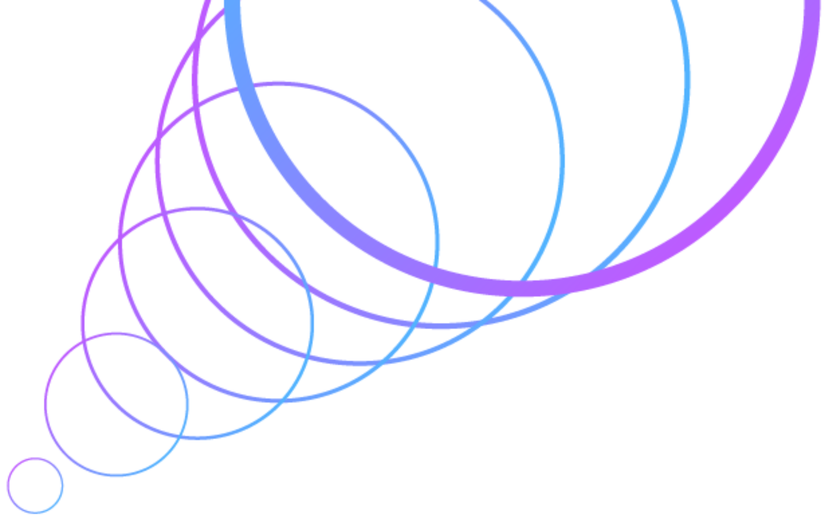


DEVIEW 2021

# 데이터 시각화의 꽃 APM, FE 개발 이야기

김동빈, 김도윤 NAVER PLATFORM LABS

# CONTENTS

# APM

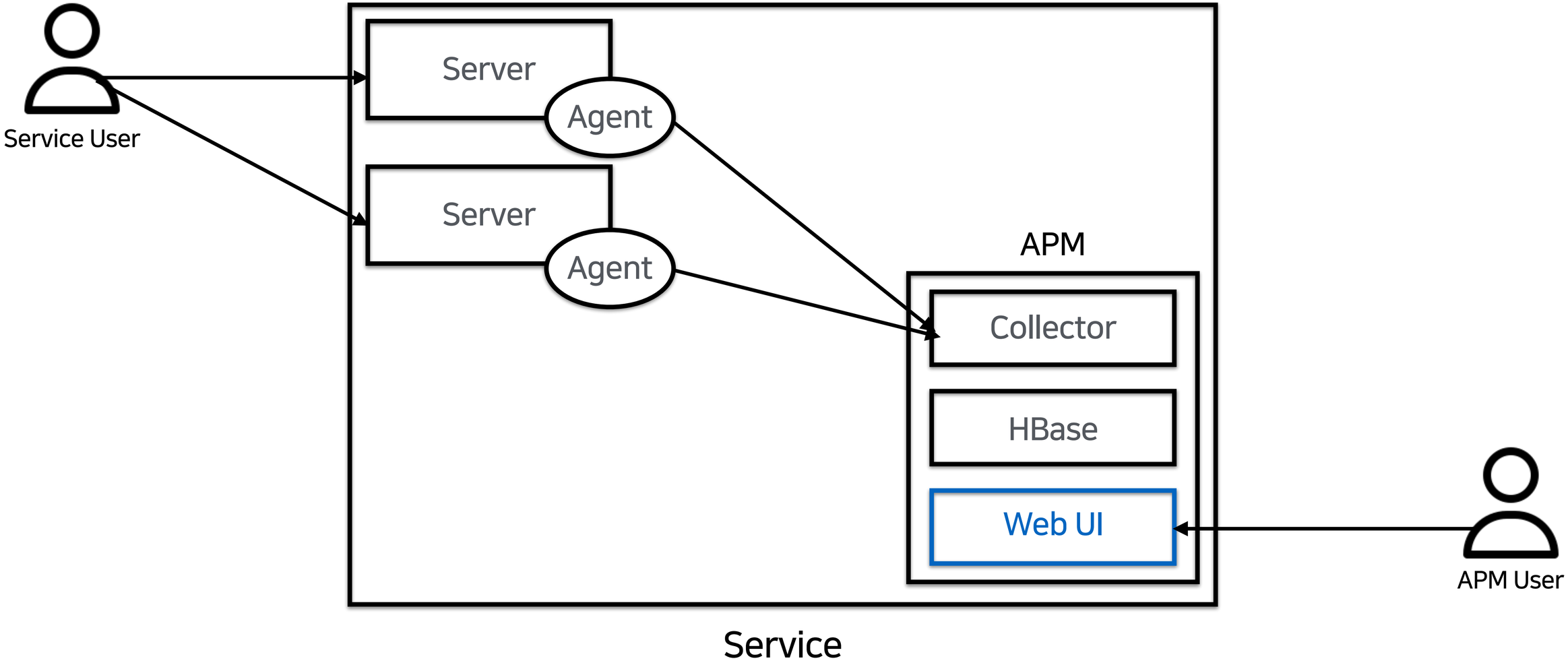
# APM이란



Application Performance Management

# APM이란




Service User
Server
Agent
Server
Agent
APM
Collector
HBase
Web UI
APM User
Service

