



A u t o c o l o r r e c o m m e n d e r

Color Bound

김 다 сом

1

개요

서비스 벤치마킹
주제 선정 동기
프로젝트 목표

2

내용

개발 환경
크롤링 개발
서버 개발
클라이언트 개발

3

결론

추후 과제
발전 방향

1

개요

서비스 벤치마킹
주제 선정 동기
프로젝트 목표

1. 서비스 벤치마킹

Adobe Color CC

Adobe에서 만든
컬러 선택 및 공유 도구

컬러 선택 서비스의 1인자

<https://color.adobe.com>



1. 서비스 벤치마킹

이미지에서 색 조합을
뽑아 내는 서비스

Adobe Color CC

어둡게
사용자 정의



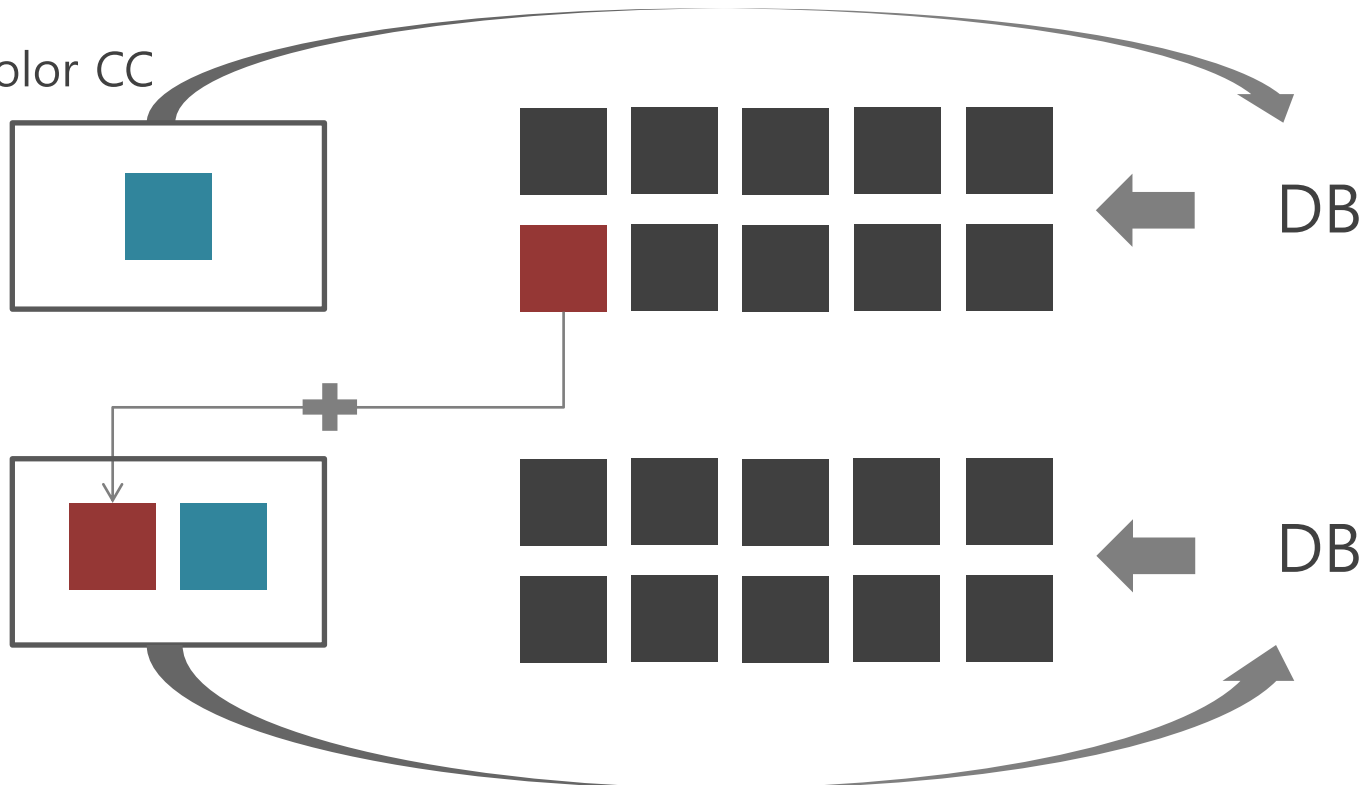
1. 주제 선정 동기

Adobe Color CC 서비스를 보고 프로젝트 주제 구상 중

이미지에서 뽑은 색 데이터들을 한 데 모은 뒤
그 중에서 자신이 원하는 색에 대한 연관성이 높은 데이터를
뽑을 수 있지 않을까?

그 연관성이 높은 데이터들 중에서 하나를 더 붙여 검색한 뒤
또 그 두 개의 색과, 그 세 개의 색과 연관성이 높은 데이터를
뽑아 낼 수 있지 않을까?

From
Adobe Color CC



1. 주제 선정 동기

From
Adobe Color CC

이미 여러 사람들에게 객관적으로 색감이 좋다는 호평을 받은 사진이나 그림에서 색감을 뽑아내어

그 데이터들을 이용해서 자신이 원하는 색들로
연관성이 높은 색들을 검색 할 수 있다면

번득이는 아이디어나 센스가 없어도
색 배합을 괜찮게 뽑아 낼 수 있지 않을까?



1. 프로젝트 목표

고객층

색감에 전문적이지 않은 일반인들
색을 선택하는 것에 도움이 필요한 전문가 또는 일반인

결과물

테마로 구별한 색 데이터 베이스에 따른
보편적인 색 선택 기회 제공

Java, python, javascript

작성 능력 확장

2

내용

개발 환경
크롤링 개발
서버 개발
클라이언트 개발

2. 개발 환경

Java

구글 크롬 웹 드라이버를 이용한
이미지 크롤링 프로그램 작성

Python(anaconda3, jupyter)

크롤링 된 이미지로 K-means를 사용하여
이미지의 대표적인 색조합들 추출

mySQL

데이터 베이스 구축

Spring

MyBatis를 이용한 데이터베이스와 클라이언트 간
통신 환경 구축

JavaScript

MDN Color Picker Tool 개조 및
Ajax로 MyBatis와 클라이언트 간 통신 환경 구축

2. 크롤링 개발

List of Colors









List of Colors

[https://en.wikipedia.org/wiki/List_of_colors_\(compact\)](https://en.wikipedia.org/wiki/List_of_colors_(compact))

공식적으로 지정된 색과 그 이름의 목록

컴퓨터에서 표현될 수 있는 색의 수는 255의 세제곱이다.

RGB 표현 값이 조금이라도 차이가 나더라도
검색이 될 수 있도록,
지정 된 색이 이 목록 안의 색에서 가장 가까운 색으로
변환 시키는 작업을 위해 필요하다.

