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | char(100)     | NO   | PRI | NULL    |       |
| colors| varchar(1000) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

[그림 2 - 29] test 테이블과 gogh 테이블의 정보

```
mysql> select * from test;
+-----+-----+-----+-----+-----+-----+
| name          | colors          |
+-----+-----+-----+-----+-----+-----+
| 004EAQQ9SC    | #6e7f80 #523b35 #cd9575 #965a3e #141414 |
| 00Q7AI2NZJ    | #f5f5f5 #3b3c36 #676767 #b3b3b3 #1b1b1b |
| 00QY8CD88B    | #f2f3f4 #080808 #aec6cf #555555 #c4d8e2 |
| 01IUV64AGL    | #71706e #1b1b1b #ebecf0 #bea493 #543d37 |
| 01SOPQS4UN    | #3b331c #c8ad7f #6e6e30 #92a1cf #918151 |
| 02GZUTSQ57    | #2887c8 #00468c #355e3b #8cbcd6 #9fa91f |
| 02JCMP6EPE    | #545a2c #a6a6a6 #738678 #004953 #2a2f23 |
| 02JPTZMK3V    | #233067 #188bc2 #004f98 #c4d8e2 #8a7f80 |
| 02PW14YN0P    | #3b2f2f #d8d8d8 #00416a #72a0c1 #1164b4 |
| 02TRPYPZCI    | #dcdcdc #674c47 #ebecf0 #a6a6a6 #080808 |
| 02VKCD76AO    | #6a5445 #bfafb2 #1b1811 #98817b #4c2f27 |
| 0302SD6X8A    | #000000 #986960 #664228 #bea493 #2c1608 |
| 03CS4FNBSJ    | #556b2f #d1bea8 #354230 #acbf60 #6f9940 |
| 03M9ADMY7I    | #413628 #d99a6c #a45a52 #6f4e37 #2a2f23 |
| 03OKBK3DWH    | #1b1811 #444c38 #bfafb2 #71706e #2a2f23 |
| 03P9371157    | #000000 #664228 #000000 #671515 #422802 |
```

[그림 2 - 30] 입력된 데이터들 확인

2.4 서버 사이드 개발

2.4.1 MyBatis 연동

클라이언트와 서버의 데이터베이스간 원활한 통신을 위하여 Spring과 MyBatis의 연동 작업을 통해 SQL처리를 목표로 삼고 작업을 시작하였다.

스프링 프로젝트는 Spring Legacy Project로 시작하였다.

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.41</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.1</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.0</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${org.springframework-version}</version>
  <exclusions>
    <!-- Exclude Commons Logging in favor of SLF4j -->
    <exclusion>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
```

[그림 2 - 31] 연동을 위해 프로젝트의 pom.xml 에 추가된 dependency

Spring과 MyBatis 사이에 두 프레임 워크의 접착제 역할을 하는 MyBatis-Spring, MyBatis, mysql을 포함한 연동에 필요한 모듈들을 추가한다.

<input checked="" type="checkbox"/>		aop - http://www.springframework.org/schema/aop
<input checked="" type="checkbox"/>		beans - http://www.springframework.org/schema/beans
<input type="checkbox"/>		c - http://www.springframework.org/schema/c
<input type="checkbox"/>		cache - http://www.springframework.org/schema/cache
<input checked="" type="checkbox"/>		context - http://www.springframework.org/schema/context
<input checked="" type="checkbox"/>		jdbc - http://www.springframework.org/schema/jdbc
<input type="checkbox"/>		jee - http://www.springframework.org/schema/jee
<input type="checkbox"/>		lang - http://www.springframework.org/schema/lang
<input type="checkbox"/>		mvc - http://www.springframework.org/schema/mvc
<input checked="" type="checkbox"/>		mybatis-spring - http://mybatis.org/schema/mybatis-spring
<input type="checkbox"/>		p - http://www.springframework.org/schema/p
<input type="checkbox"/>		task - http://www.springframework.org/schema/task
<input type="checkbox"/>		tx - http://www.springframework.org/schema/tx
<input type="checkbox"/>		util - http://www.springframework.org/schema/util

[그림 2 - 32] root-context.xml의 네임스페이스 추가

Root-context.xml 에 aop, context, jdbc, mybatis-spring의 네임스페이스를 추가하고, mySQL과의 연결을 담당하는 DataSource를 설정 및 MyBatis 연결 부분을 작성한다.

```

<!-- ROOT CONTEXT: defines shared resources visible to all other web components -->
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
    <property name="url" value="jdbc:log4jdbc:mysql://127.0.0.1:3306/project"></property>

    <property name="username" value="root" />
    <property name="password" value="248624" />
</bean>
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:/mybatis-config.xml"></property>
    <property name="mapperLocations" value="classpath:mappers/**/*.xml"></property>
</bean>

<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
    destroy-method="clearCache">
    <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>
</bean>
<context:component-scan base-package="com.pr.persistence">
</context:component-scan>
<context:component-scan base-package="com.pr.service">
</context:component-scan>
</beans>

```

[그림 2 - 33] DataSource 설정, MyBatis 연결, XML Mapper 인식

```

<mapper namespace="com.pr.mapper.colorMapper">

    <select id="find_zero" resultType="com.pr.domain.ColorsVO">
        <if test="color0 == 'test'">
            SELECT * FROM test
        </if>
        <if test="color0 == 'gogh'">
            SELECT * FROM gogh
        </if>
    </select>

    <select id="find_one" resultType="com.pr.domain.ColorsVO">
        <if test="color0 == 'test'">
            SELECT * FROM test
        </if>
        <if test="color0 == 'gogh'">
            SELECT * FROM gogh
        </if>
        WHERE colors
        like CONCAT('%', #{color1}, '%')
    </select>

    <select id="find_two" resultType="com.pr.domain.ColorsVO">
        <if test="color0 == 'test'">
            SELECT * FROM test
        </if>
        <if test="color0 == 'gogh'">
            SELECT * FROM gogh
        </if>
        WHERE
        colors like CONCAT('%', #{color1}, '%') and
        colors like CONCAT('%', #{color2}, '%')
    </select>

```

[그림 2 - 34] colorMapper.xml 중 일부

위의 그림은 MySQL에 보낼 쿼리문을 myBatis의 매퍼로 작성한 일부분이다. Test 테이블과 gogh 테이블중에서 골라 정하고 검색 할 색의 숫자에 따라서 and를 붙여 CONCAT문을 계속 추가한다.