# APM - Pinpoint

DEVIEW 2021


https://github.com/pinpoint-apm/pinpoint

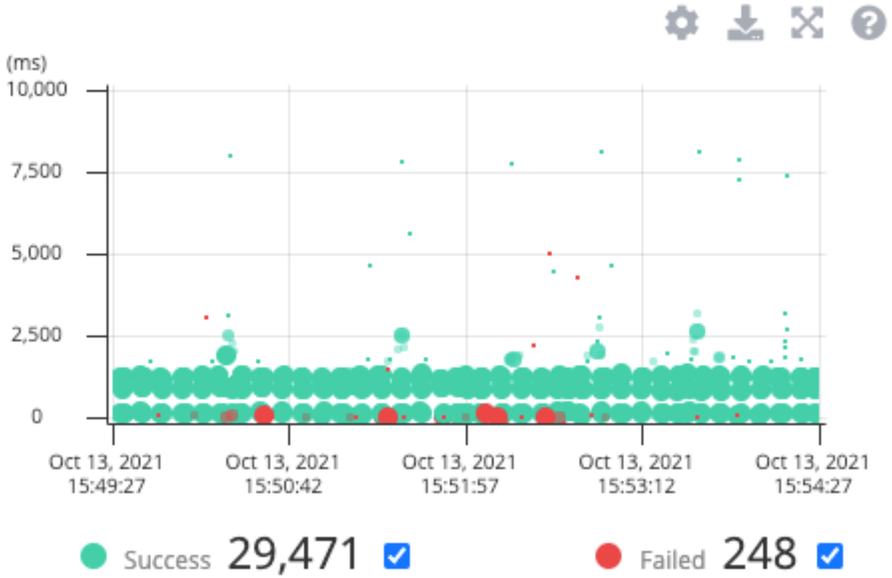
# APM에서의 FE개발

DEVIEW 2021

## APM FE개발자의 주요업무: 데이터 시각화

# 주요 시각화 컴포넌트들



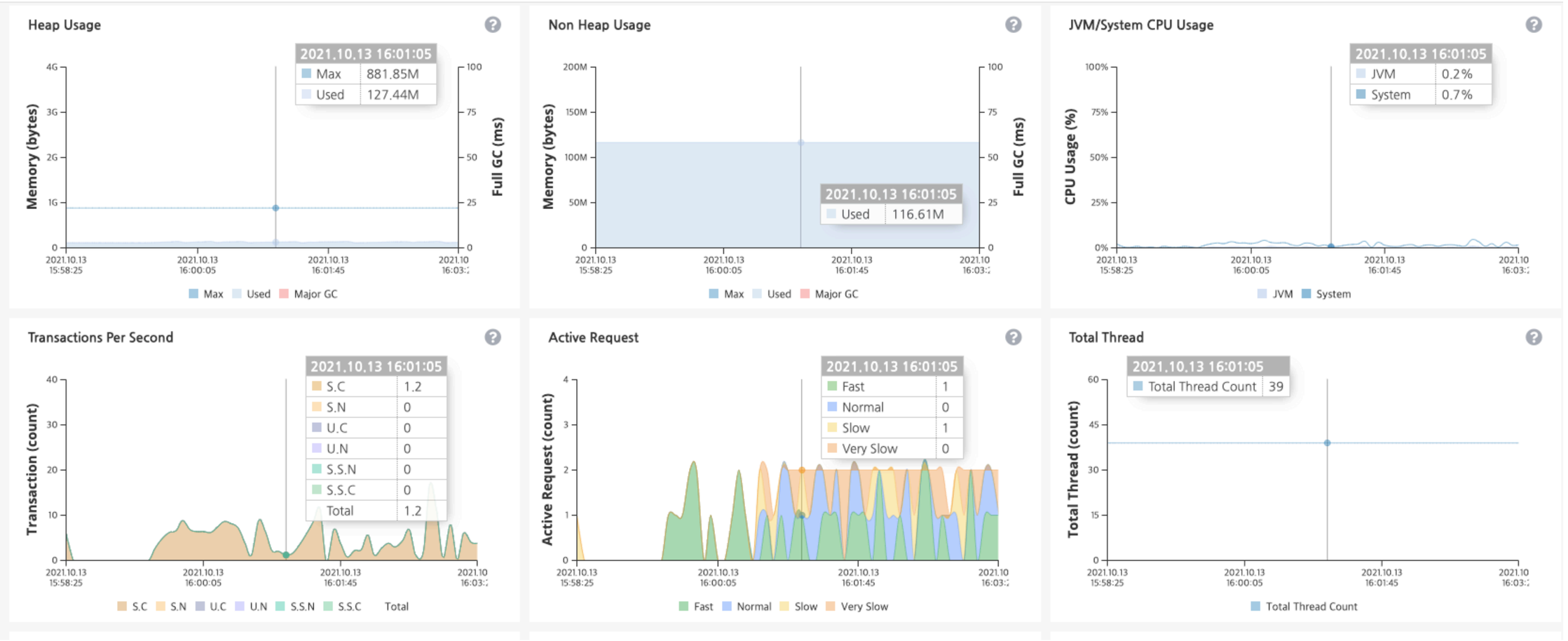