	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%

2. 크롤링 개발

List of Colors

```
[
  {
    "R": 0,
    "B": 186,
    "G": 72,
    "name": "Absolute zero"
  },
  {
    "R": 76,
    "B": 39,
    "G": 47,
    "name": "Acajou"
  },
  {
    "R": 176,
    "B": 26,
    "G": 191,
    "name": "Acid green"
  },
  {
    "R": 124,
    "B": 232,
    "G": 185,
    "name": "Aero"
  },
  ...
]
```

List of Colors -> JAVA

import org.openqa.selenium.chrome.*
를 이용하여 구글 크롬 웹드라이버로 웹 데이터를 부르고,

import org.jsoup.*
를 이용하여 데이터중에서 필요한 부분을 선택해서,

import org.json.simple.parser.*
로 JSON 형식의 파일을 RGB값과 이름을 포함해 입력했다.

```
[
  {
    "R":0,
    "B":186,
    "G":72,
    "name":"Absolute zero"
  },
  {
    "R":76, ... etc
  },
  ...
]
```

2. 크롤링 개발

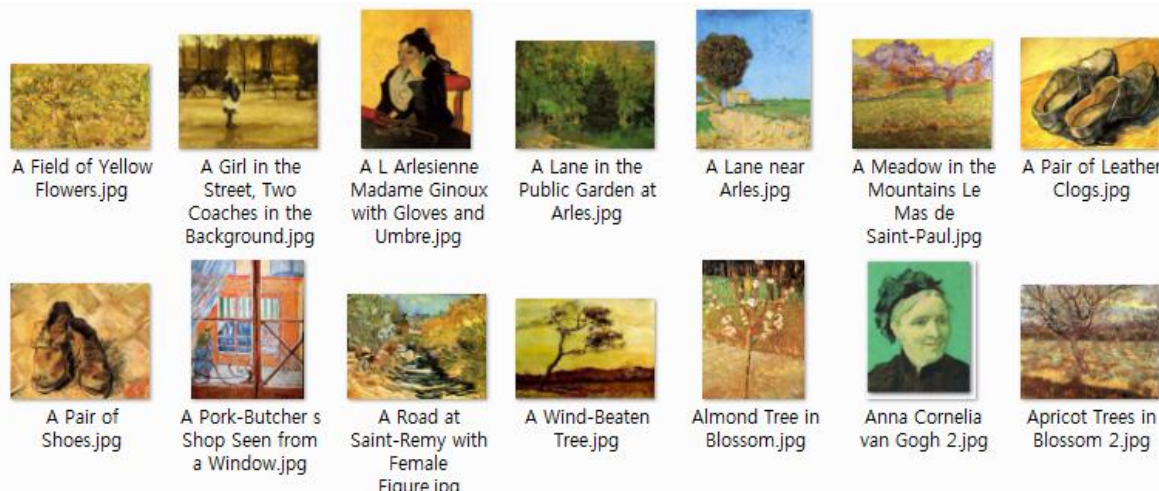
이미지들

이미지들

이미지에서 색 데이터를 추출하기 전에

우선 객관적으로 색감이 좋은 평가의 사진이 모여있는 StockSnap(<https://stocksnap.io/>)의 사진들과

사진이 아닌 일러스트도 실제 구현을 살펴보기 위해서 빈센트 반 고흐의 이미지들의 크롤링이 필요하다.



2. 크롤링 개발

이미지들

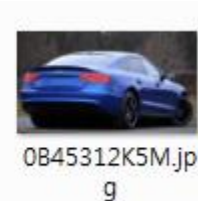
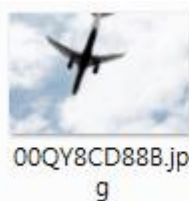
이미지들 -> JAVA

웹 데이터 불러오기와 선택은
List of Colors 크롤링과 같다.

```
import java.net.URL;  
import java.net.URLConnection;  
을 이용하여 url을 선택하고 연결하여 파일들을 다운받는다.
```

StockSnap에서는 로봇을 허가 하지 않는지
연결허가가 나지 않아서

User-Agent 프로퍼티를
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0)
Gecko/20100101 Firefox/31.0"으로 세팅해야
커넥션이 가능 했다.



2. 크롤링 개발

이미지의 색조합

이미지의 색조합

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

3차원으로 표현된 R, G, B의 포인트들을 K-means를 사용해 클러스터들을 나누어 그 클러스터의 평균값을 대표적인 색으로 취급하여 뽑아 낼 것이기 때문에

그래프 그리기에 용이하면서 K-means 모듈이 있는 Python(Anaconda3) Jupyter로 작업을 했다.

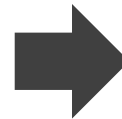


2. 크롤링 개발

이미지의 색조합

이미지의 색조합 -> Python

우선 그림을 수치화 시켜 클러스터를 나누기 위해
이미지를 읽어
한 픽셀당 입력 되어 있는 R, G, B 수치의 값을
배열로 만든다.



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

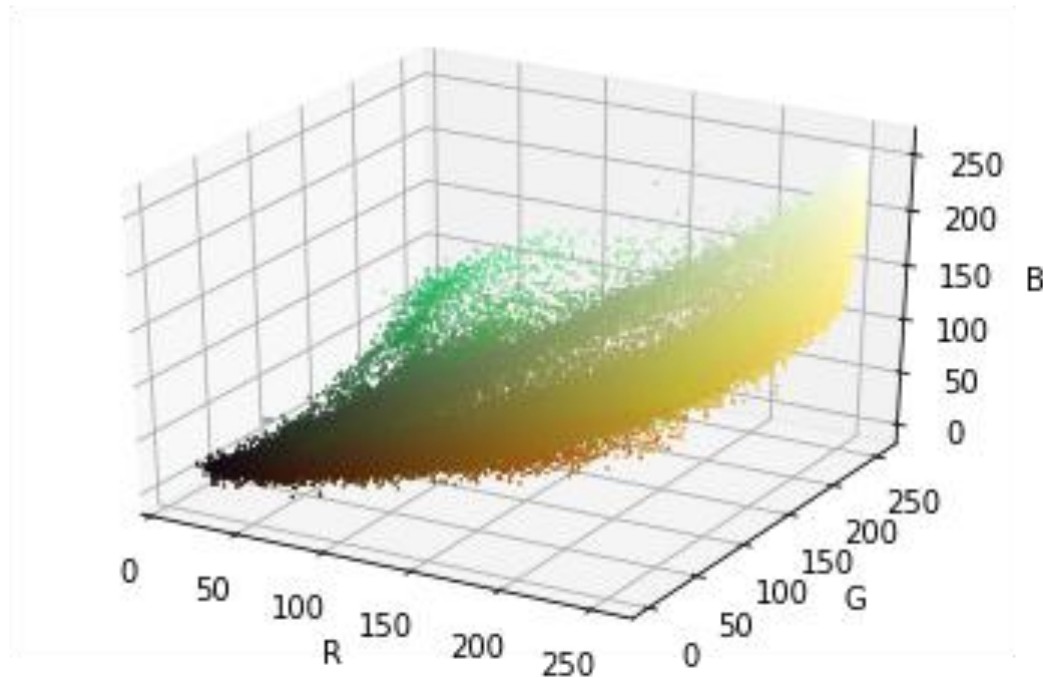
2. 크롤링 개발

이미지의 색조합

이미지의 색조합 -> Python

다음은 수치화된 배열을 R, G, B 포인트들을 뿌려
그려낸 3차원 스캐터 그래프이다.