```

package com.pr.persistence;

import java.util.HashMap;

@Repository
public class ColorsDAOImpl implements ColorsDAO {

    @Inject
    private SqlSession sqlSession;

    private static final String namespace = "com.pr.mapper.colorMapper";

    @Override
    public List<ColorsVO> find_zero(String color0) throws Exception {
        Map<String, Object> paramMap = new HashMap<String, Object>();

        paramMap.put("color0", color0);

        return sqlSession.selectList("com.pr.mapper.colorMapper.find_zero", paramMap);
    }

    @Override
    public List<ColorsVO> find_one(String color0, String color1) throws Exception {
        Map<String, Object> paramMap = new HashMap<String, Object>();

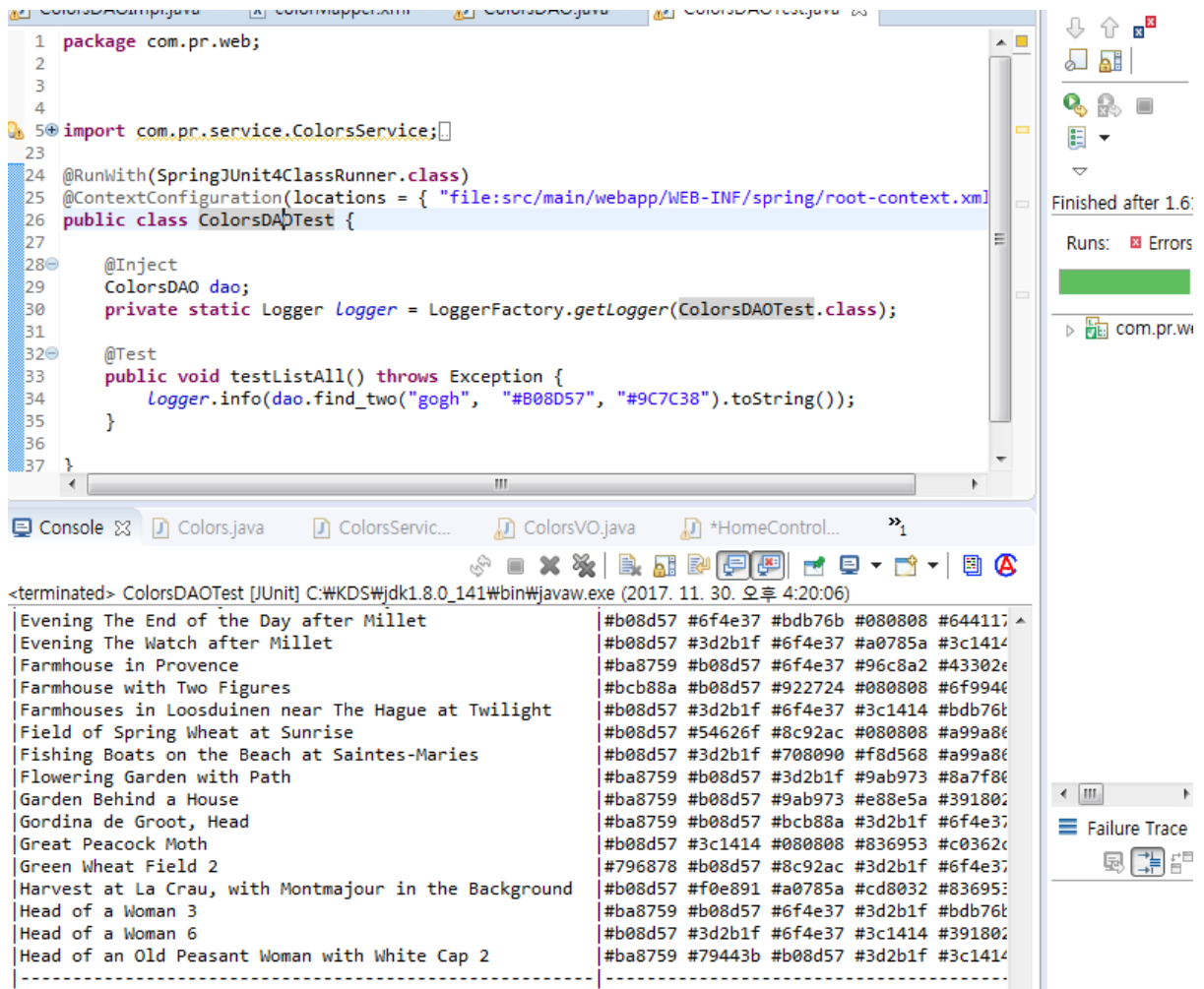
        paramMap.put("color0", color0);
        paramMap.put("color1", color1);