Scatter Chart


Server Map

# 주요 시각화 컴포넌트들





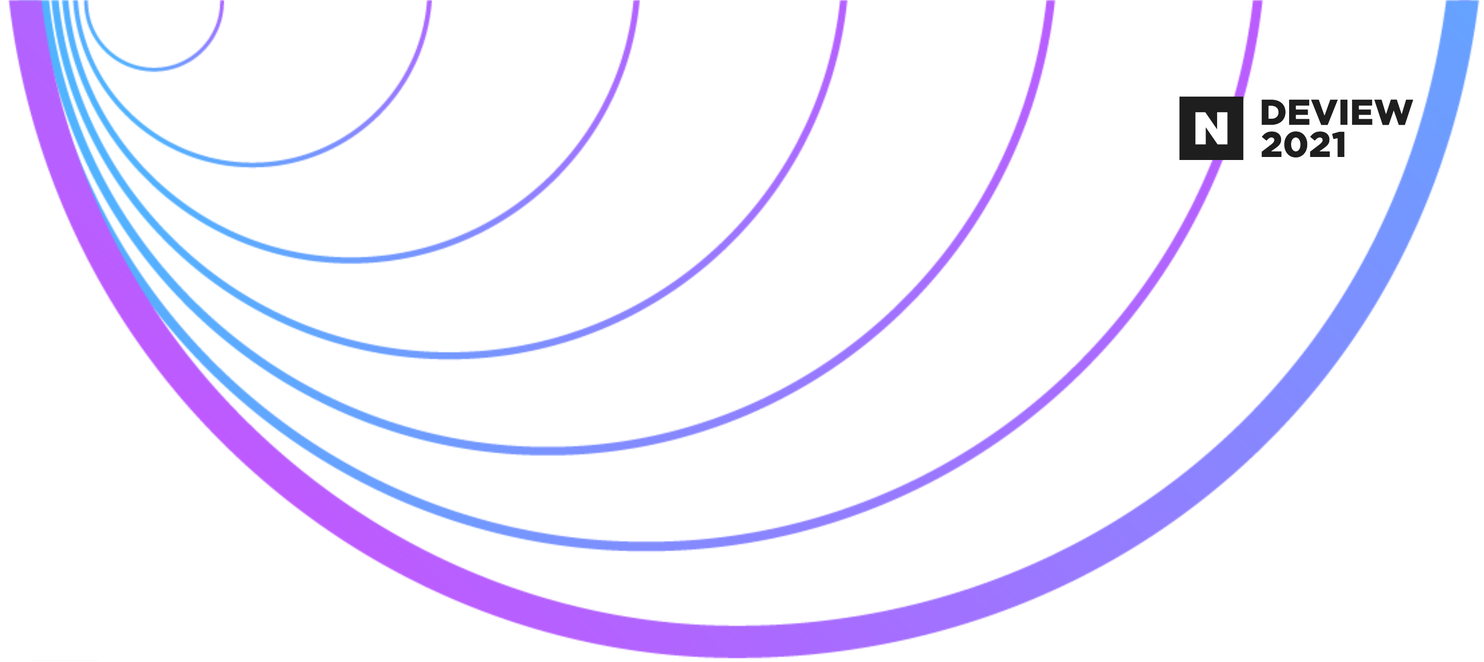

Inspector Chart

# APM에서의 FE개발



단순히 여러 차트, 시각화 컴포넌트들 개발만하는 것? Yes and No

그럼 APM FE 개발자가 신경써야 할 것(Focus)들은?