그래프로 그리기에 충분한 이미지의 수치화가 완성되었다.



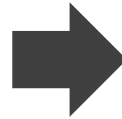
2. 크롤링 개발









이미지의 색조합

이미지의 색조합 -> Python

그전에 이미지에서 적게 쓰였지만
포인트가 될 수도 있는 색이 묻히는 현상을 막기 위해
수치화된 한 픽셀당 RGB값들을 공식적인 색 목록에 있는
색들 중 가장 가까운 색으로 변환 시킨 뒤
같은 색들은 중복 제거했다.

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



	American Bronze	#391802
	American Brown	#804040
	American Gold	#D3AF37
	American Green	#34B334
	American Orange	#FF8B00
	American Pink	#FF9899
	American Purple	#431C53
	American Red	#B32134

2. 크롤링 개발

이미지의 색조합 -> Python

제일 가까운 색을 찾아주는 함수는
3차원 거리 계산을 이용하여 작성하였다.

이미지의 색조합

$d =$

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

r = 65
g = 91
b = 91

Feldgrau
R : 65
G : 91
B : 91



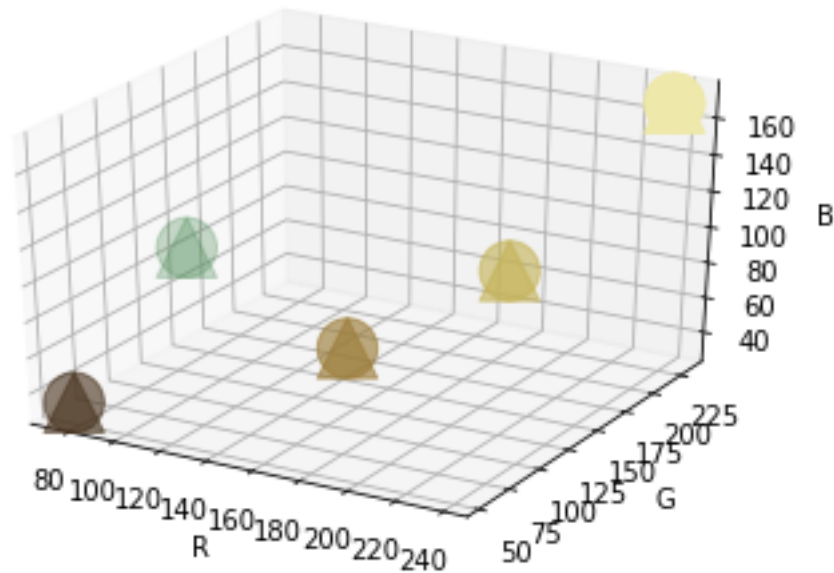
2. 크롤링 개발

이미지의 색조합

이미지의 색조합 -> Python

다음은 3차원 스캐터 그래프로
뽑아낸 k 클러스터들의 평균값들만 그려낸 그래프이다.

#4b3621, #9c7c39, #4d8c57, #c5b358, #eee8aa 이
결과값으로 나왔다.



2. 크롤링 개발

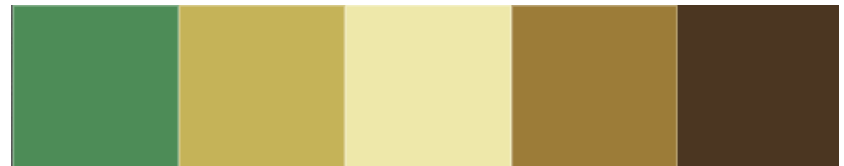
이미지의 색조합 -> Python

좌측의 이미지는 Adobe Color CC의 결과값이다.
하지만 항상 이렇게 결과값이 비슷하진 않다.

이미지의 색조합



#4b3621, #9c7c39, #4d8c57, #c5b358, #eee8aa



2. 크롤링 개발

이미지의 색조합 -> Python

좌측의 Adobe Color와 비교2

이미지의 색조합

Adobe Color CC



프로젝트 결과값



2. 크롤링 개발

이미지의 색조합 -> Python

좌측의 Adobe Color와 비교3

이미지의 색조합

Adobe Color CC



프로젝트 결과값



2. 크롤링 개발

이미지의 색조합 -> Python

좌측의 Adobe Color와 비교3

이미지의 색조합

Adobe Color CC



프로젝트 결과값



2. 크롤링 개발

이미지의 색조합

이미지의 색조합 -> Python

Adobe Color와 비교해보았을 때

장점

비교적 색상이 다양하게 나온다

클러스터 수를 사진에 맞도록 조정 할 수 있다.(elbow 함수 사용)

단점

채도가 공통적으로 약간 낮게 나온다.

Adobe Color의 무드 선택 기능 부재

Elbow 함수:최적의 클러스터 수를 찾아내는 함수

2. 크롤링 개발

이미지의 색조합

이미지의 색조합 -> Python

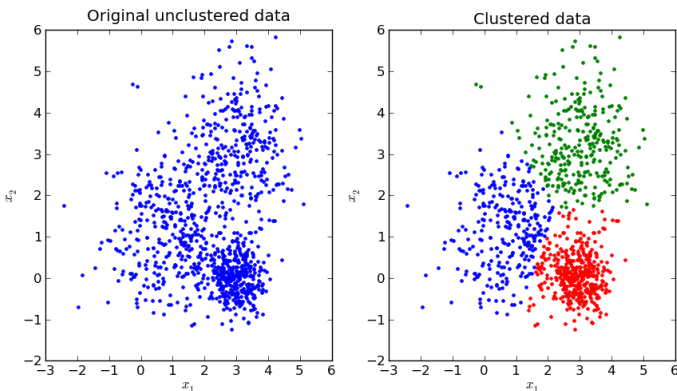
채도가 공통적으로 약간 낮게 나오는 문제
-> 클러스터들의 "평균값" 이 결과물이므로
클러스터 군집 안에서 채도가 높은 포인트보다는
좀더 낮은 채도의 포인트가 도출 된 것 아닐까?

-> 클러스터 군집 안에서 클러스터의 평균값과
색상이 제일 가까우며 R, G, B의 값 차이가 극명하게
나는 것(R, G, B의 수치가 가까울수록 채도가 낮다)
을 선택 해야 함

-> sklearn.cluster.KMeans로 아직 구현 가능한 방법
찾지 못함

Adobe Color의 무드 선택 기능 부재

-> 현재 작업 목적에서는 무드 선택이 불필요



2. 서버 개발

데이터 베이스

MySQL

이 프로젝트에서 검색해야 할 것은
유저가 검색한 색 하나와 연관되어있는 색들이다.

나중 작업에서 split 함수로 쪼개어 사용하기 위해
이전 과정에서 도출한 이미지의 색조합을
그림 하나당 하나의 문자열들로 만들어
데이터베이스에 들어갈 정보 형식을 정했다.

ex)

name : "파일이름 1"

colors : "#aaaaaa #bbbbbb #ccccc #ddddd #eeeeee"

Field	Type	Null	Key	Default	Extra
name	char(100)	NO	PRI	NULL	
colors	char(100)	NO		NULL	

2. 서버 개발

데이터 베이스

MySQL

다음은 실제로 데이터가 입력된 데이터베이스의 test 테이블이다.