        return sqlSession.selectList("com.pr.mapper.colorMapper.find_one", paramMap);
    }
}

```

[그림 2 - 35] colorDAOImpl.java 중 일부

위는 Mapper 인터페이스를 구현한 클래스 colorDAOImpl 이다. 파라미터가 두 개 이상 전달되는 경우, Map 타입의 객체를 구성해서 파라미터로 사용한다.

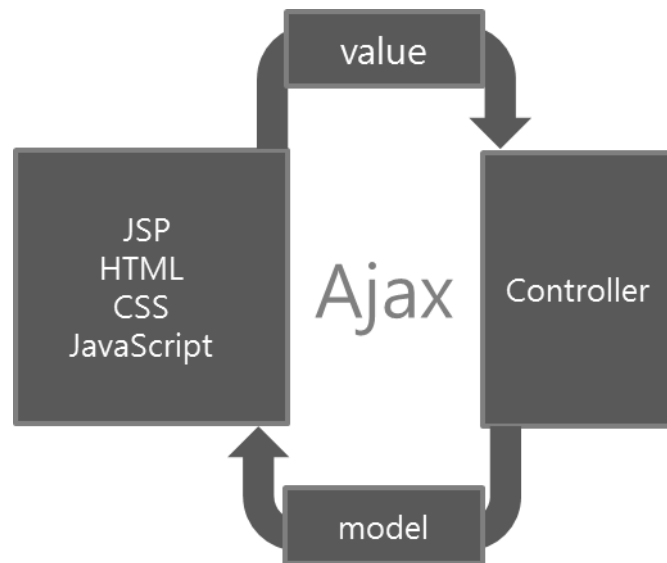


[그림 2 - 36] colorDAO 테스트 성공

현재까지 작업이 잘 연결되어 작동되고 있는지 확인한 결과이다. junit에 성공했다는 의미로 녹색이 떴고, 셀렉트 문의 결과가 콘솔에 잘 뜨고 있다.

여기까지 스프링과 MyBtis의 연동이 끝났다.

2.4.2 Ajax와 MyBatis간 연동



[그림 2 - 37] 프론트 페이지와 스프링 컨트롤러간 정보교환 흐름

이 프로젝트의 디자인은 다른 페이지로 넘어가지 않고 계속해서 한 페이지에 동작되는 것을 목표로 했기 때문에 스프링 컨트롤러에 건네고 받을 데이터를 Ajax를 사용하여 통신하였다.

클라이언트 사이드에서 Ajax로 보낼 데이터를 JSON 형식으로 파싱하여 보내고 스프링 컨트롤러에서는 JSON을 받기 위해 만든 Color라는 객체로 Ajax로 받은 데이터를 만들어 서비스로 보낸다.

```

function colorAjax() {
    var Colors = {};
    var urls = "";
    for (var i = 0; i < arguments.length; i++) {
        eval("Colors.color" + i + "=arguments[" + i + "]");
    }
    switch (arguments.length) {
    case 1:
        urls = "get0.do";
        break;

    case 2:
        urls = "get1.do";
        break;

    case 3:
        urls = "get2.do";
        break;

    case 4:
        urls = "get3.do";
        break;
    }
}

```

[그림 2 - 38] home.jsp 안의 colorAjax()의 일부분

Home.jsp 안에서 Ajax작업을 실행시키는 자바스크립트 함수 colorAjax는 자신을 실행시킬 때 들어 있던 인수의 수에 따라서 오브젝트 형식의 변수 Colors 안에 함수의 인수의 정보를 집어넣는 구절이 있다. 인수의 수에 따라서 홈 컨트롤러에 연결 할 url 또한 다르게 설정한다.


```

        case 10:
            urls = "get9.do";
            break;
    }

    var a;
    $.ajax({
        method : "POST",
        url : urls,
        data : JSON.stringify(Colors),
        async : false,
        type : "json",
        contentType : "application/json",
        success : function(data, status, xhr) {
            a = data;
            return a
        },
        error : function(jqXHR, textStatus, errorThrown) {
            console.log(jqXHR.responseText);
            console.log("Ajax false")
        }
    });
    return a
}

```

[그림 2 - 39] home.jsp 안의 colorAjax()의 일부분2

인수들이 저장된 Colors 변수를 JSON형식으로 컨트롤러에 담아 보내고, 데이터를 응답 받았다면 변수에 데이터를 담아 리턴 시키는 형식이다.

```

@RequestMapping(value = "/get0.do", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody Map<String, Object> testGet0(Locale locale, Model model, ColorsVO vo, @RequestBody Colors colors) throws Exception {
    return (Map<String, Object>) service.find0(vo, colors.color0);
}

@RequestMapping(value = "/get1.do", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody Map<String, Object> testGet1(Locale locale, Model model, ColorsVO vo, @RequestBody Colors colors) throws Exception {
    return (Map<String, Object>) service.find1(vo, colors.color0, colors.color1);
}

@RequestMapping(value = "/get2.do", method = {RequestMethod.POST, RequestMethod.GET})
public @ResponseBody Map<String, Object> testGet2(Locale locale, Model model, ColorsVO vo, @RequestBody Colors colors) throws Exception {
    return (Map<String, Object>) service.find2(vo, colors.color0, colors.color1, colors.color2);
}

```

[그림 2 - 40] home.jsp 안의 colorAjax()의 일부분2

컨트롤러에서는 Ajax에서 보낸 데이터를 @RequestBody로 Colors 객체로 받아내고, 그 인수를 적절한 서비스 객체를 찾아 호출해 보낸다

```

@Override
public Map<String, Object> find0(ColorsVO vo, String string0) throws Exception {
    Map<String, Object> map = new HashMap<String, Object>();

    map.put("list", dao.find_zero(string0));

    return map;
}

@Override
public Map<String, Object> find1(ColorsVO vo, String string0, String string1) throws Exception {
    Map<String, Object> map = new HashMap<String, Object>();

    map.put("list", dao.find_one(string0, string1));

    return map;
}

@Override
public Map<String, Object> find2(ColorsVO vo, String string0, String string1, String string2) throws Exception {
    Map<String, Object> map = new HashMap<String, Object>();

    map.put("list", dao.find_two(string0, string1, string2));

    return map;
}

```

[그림 2 - 41] ColorsServiceImpl.java의 일부분2

스프링의 service 부분에서는 검색 할 색의 수에 맞는, 매퍼와 연결했던 적절한 DAO 메소드들과 연결한다.

위의 결과를 웹페이지 html안의 textarea 안으로 보낸 것을 구글 크롬 웹개발자 툴로 확인해 보았다.