DEVIEW 2021

# Focus

# APM의 목표



사용자에게 좋은 모니터링 환경을 제공하기

# 목표 달성을 위해선?

DEVIEW 2021

## 성능

많은 데이터에도 원활한 동작이 가능해야 함

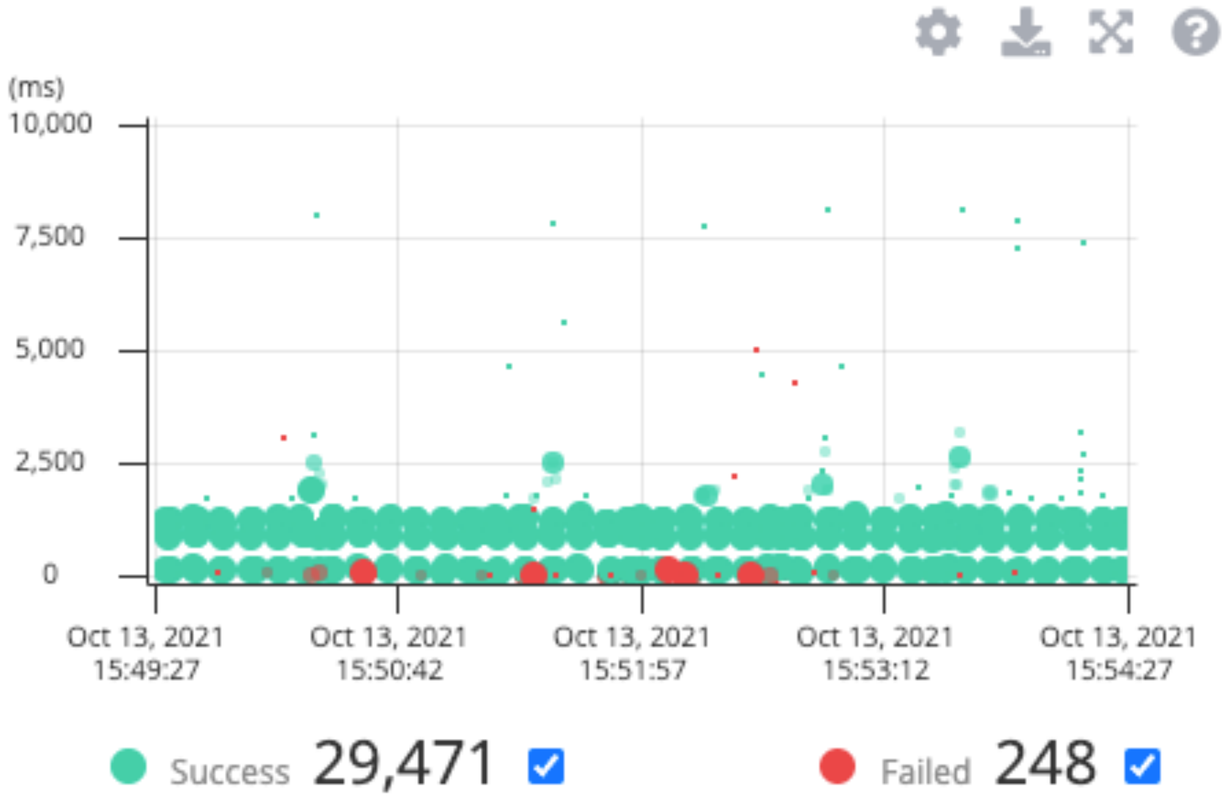
## 직관성

사용자가 우리가 제공한 지표들을 통해 그 의미들을 쉽게 이해할 수 있어야 함

# 고민과 해결

# Scatter Chart에서의 성능 개선

# Scatter Chart란




Scatter Chart

수집된 Transaction들의
시간대 별 응답시간 분포도

# Scatter Chart 구현하기



## 준비물


데이터

×


HTML Canvas Element

# Javascript Canvas API

DEVIEW 2021

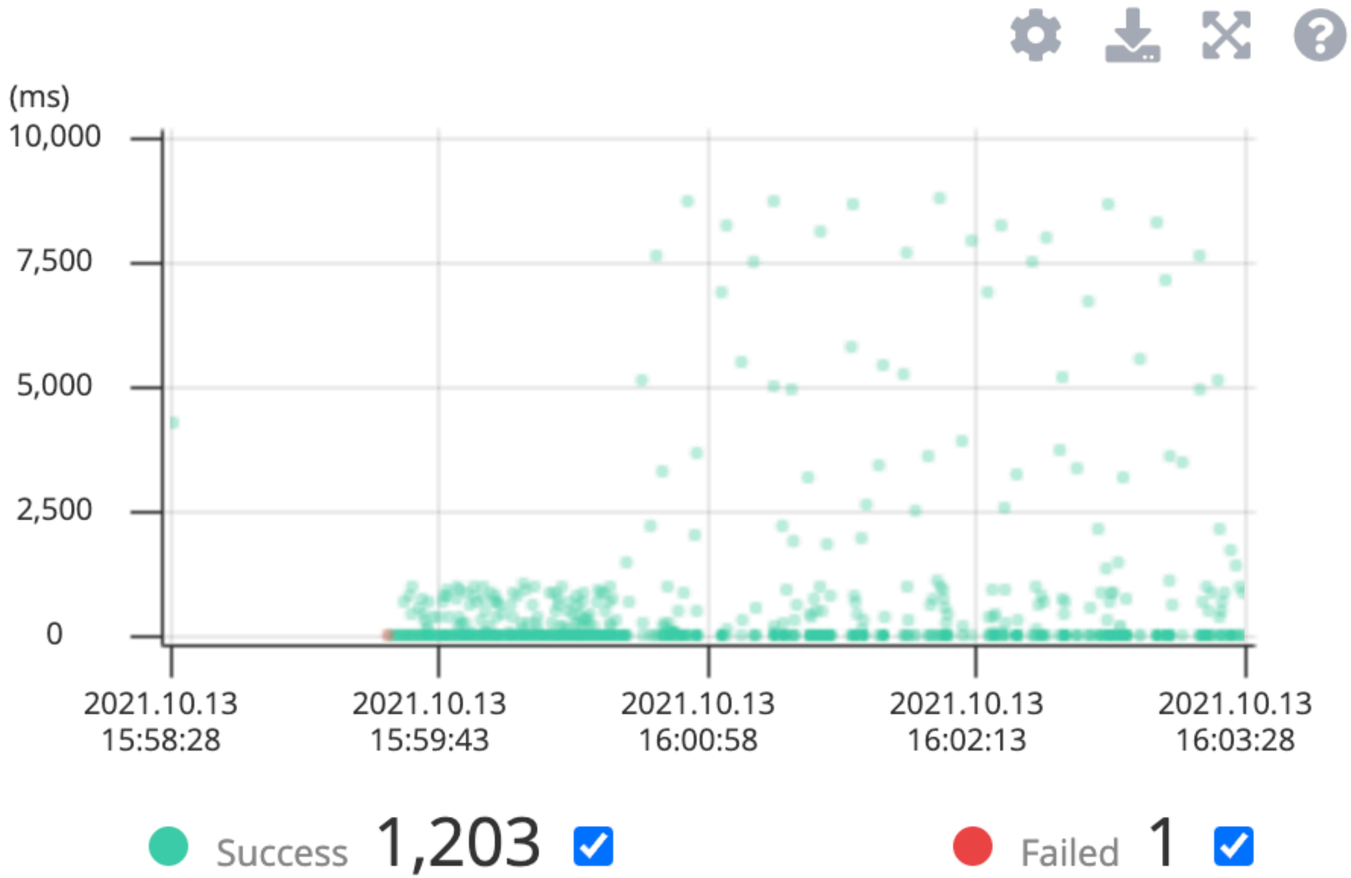
Let’s draw circles!

```
const context = canvas.getContext('2d');

context.beginPath();
context.fillStyle = color;
context.strokeStyle = color;
context.arc(x, y, r, 0, Math.PI * 2, true);
context.fill();
```