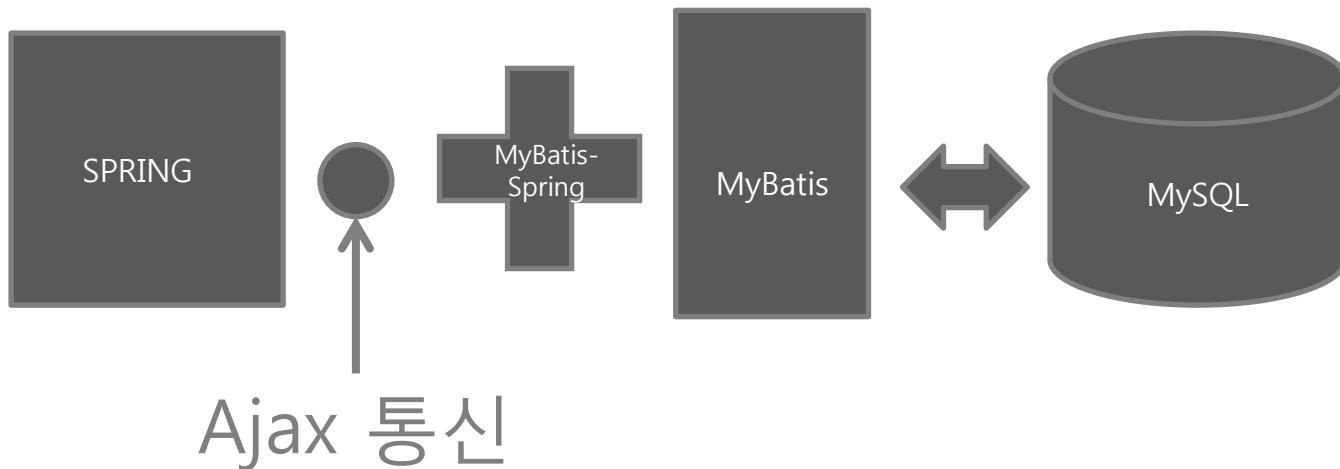
name	colors
004EAQQ9SC	#6e7f80 #523b35 #cd9575 #965a3e #141414
00Q7AI2NZJ	#f5f5f5 #3b3c36 #676767 #b3b3b3 #1b1b1b
00QY8CD88B	#f2f3f4 #080808 #aec6cf #555555 #c4d8e2
01IUV64AGL	#71706e #1b1b1b #ebecf0 #bea493 #543d37
01SOPQS4UN	#3b331c #c8ad7f #6e6e30 #92a1cf #918151
02GZUTSQ57	#2887c8 #00468c #355e3b #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
03P9JTLVF7	#c9c0bb #664228 #bea493 #a67b5b #43302e

3. 서버 개발

MyBatis

MyBatis

아래는 현재 프로젝트의 구동 구조의 목표이다.
클라이언트와 서버의 원활한 통신을 위해
Spring과 MyBatis의 연동 작업을 통해
SQL처리를 목표로 한다.

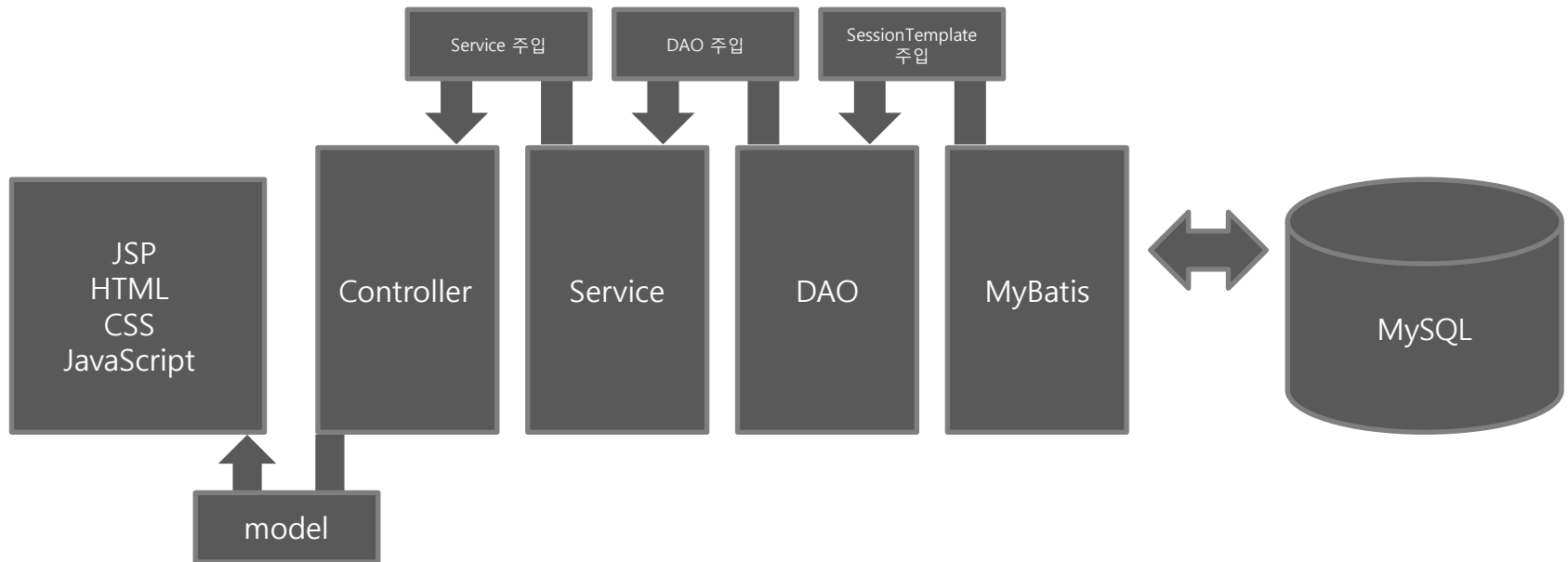


3. 서버 개발

MyBatis

MyBatis

첫번째로 Spring과 MyBatis를 연동시킨다.