```

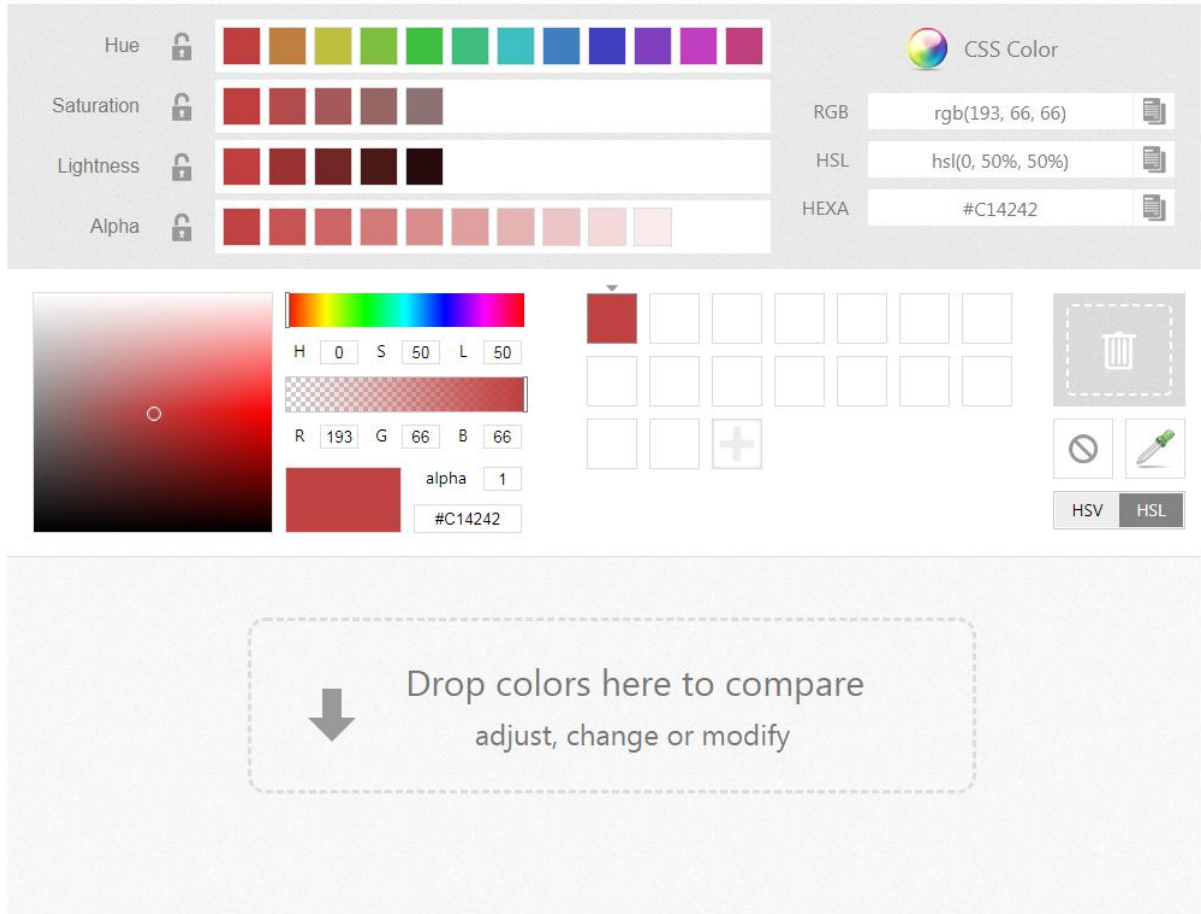
▼<textarea class="input" id="source" style="visibility:hidden">
    "#d8e4bc #8a9a5b #543d37 #bdb76b #826644"
    "#b5a642 #004040 #d8e4bc #6e6e30 #2f847c"
    "#85754e #333333 #d8e4bc #acbf60 #635147"
    "#965a3e #d8e4bc #664228 #ff9966 #c39953"
    "#acbf60 #d8e4bc #6f9940 #a9ba9d #556b2f"
    "#8a9a5b #4a5d23 #3b331c #4f7942 #d8e4bc"
    "#483c32 #d8e4bc #100c08 #757575 #b5a642"
    "#f8f8f8 #bcb88a #444c38 #d8e4bc #78866b"
</textarea>

```

[그림 2 - 42] html에서 데이터 확인 부분

2.5 클라이언트 사이드 개발

2.5.1 아웃풋 기능 추가

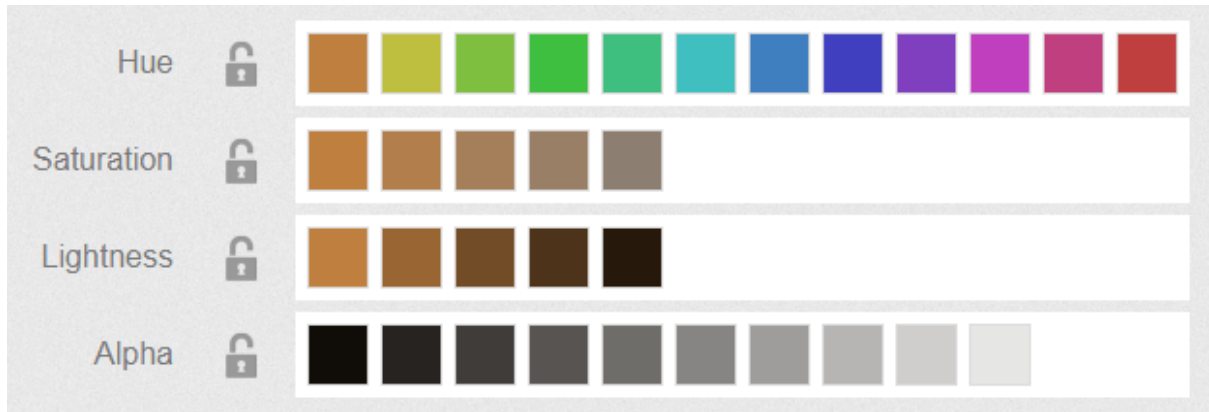


[그림 2 - 43] MDN Color picker tool

MDN에서 제공되는 웹 기준의 사용자 정의 색상을 쉽게 만들고 조정하고 실험 할 수 있는 툴이다. 이 프로젝트의 클라이언트 사이드의 작업 대부분은 이 툴을 개조하여 작성하였다.

Color picker tool의 js파일은 2000줄이 넘는데, 그 중에서 여러 클래스와 함수들을 분석하고 다시 재해석하여 기능을 추가하여 넣는 작업들을 했다.

다음 이미지는 셀렉트 된 데이터들이 워드카운터 된 데이터를 이용하여 나오는 아웃풋을 만들기 위해 원래 color picker tool에 포함된 부분이다.



[그림 2 - 44] CPT 에 포함되어 아웃풋을 만들기 위해 이용될 부분

그리고 빠른 작업을 위해 워드카운트 로직을 직접 만들지 않고 fengyuanchen이라는 github 유저가 만든 WordCounter(<https://github.com/fengyuanchen/wordcounter>)를 사용하였다.

아래의 이미지는 불러낸 데이터에서 워드카운트를 실시한 String 데이터이다.