× 데이터의 개수

# 완성된 모습



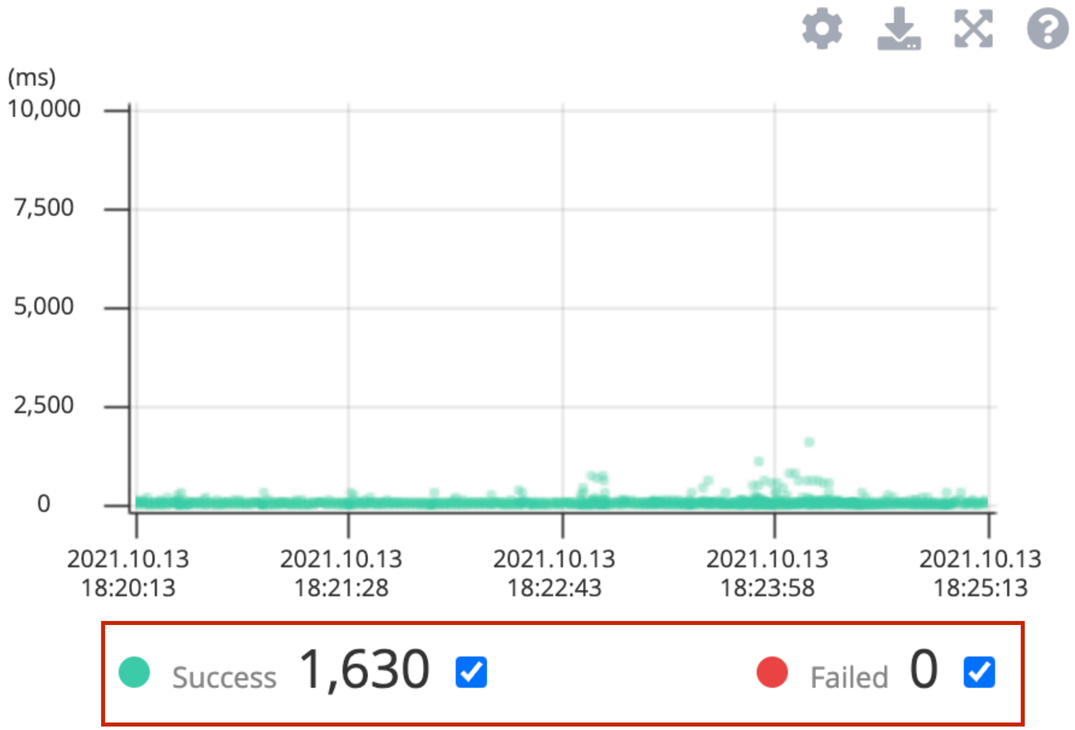



Scatter Chart 완성!

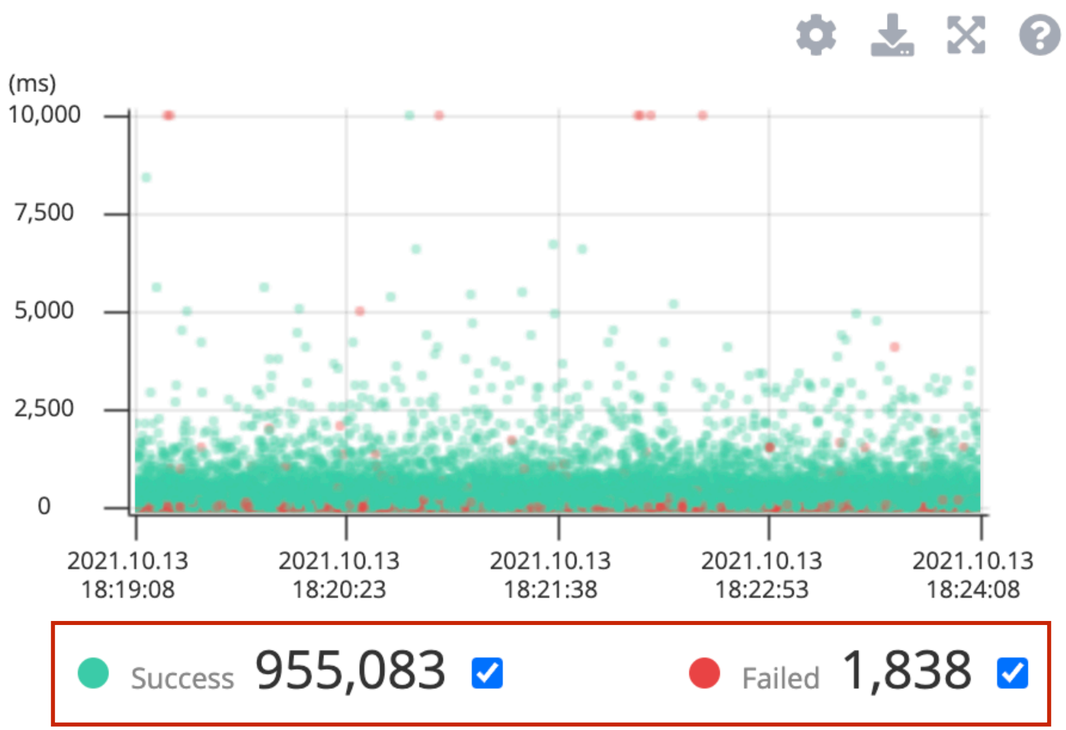
# 고민 - 대용량 데이터 렌더링

DEVIEW 2021

- 성능
- 효과적 표현




VS




🙂 😕

# 고민 해결하기 - Canvas Optimization

DEVIEW 2021

- 반복적인 요소를 미리 그려놓고 가져다 쓰기
- 필요한 크기들은 미리 그려놓기 (drawImage에서 크기 조절하지 않기)
- Multi-layered Canvas 사용하기

출처 - Canvas Optimization MDN

# 반복적인 요소를 미리 그려놓고 가져다 쓰기

DEVIEW 2021

기존:

```
const context = canvas.getContext('2d');

context.beginPath();
context.fillStyle = color;
context.strokeStyle = color;
context.arc(x, y, r, 0, Math.PI * 2, true);
context.fill();
```

× 데이터의 개수

개선:

```
const context = canvas.getContext('2d');

context.drawImage(preRenderedCircle, x - r, y - r);
```

× 데이터의 개수

효과: 매번 원을 그리는 비용을 아낄 수 있음.