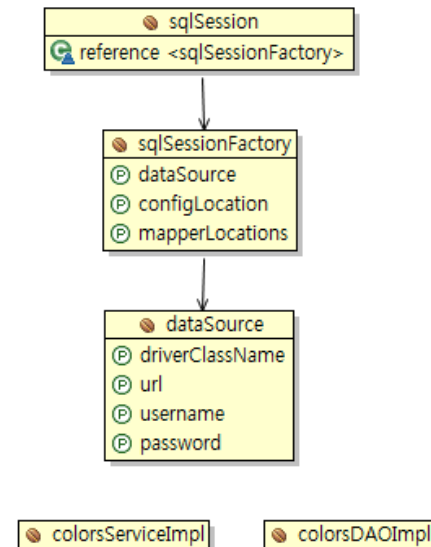
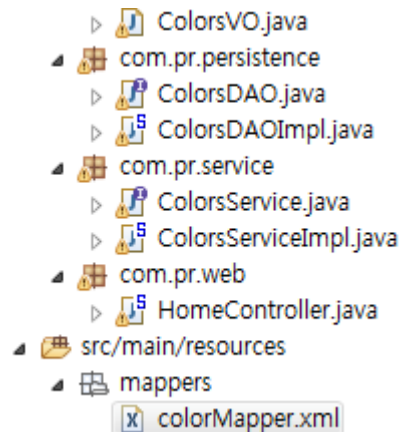
3. 서버 개발

MyBatis

MyBatis

좌측의 이미지를 확인하면
mapper는 colorMapper.xml,
DAO는 ColorsDAO.java, ColorsDAOImpl.java,
Service는 ColorsService.java, ColorsServiceImpl.java
Controller는 프로젝트 생성 때부터 있던
HomeController.java로 작성했다.

우측의 이미지는 root-context.xml 파일에서
bean graph로 myBatis 연동이 확인된 모습이다.



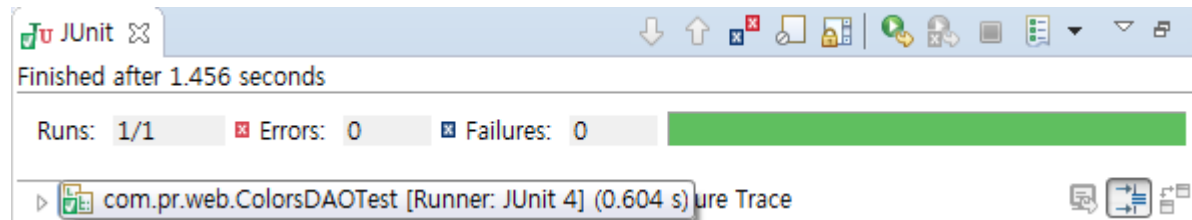
3. 서버 개발

MyBatis

색 두개를 검색하는 내용의 DAO를 사용한 테스트이다.
jUnit 테스트에서 성공적으로 작동하고,
로그에서 결과물들이 아웃풋 되는 것을 확인 할 수 있었다.

MyBatis

```
@Test
public void testListAll() throws Exception {
    logger.info(dao.find_two("gogh", "#B08D57", "#9C7C38").toString());
}
```



```
|Marshy Landscape
|Meadow in the Garden of Saint-Paul Hospital
|Mother Roulin with Her Baby
|Mother Roulin with Her Baby 2
```

```
|#5d1916 #9c7c38 #a0785a #674403 #88
|#9c7c38 #545a2c #c19a6b #cd8032 #86
|#9c7c38 #545a2c #e9d66b #867e36 #4b
|#9c7c38 #545a2c #3d2b1f #c19a6b #e9
```

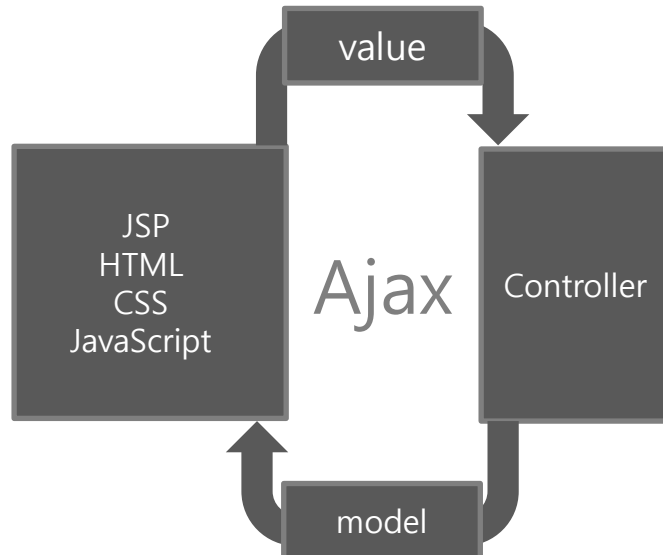

3. 서버 개발

Ajax<->HomeController

Ajax

이 프로젝트의 디자인은 다른 페이지로 넘어가지 않고
계속해서 한 페이지에서 동작되는 것을 목표로 했기 때문에

HomeController에 건네고 받을 데이터를
Ajax를 사용하여 통신하였다.



3.

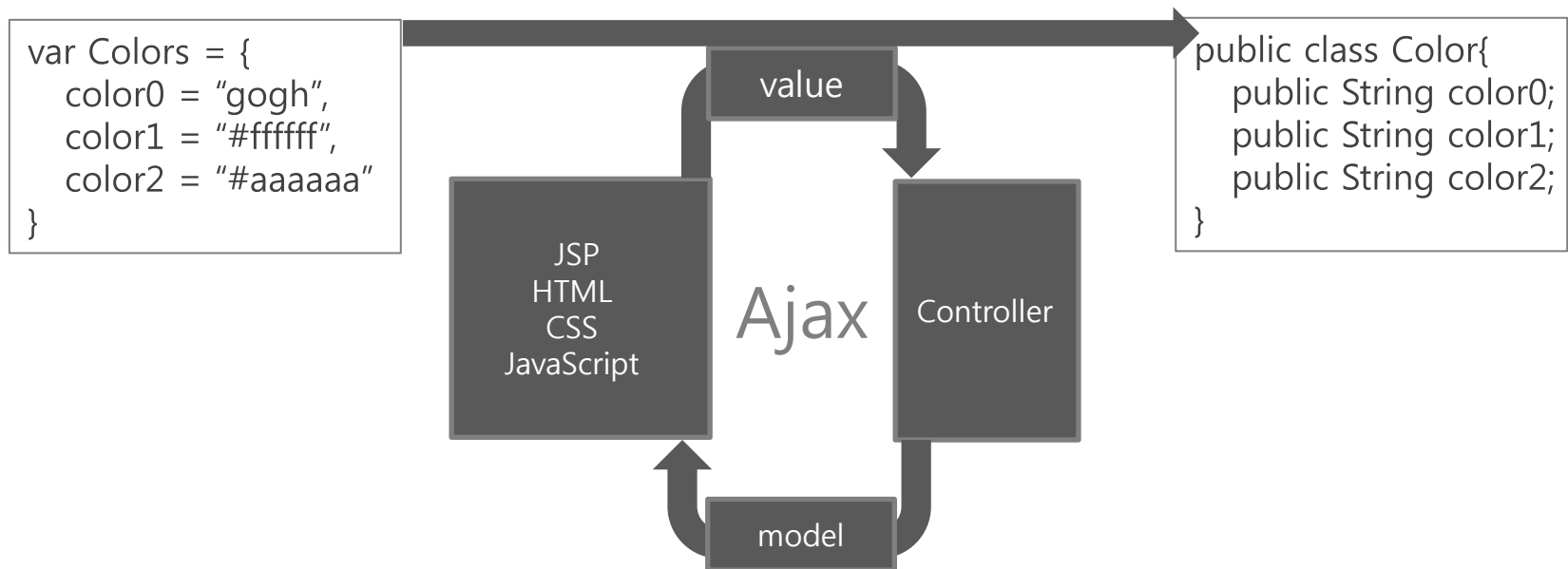
서버 개발

Ajax<->HomeController

Ajax

클라이언트 사이드에서는 Ajax로 보낼 데이터를 JSON 형식으로 파싱하여 보내고

홈 컨트롤러에서는 JSON 형식에 맞추어 만든 Color라는 객체로 Ajax로 받은 데이터를 받아 서비스로 보낸다.