```
1> 100c08: 329
2> b08d57: 327
3> 2c1608: 308
4> 9c7c38: 288
5> 918151: 278
6> 3d2b1f: 250
```

[그림 2 - 45] 워드카운터 된 데이터들 중 일부

```

var createHuePalette = function createHuePalette() {
    var palette = new Palette('Hue', 12);

    UIColorPicker.subscribe('picker', function(color) {
        if (palette.locked === true)
            return;

        for (var i = 0; i < 12; i++) {
            palette.samples[i].updateHue(color, 30, i);
        }
    });

    color_palette.appendChild(palette.container);
};

var createSaturationPalette = function createSaturationPalette() {
    var palette = new Palette('Saturation', 11);

    UIColorPicker.subscribe('picker', function(color) {
        if (palette.locked === true)
            return;

        for (var i = 0; i < 11; i++) {
            palette.samples[i].updateSaturation(color, -10, i);
        }
    });

    color_palette.appendChild(palette.container);
};

```

[그림 2 - 46] 아웃풋 부분을 만들기 위해 참고할 부분

위의 그림은 [그림 2-37]의 이미지를 형성하고 있는 코드 중 일부분이다. 이 부분을 참고해서 자신만의 createThemaPalette를 만들었다.

```

var createThemaPalette = function createThemaPalette() {
    var arr_c = [];
    var arr_c_f = wordcount().length;
    var palette_t = new Palette_t('Thema', arr_c_f);

    UIColorPicker.subscribe('picker',
        function() {
            arr_c = wordcount();
            console.log("arr_c[i].length : " + arr_c.length);
            for (var i = 0; i < arr_c_f; i++) {
                palette_t.samples[i].updateThema(arr_c[i], i,
                    arr_c.length);
            }
        });
    color_palette_t.appendChild(palette_t.container);
};

```

[그림 2 - 47] createThemaPalette

```

var ajaxToPalette = function ajaxToPalette(hex) {
    var hexToRgb = function hexToRgb(hex) {
        var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i
            .exec(hex);
        return result ? {
            r : parseInt(result[1], 16),
            g : parseInt(result[2], 16),
            b : parseInt(result[3], 16)
        } : null;
    }
    var color = new Color();
    color.setRGBA(hexToRgb(hex).r, hexToRgb(hex).g, hexToRgb(hex).b, 1);
    return color;
}

var wordcount = function wordcount() {
    var arr_c = [];
    wordcounter.count($source.val(), function(result, logs) {
        for (i = 0; i < logs.split('\n').length - 1; i++) {
            arr_c[i] = logs.split('\n')[i].split('>')[1].split(' ')[1]
                .split(':')[0];
            arr_c[i] = ajaxToPalette(arr_c[i]);
        }
    });
    console.log(arr_c);
    return arr_c;
}

```

[그림 2 - 48] 42createThemaPalette에 쓰인 함수 두 개

createThemaPalette 함수가 원래 쓰여있던 함수와 다른 점은 각자의 update 함수에 넣을 color 라는 데이터를 wordcount()안에서 Ajax로 미리 데이터를 저장시켜 놓았던 textarea안의 데이터들을 워드카운트 해서 그 색의 순서대로 나열해 arr_c라는 배열 안에 저장한 것으로 바꿔치기 했다는 것과,

원래의 지정된 div가 아닌 따로 만들어 둔 div를 지정해 그곳에 appendChild()를 사용하여 그곳에 samples의 노드들이 추가 될 수 있도록 한 것이다.

참고로 samples의 색상데이터를 입력하기 위해선 color picker tool의 Color()객체로 입력해야 색의 정보를 쉽고 온전하게 저장 할 수 있다.