# 필요한 크기들은 미리 그려놓기

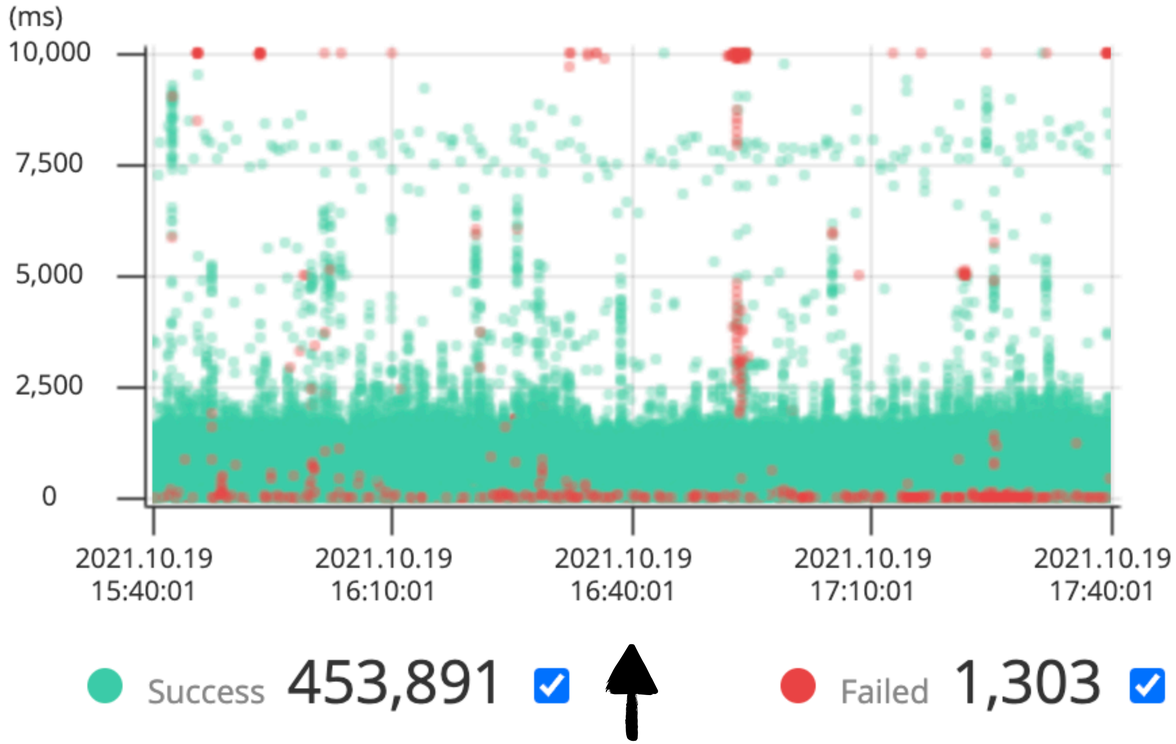


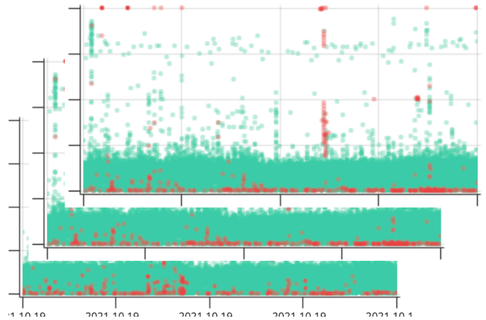
```
const offscreenCanvasMap = new Map<number, HTMLCanvasElement>();

for (let r = maxRadius; r > 0; r--) {
    const canvas = document.createElement('canvas');
    const ctx = canvas.getContext('2d');
    const size = r * 2;
    const x = r;
    const y = r;

    canvas.width = size;
    canvas.height = size;

    ctx.beginPath();
    ctx.fillStyle = color;
    ctx.strokeStyle = color;
    ctx.arc(x, y, r, 0, Math.PI * 2, true);
    ctx.fill();

    offscreenCanvasMap.set(r, canvas);
}

// Use circles
context.drawImage(offscreenCanvasMap.get(r), x - r, y - r);
```

1 ~ maxRadius 반지름의 원들을 미리 그려놓고, 필요한 사이즈의 원을 가져다 사용하기

# Multi-layered Canvas 사용하기

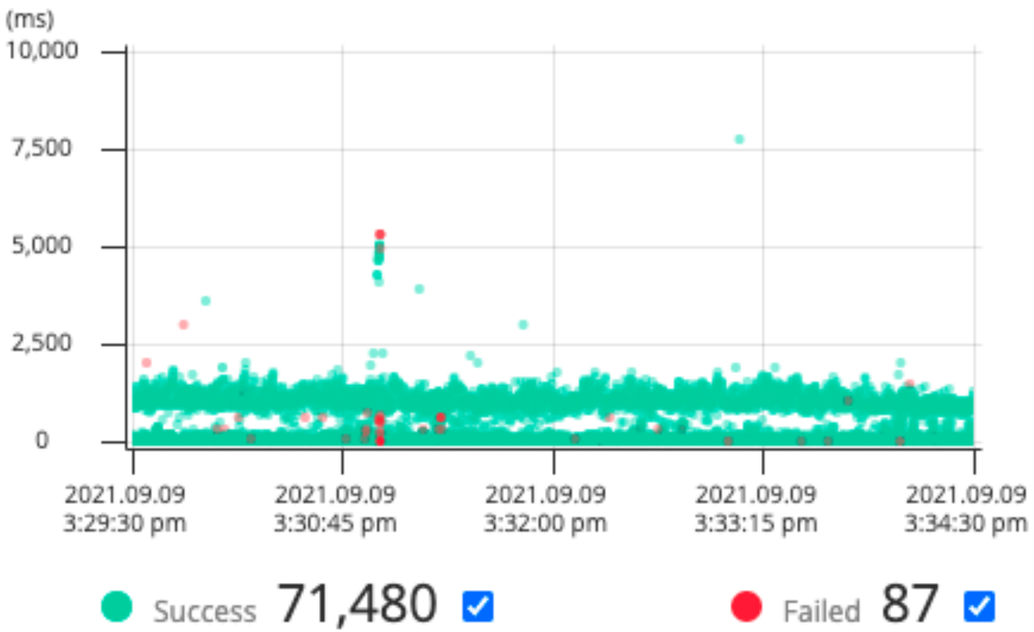
DEVIEW 2021



```
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 0px; z-index: 110; position: absolute;
display: block;">
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 354px; z-index: 111; position: absolut
e; display: block;">
<canvas width="354px" height="170px" data-type="failed" style
"top: 0px; left: 0px; z-index: 112; position: absolute; displ
y: block;">
<canvas width="354px" height="170px" data-type="failed" style
"top: 0px; left: 354px; z-index: 113; position: absolute; dis
lay: block;">
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 0px; z-index: 114; position: absolute;
display: block;">
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 354px; z-index: 115; position: absolut
e; display: block;">
<canvas width="354px" height="170px" data-type="failed" style
"top: 0px; left: 0px; z-index: 116; position: absolute; displ
y: block;">
<canvas width="354px" height="170px" data-type="failed" style
"top: 0px; left: 354px; z-index: 117; position: absolute; dis
lay: block;">
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 0px; z-index: 118; position: absolute;
display: block;">
<canvas width="354px" height="170px" data-type="success"
style="top: 0px; left: 354px; z-index: 119; position: absolut
e; display: block;">
```