3. 서버 개발

Ajax

Ajax

다음은 구글 크롬 웹 개발자툴로 확인한 Ajax통신이 성공되고 받은 데이터들이다.

select 문을 사용하여 색에 관련된 모든 테이블의 행의 colors 데이터들을 클라이언트 페이지에 저장한다.

```
▼<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>
```

4. 클라이언트 개발

WordCounter

WordConter

<https://github.com/fengyuanchen/wordcounter>

가장 많이 쓰인 색 순서대로 나열하고 정리하기 위해 사용하였다.

특수문자나 기호 제거가 용이하고
클라이언트 사이드에서 사용하기 간편하다.

Word Counter

[Home](#) [Code](#) [Example](#) [Docs](#) [GitHub](#)

```
1> if: 879
2> elem: 749
3> jQuery: 663
4> function: 612
5> return: 583
6> this: 464
7> the: 420
8> i: 415
```

4. 클라이언트 개발

WordConter

다음은 테이블 안의 모든 행들의 color를 select한
데이터들을 워드카운터를 사용한 모습이다.

WordCounter

1> 100c08: 329	59> 6c541e: 153
2> b08d57: 327	60> 555d50: 152
3> 2c1608: 308	61> 563c0d: 150
4> 9c7c38: 288	62> a67b5b: 150
5> 918151: 278	63> 836953: 149
6> 3d2b1f: 258	64> b78727: 148
7> 4b3621: 258	65> 483c32: 146
8> a99a86: 256	66> c5b358: 145
9> 965a3e: 249	67> deb887: 143
10> c39953: 247	68> 592720: 138
11> 8a795d: 245	69> d2b48c: 137
12> 81613c: 244	70> 556b2f: 134
13> 6e6e30: 237	71> 6b4423: 134

4. 클라이언트 개발

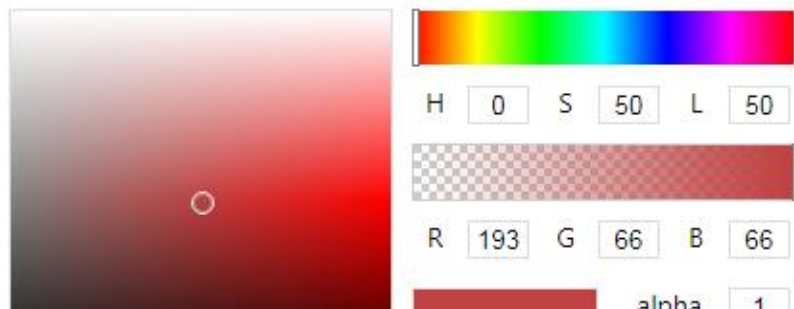
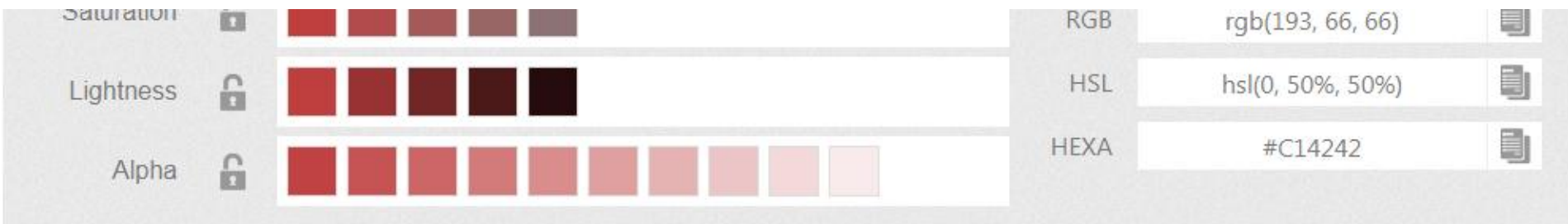
Color picker tool

Color picker tool

https://developer.mozilla.org/ko/docs/Web/CSS/CSS_Colors/Color_picker_tool

MDN에서 제공하는 웹 기준의 사용자 정의 색상을 쉽게 만들고 조정하고 실험 할 수 있는 툴.

이 프로젝트의 클라이언트 사이드의 대부분은 이 툴을 개조하여 작성하였다.



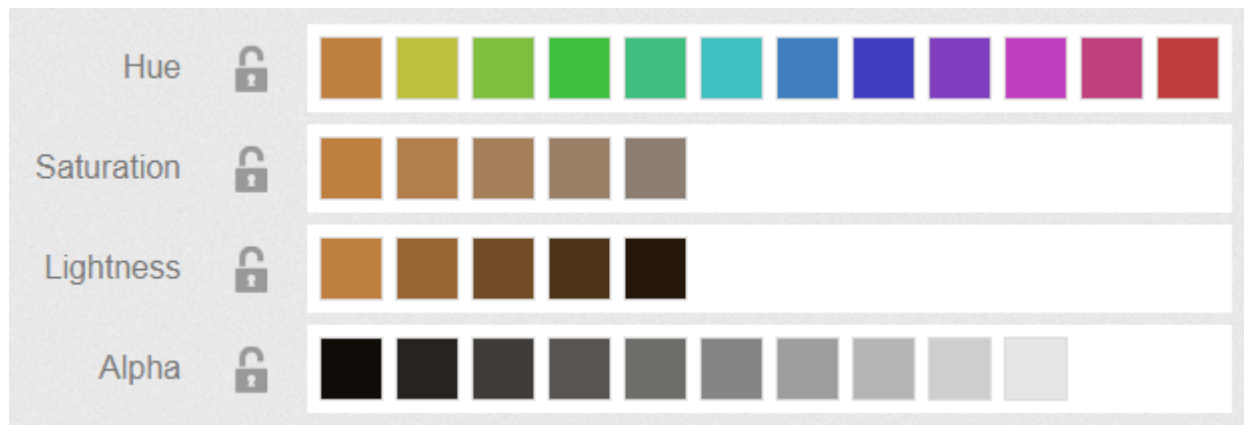
4. 클라이언트 개발

Color picker tool

Color picker tool

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

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



4. 클라이언트 개발

Color picker tool

Color picker tool

다음은 color picker tool안의 코드를 참조하여
기능을 넣고 워드카운트된 데이터들을 사용하여
나열한 색 선택지들을 구현한 것이다.

원래의 color picker tool과도 연동이 가능하다.



4. 클라이언트 개발

Color picker tool

마찬가지로 다음 이미지는 데이터들을 선택할 때
필요한 컬러들을 선택하는 기능을 만들기 위해 이용할
원래 color picker tool에 포함된 부분이다.