```
function Color(color) {  
    if (color instanceof Color === true) {  
        this.copy(color);  
        return;  
    }  
  
    this.r = 0;  
    this.g = 0;  
    this.b = 0;  
    this.a = 1;  
    this.hue = 0;  
    this.saturation = 0;  
    this.value = 0;  
    this.lightness = 0;  
    this.format = 'HSV';  
}
```

[그림 2 - 49] Color 객체 형식

```

ColorSample.prototype.updateBrightness = function updateBrightness(
    color, value, steps) {
    var brightness = color.value + value * steps;
    if (brightness <= 0) {
        this.node.setAttribute('data-hidden', 'true');
        return;
    }
    this.node.removeAttribute('data-hidden');
    this.color.copy(color);
    this.color.setValue(brightness);
    this.updateBgColor();
};

ColorSample.prototype.updateAlpha = function updateAlpha(color, value,
    steps) {
    var alpha = parseFloat(color.a) + value * steps;
    if (alpha <= 0) {
        this.node.setAttribute('data-hidden', 'true');
        return;
    }
    this.node.removeAttribute('data-hidden');
    this.color.copy(color);
    this.color.a = parseFloat(alpha.toFixed(2));
    this.updateBgColor();
};

```

[그림 2 - 50] 각자의 함수에서 쓰인 update 함수들

```

ColorSample.prototype.updateThema = function updateThema(color, i,
    length) {
    if (i >= length) {
        this.node.setAttribute('data-hidden', 'true');
        return;
    }

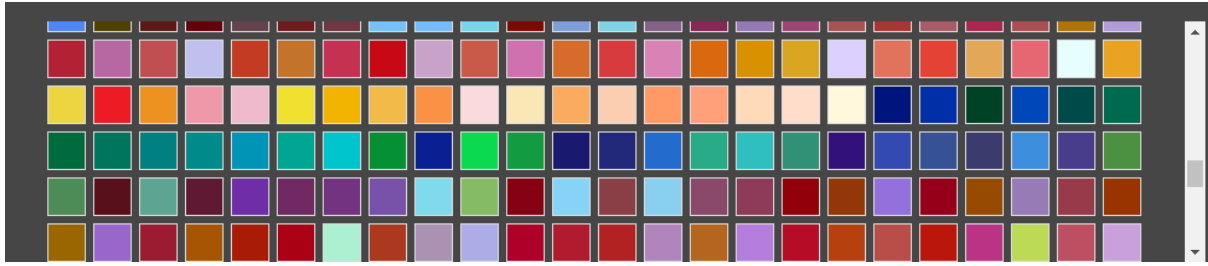
    this.node.removeAttribute('data-hidden');
    this.color.copy(color);
    this.updateBgColor();
};

```

[그림 2 - 51] createThemaPalette에 쓰인 updateThema 함수

updateThema 함수에서 2번째 이후의 검색에서 원래 있던 샘플의 수들이 나중에 검색된 샘플의 수보다 많을 때 검색된 샘플들을 남겨두고 나머지 샘플들을 보이지 않게 하는 작업이 다른 update 함수랑 비교했을 때의 차이점이다.

아래의 이미지는 이렇게 작성된 코드들로 완성된 Ajax로 받은 데이터를 워드카운드해서 나열한 아웃풋 표현을 MDN Color picker tool에서 추가한 이미지다.



[그림 2 - 52] Ajax로 받은 데이터로 표현된 아웃풋 결과

2.5.2 인풋 기능 추가



[그림 2 - 53] 47CPT 에 포함되어 인풋을 만들기 위해 이용될 부분

위의 그림은 MDN picker tool에서 색을 저장하고 모아두는 툴의 부분이다. + 를 선택하여 샘플의 노드를 추가 할 수도 있다. 이 부분을 이용해서 색을 선택하는 인풋 부분을 만들려 한다.

위 이미지의 기능을 작성한 코드는 `ColorPickerSamples()` 라는 함수이다. 이 함수를 통째로 하나 더 추가하고 함수 이름을 `ColorPickerSamples_t()`라고 작성 한 뒤, append 될 엘리먼트를 사정에 맞게 수정하고 쓸데없는 부분은 삭제했고 처음부터 add 기능이 있는 샘플을 제외하고 모든 샘플은 처음부터 없는 상태로 만들었다. .

아래는 새로 만들어진 `ColorPickerSamples_t()`만의 새로운 기능을 추가한 코드 부분이다.

```

ColorSample.prototype.printResult = function printResult() {
    $("#source").empty();
    var r = [];
    var g = [];
    var b = [];
    var obj = [];
    var nameConverters = [];
    var ajaxValue = [];
    var result;

    console.log(samples.length);
    console.log(nr_samples);

    for (a = 0; a < samples.length; a++) {
        if (samples[a] !== null) {

            r[a] = samples[a]["color"]["r"];
            g[a] = samples[a]["color"]["g"];
            b[a] = samples[a]["color"]["b"];
            nameConverters.push(this.nameConverter(r[a], g[a], b[a]));
        } else {
            nameConverters.push({});
        }
    }
    for (b = 0; b < samples.length; b++) {
        if (samples[b] !== null) {
            var rgbToHex = ("#" + ((1 << 24)
                + (nameConverters[b]["R"] << 16)
                + (nameConverters[b]["G"] << 8) + nameConverters[b]["B"])
                .toString(16).slice(1));
            console.log(rgbToHex);
            console.log(b);
            ajaxValue.push(rgbToHex);
        }
    }

    switch (nr_samples) {
        case 0:
            console.log(ajaxValue);
            result = colorAjax($thema.val());
            break;

        case 1:
            console.log(ajaxValue);
            result = colorAjax($thema.val(), ajaxValue[0]);
            break;