# 고민 해결하기 - Data Sampling




기존의 Scatter Chart

## 기존의 Scatter Chart 렌더링 방식

- 모든 Transaction들을 동일한 크기의 원으로 표현
- 데이터와 그리는 원을 1:1로 매칭: 데이터 500개 = 원 500개

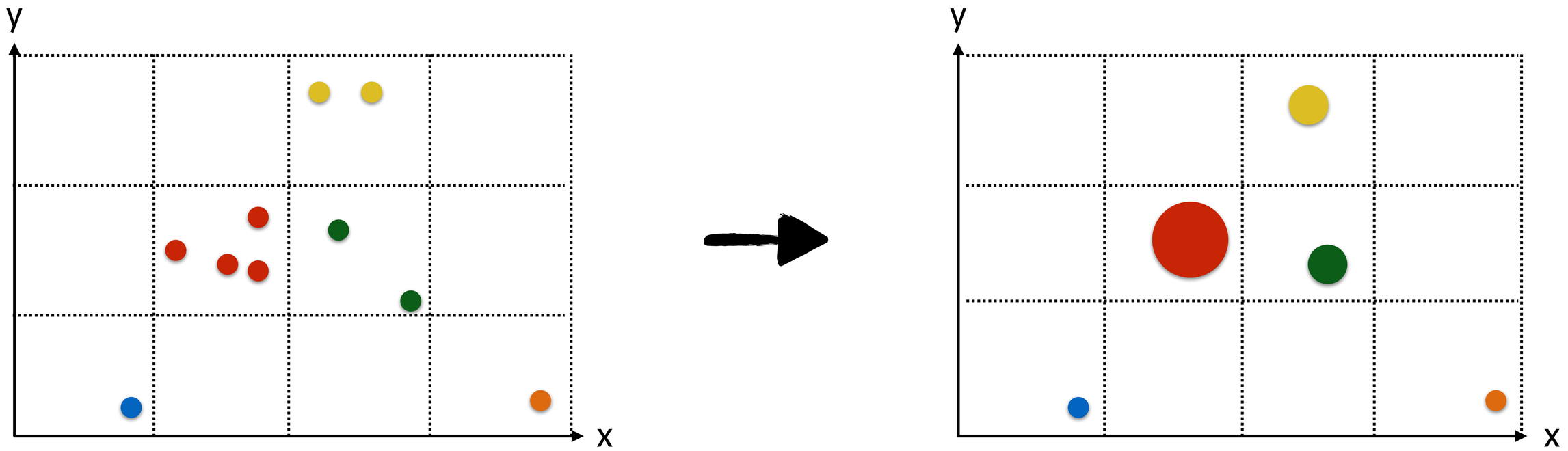
## 고민

- 특정 구간에 여러개의 원이 겹쳐져 있는 경우, 인식하기 어려움
- 있는 그대로 다 그리는게 의미가 있을까 의구심이 생김

# Data Sampling



Canvas 영역을 가상의 그리드로 나누어서, 각 영역에 위치하는 점들을 하나의 점으로 표현하기



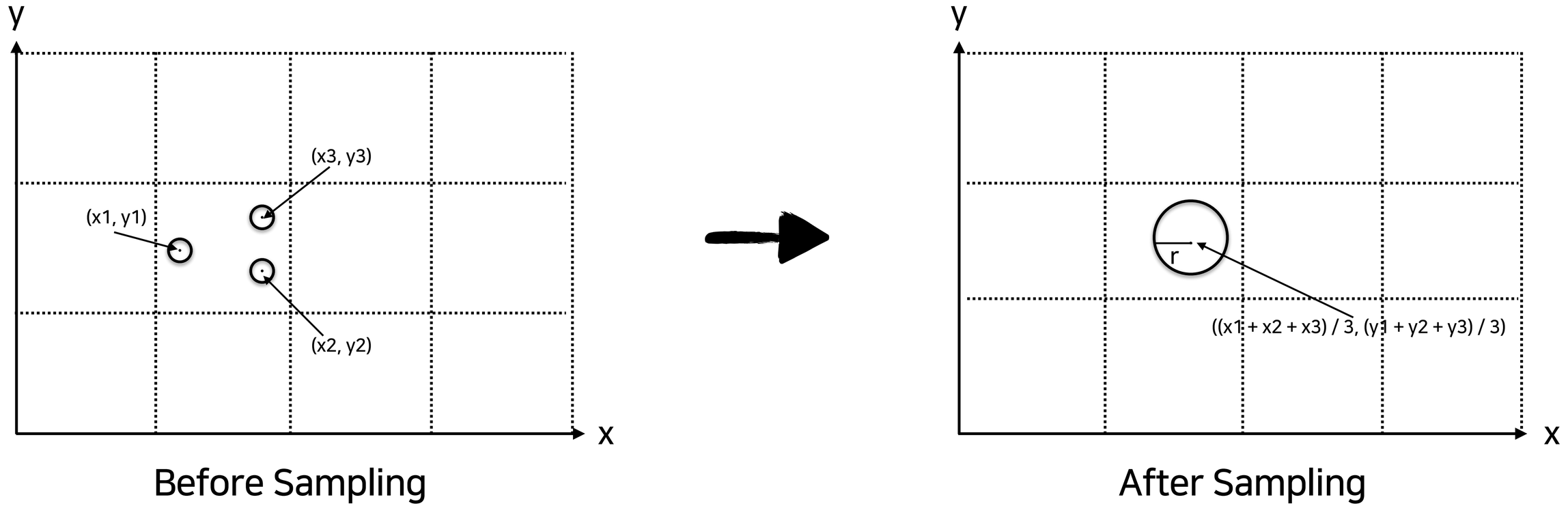

Before Sampling

After Sampling

# 고려할 점들



- 샘플링 된 원의 위치는 샘플링 대상들 x, y 값의 평균값.
- 샘플링 된 원의 크기(반지름)는 샘플링 대상들의 개수에 비례하도록 하되, 최대 반지름은 가상의 그리드 영역 크기의 절반 (r = count * 2 >= gridUnit ? gridUnit / 2 : count)




Before Sampling

After Sampling

# Sampling 진행하기



1. 각각의 데이터를 Canvas에서의 x, y 위치에 따라 가상의 그리드 영역에 넣어준다.

```
{
  [`${x1}-${x2}-${y1}-${y2}`]: [dot1, dot2, ...],