Color picker tool



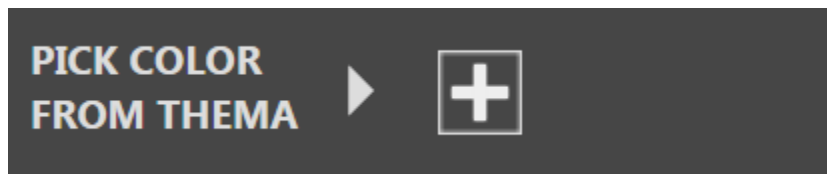
4. 클라이언트 개발

Color picker tool

Color picker tool

다음은 color picker tool안의 코드를 참조하여
색을 선택하고, 그 색은 자동으로 공식적인 색 목록의
색들로 변환 된 뒤, 셀렉트 쿼리 문에 사용될 수 있도록
한 부분이다.

클릭할 때마다 색 선택지가 업데이트 된다.

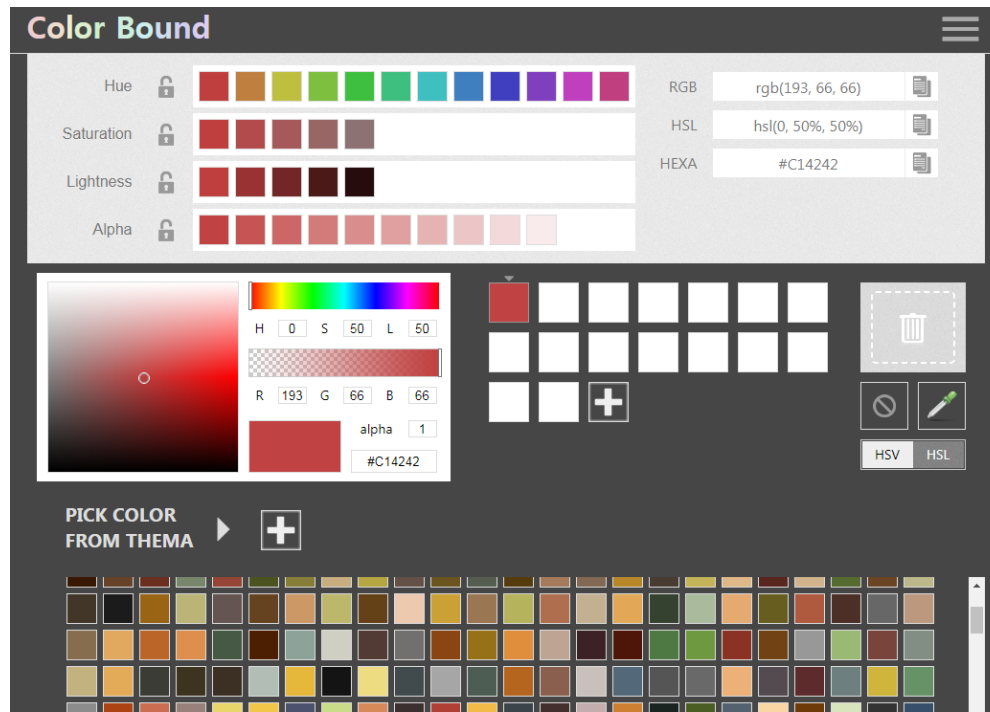


4. 클라이언트 개발

Color picker tool

최종적으로 완성된 모습.

Color picker tool



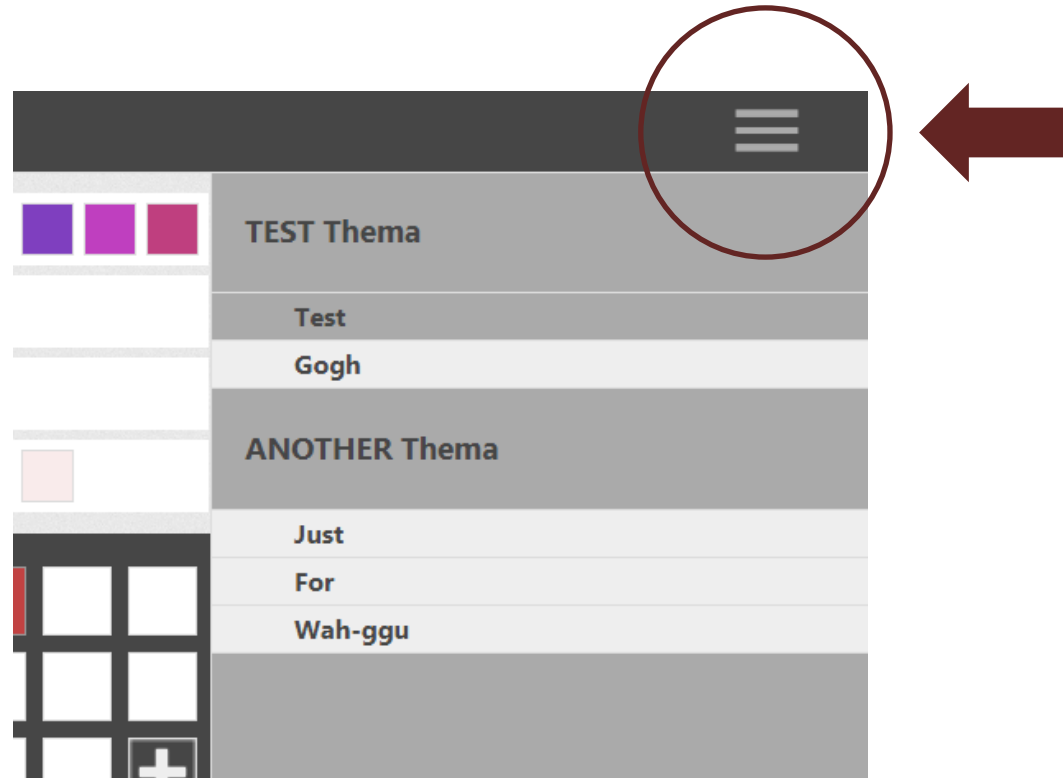
4. 클라이언트 개발

Color picker tool

Color picker tool

오른쪽 상단의 메뉴를 클릭하여
데이터베이스에서 불러 올 테마를 선택한다.

현재는 테스트용으로 테스트 테마와 고흐 테마 뿐이다.