        case 2:
            console.log(ajaxValue);
            result = colorAjax($thema.val(), ajaxValue[0],
                ajaxValue[1]);
            break;
    }
}

```

[그림 2 - 54] ColorPickerSamples_t()안의 PrintResult()함수

this.nameConverter는 ColorPickerSamples_t에서 만들어 둔 공식적인 색 목록의 색들로 변환시키는 함수이다.

nameConverters라는 배열 안에 사용자가 색을 정해 검색하려는 색의 데이터를 가지고 있는 샘플들의 R, G, B의 데이터를 저장하고 그 저장된 값에서 공식적인 색 목록에서 가까운 색으로 변환시킨다. 만약에 샘플들 중에서 삭제된 샘플은 null값으로 판정이 되는데, 그때는 빈 오브젝트 값을 넣어준다.

이후 샘플이 삭제된 샘플이 아닐 경우 그 샘플에 해당하는 nameConverters 라는 배열의 값을 복사 코드로 바꾸어 새로운 배열 ajaxValue의 안에 집어 넣고, 그 배열로 nr_sample(현재 검색하려고 활성화 시킨 샘플의 수)의 수에 따라 적절하게 colorAjax에 실행시킬 인수들의 수를 정하고 넣는다.

```
case 9:
    console.log/ajaxValue);
    result = colorAjax($thema.val(), ajaxValue[0],
        ajaxValue[1], ajaxValue[2], ajaxValue[3],
        ajaxValue[4], ajaxValue[5], ajaxValue[6],
        ajaxValue[7], ajaxValue[8]);
    break;
}