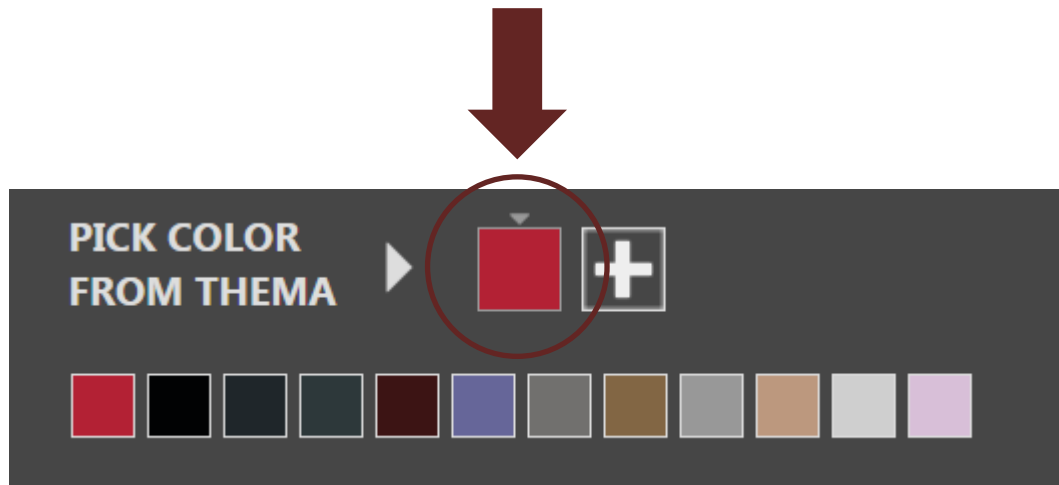


4. 클라이언트 개발

Color picker tool

pick color from thema에서 검색할 색을 선택한다.
선택할 색만큼 노드를 추가하고 클릭하면
그 색 또는 색들과 색 배합에 많이 쓰인
색 순서대로 나열된다.

Color picker tool



3

결론

추후 과제
발전 방향

3. 추후 과제

문제점_1

처음 홈페이지를 들어 왔을 때, 처음으로 보이는 색 선택지들, 즉 모든 색 데이터들 중에서 제일 많이 센 수의 색들 일수록 어둡고 탁한 색들이 많다.

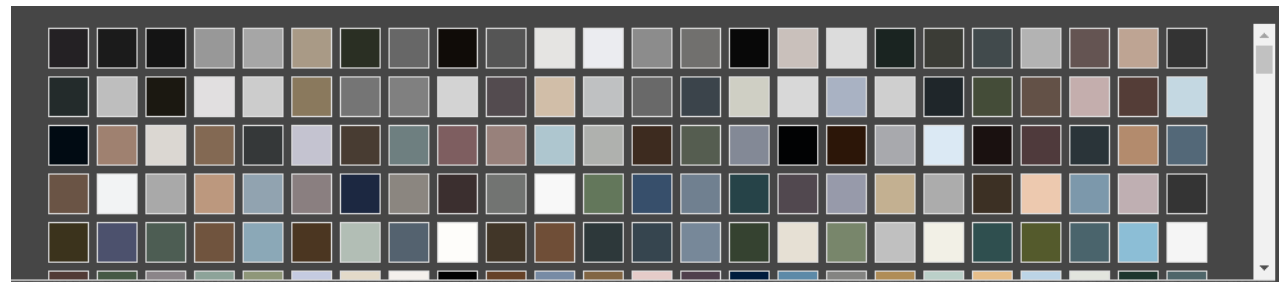
색 선택을 도와주는 서비스는 그만큼 다채롭고 화려한 색상을 먼저 보여야 하는 고정적인 이미지가 있는데, 그걸 배신하고 있다.

색을 선택하지 않는 이상은 보이지 않거나 다른 방법을 강구해야 할 것이다.

Gogh Thema



Test Thema

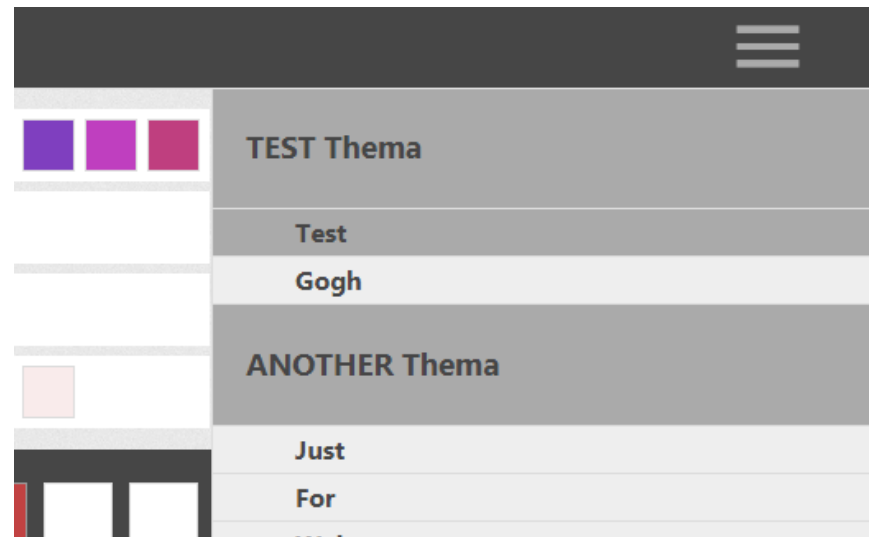
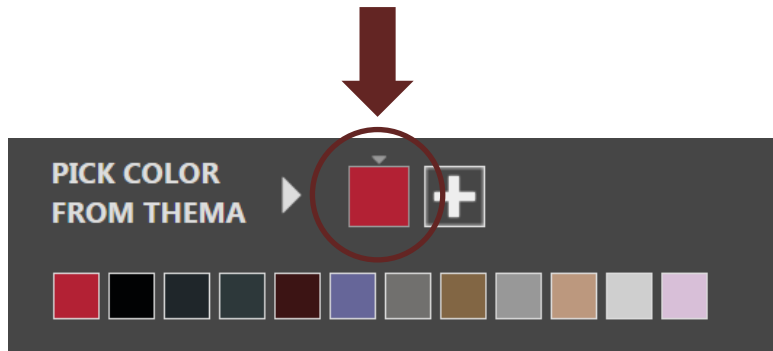


3. 추후 과제

문제점_2

선택지들의 업데이트는 반드시 pick color 부분을 클릭해야 동작한다.

따로 만들어 둔 테마를 고르는 부분에선 아직 동작 할 수 있는 방법을 찾지 못했다.



3. 추후 과제

문제점_3

고흐의 그림들 중에서
호불호가 심하게 갈리는 색감의 그림들이 많았다.

그리고 사진이 아닌 일러스트는 최적의 클러스터에서
10을 곱해 클러스터를 뽑아내서 색 조합을 저장했는데,

이 두 개의 이유 때문인지
고흐의 테마로 실험해본 프로젝트에서는
좋은 색을 뽑는 서비스를 실행하기 어려웠다.



3. 발전 방향

테마들

현재는 테스트를 위해 색감이 좋은 무작위 사진들과
고흐들의 그림들, 이 둘로 테마를 구성했지만
이 프로젝트는 테마를 특정한 목적으로 구현한다면
매우 강력한 작품이 될 수 있을 것이라고 생각한다.

예를 들어, 호평을 받은 패션 디자인의 옷들로 테마로
만들고 서비스 한다면, 자신이 원하는 색부터 시작해서
호평의 디자인의 색조합에 가까운 만족스러운
색 조합을 찾을 수 있을 것이고,

특정 유저가 좋아하는 스타일의 화가로 테마를 만들어
서비스를 한다면, 그 화가의 화풍의 색 조합들을 찾아내서
이용 할 수 있을 것이다.

이처럼 패션이나 미술, 디자인 등에서 테마에 따라
가능성은 많은 편이라고 생각한다.