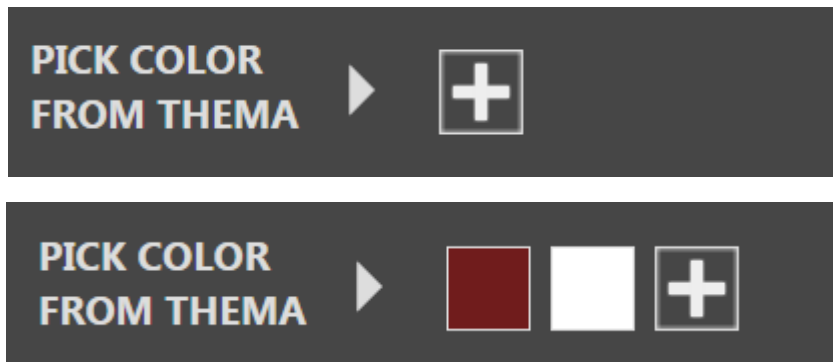
}
}
if (result !== undefined) {
    for (i = 0; i < result.list.length; i++) {
        $("#source").append(result.list[i].colors);
    }
    console.log(result);
    console.log("this is colorAjax : " + $thema.val());
    console.log(typeof ($thema.val()));
}
}
```

[그림 2 - 55] PrintResult()의 마지막 부분

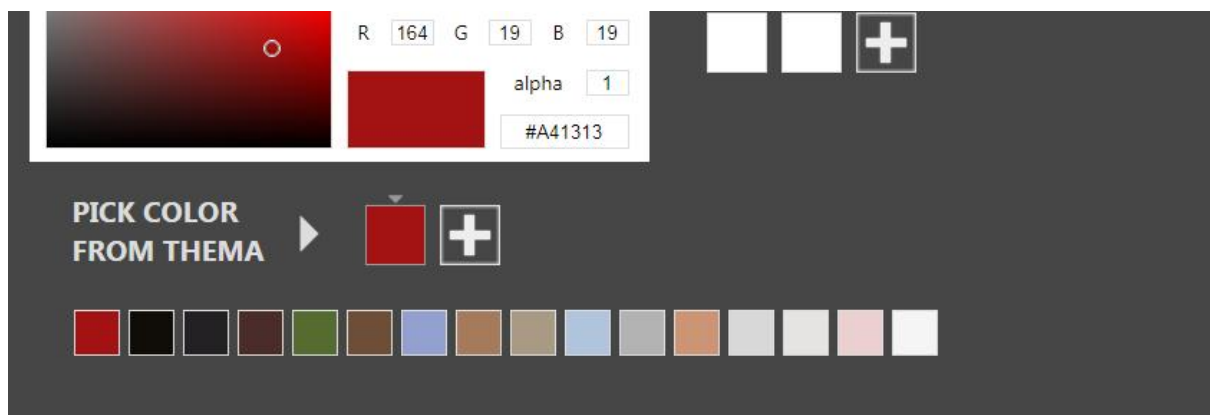
그리고 함수에 앞서 비웠던 id가 source인 textarea 안을 다시금 검색한 데이터들로 채운다

이렇게 완성된 PrintResult()를 샘플이 컬러를 업데이트 하거나, 활성화 될 때 작동시키는 함수 안에서 같이 작동시킬 수 있도록 작동시켰다.

아래는 위의 코드로 완성된 실제 html상의 모습이다.



[그림 2 - 56] 완성된 Ajax로 보낼 데이터를 정하는 인풋 부분



[그림 2 - 57] test DB에서 #A41313를 검색한 결과



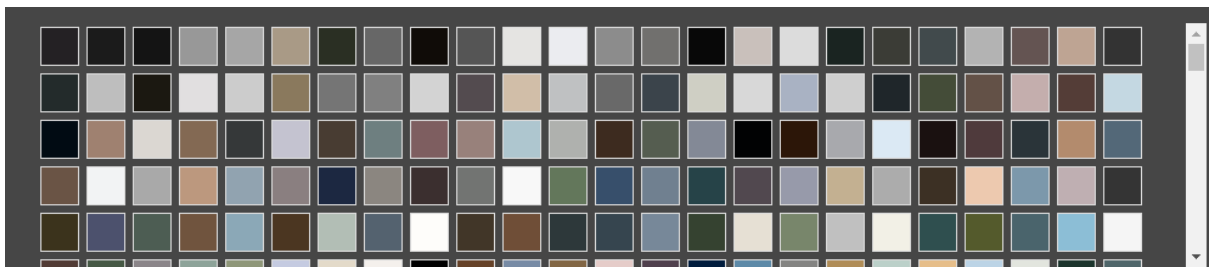
[그림 2 - 58] #A41313와 #556B2F 검색 결과

3 결론

3.1 추후 과제



[그림 3 - 1] 0개 선택시 고희 테마 아웃풋

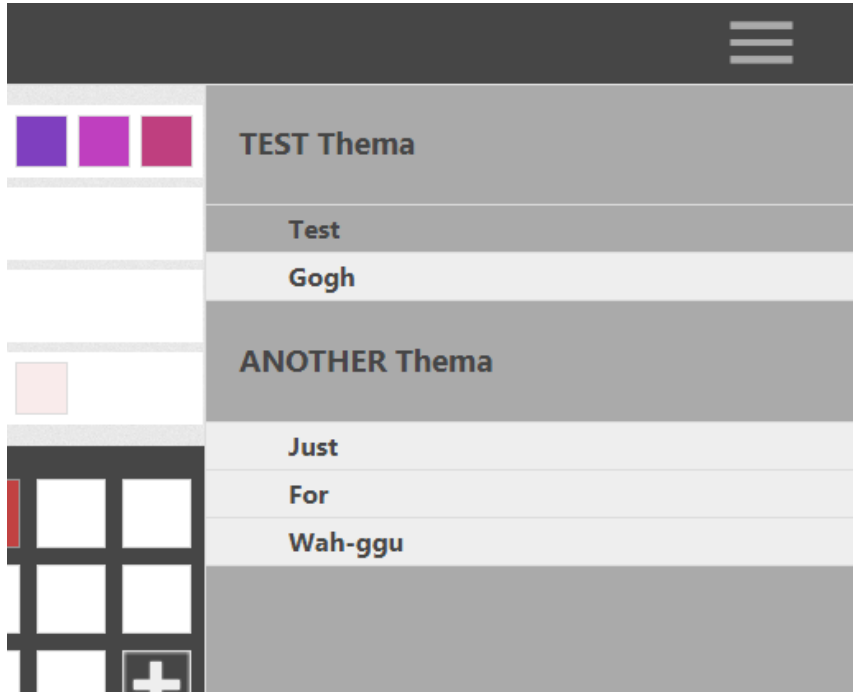


[그림 3 - 2] 0개 선택 시 test 테마 아웃풋

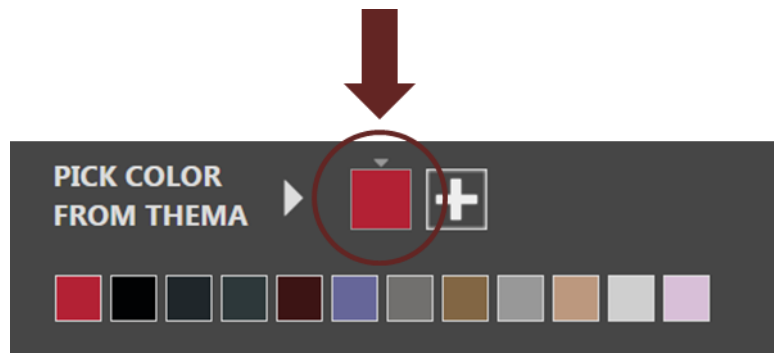
문제점 1

처음 홈페이지를 들어 왔을 때, 처음에는 0개가 선택되어 테이블 안의 모든 데이터들을 워드카운트 하여 아웃풋에 뿌려주게 되는데, 제일 많이 세어진 색 일수록 어둡고 탁한 색들이 많았다. 색 선택을 도와주는 서비스는 그만큼 다채롭고 화려한 색상을 먼저 보여야 하는 고정적인 이미지가 있는데, 그걸 배신하고 있어 그만큼 신뢰성이 떨어지게 될 수 있다.

색을 선택하지 않는 이상은 데이터가 보이지 않거나 다른 방법을 강구 해야 할 것이다.



[그림 3 - 3] 테마를 선택하는 부분



[그림 3 - 4] 눌러야 색 검색이 동작이 되는 부분

문제점 2

선택지들의 업데이트는 반드시 [그림 3 - 4]의 붉은 표시 부분을 눌러야 동작하는데, 그 밖에 있는 테마를 고르는 부분에선 동작 할 수 있는 방법을 찾지 못했다.



[그림 3 - 5] 대중적이지 않은 색감의 고흐 그림들

고흐의 그림들 중에서는 호 불호가 심하게 갈리는 색감의 그림들이 많았고, 사진이 아닌 일러스트는 최적의 클러스터에서 10을 곱한 수의 클러스터를 뽑아내서 색 조합을 데이터베이스에 저장했는데 이 두 개의 이유 때문인지 고흐의 테마로 실험해본 프로젝트에서는 좋은 색을 뽑는 서비스를 실행에 별로 도움을 받는 느낌을 받지 못했다.

3.2 발전 방향

현재는 테스트를 위해 색감이 좋은 무작위 사진들과 고흐의 그림들, 이 둘로 테마를 구성했지만 이 프로젝트는 테마를 특정한 목적으로 구현한다면 매우 강력한 작품이 될 수 있을 것이라고 생각한다. 예를들어, 호평을 받은 패션 디자인의 옷들로 테마를 만들고 서비스한다면, 자신이 원하는 색부터 시작해서 호평의 디자인의 색 조합에 가까운 만족스러운 색 조합을 찾는 것에 도움을 줄 수 있을 것이고, 특정 유저가 좋아하는 스타일의 화가로 테마를 만들어 서비스 한다면, 그 화가의 화풍의 색 조합들을 찾아내서 이용 할 수 있을 것이다.

이처럼 패션이나 미술, 디자인 등에서 테마에 따라 가능성이 많을 것이라고 기대한다.