  [`${x2}-${x3}-${y1}-${y2}`]: [dot3, dot4, ···],
  [`${x3}-${x4}-${y1}-${y2}`]: [dot5, dot6, ···],
  ...
}
```

그리드 영역에 할당된  점 데이터들

# Sampling 진행하기



2. 각 그리드 영역별로 점 데이터 하나를 만들어낸다.

```
{
  [`${x1}-${x2}-${y1}-${y2}`]: [dot1, dot2, ...],
  [`${x2}-${x3}-${y1}-${y2}`]: [dot3, dot4, ···],
  [`${x3}-${x4}-${y1}-${y2}`]: [dot5, dot6, ···],
  ...
}
```

그리드 영역에 할당된  점 데이터들

```
[
  {x: x1, y: y1, count: 3},
  {x: x2, y: y2, count: 4},
  {x: x3, y: y3, count: 5},
  ...
]
```

그리드 영역 당 점 데이터 하나 만들어냄.

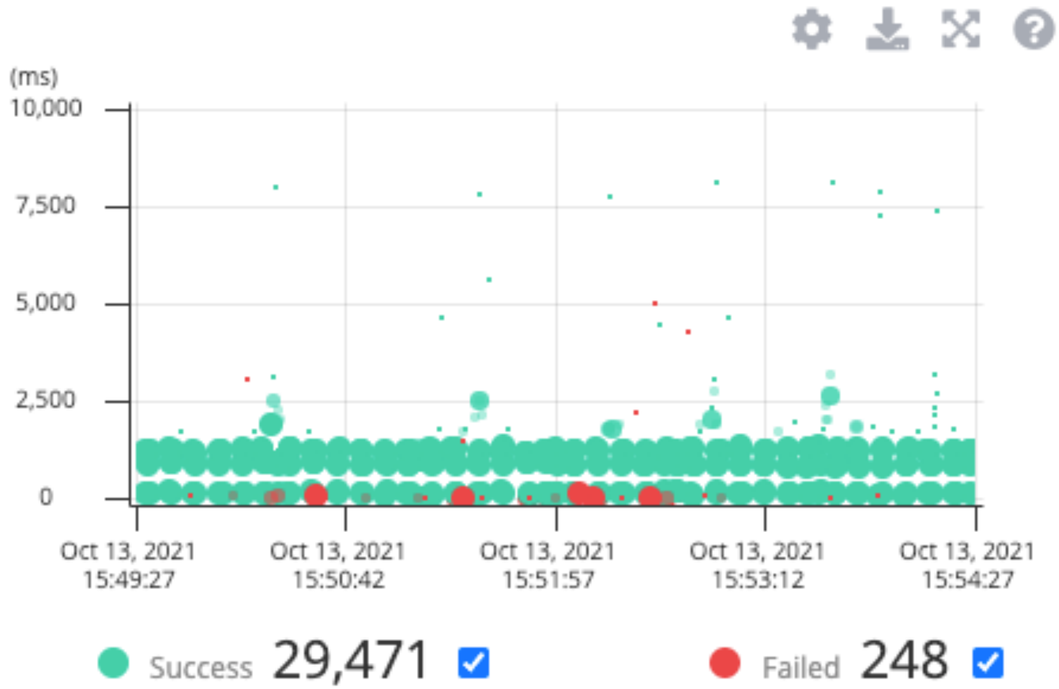
count: 각 그리드 영역에 포함된 데이터의 개수

# Sampling 데이터로 그리기

DEVIEW 2021

```
[
   {x: x1, y: y1, count: 3},
   {x: x2, y: y2, count: 4},
   {x: x3, y: y3, count: 5},
   ...
]
```

실제 Scatter Chart를 그릴 데이터

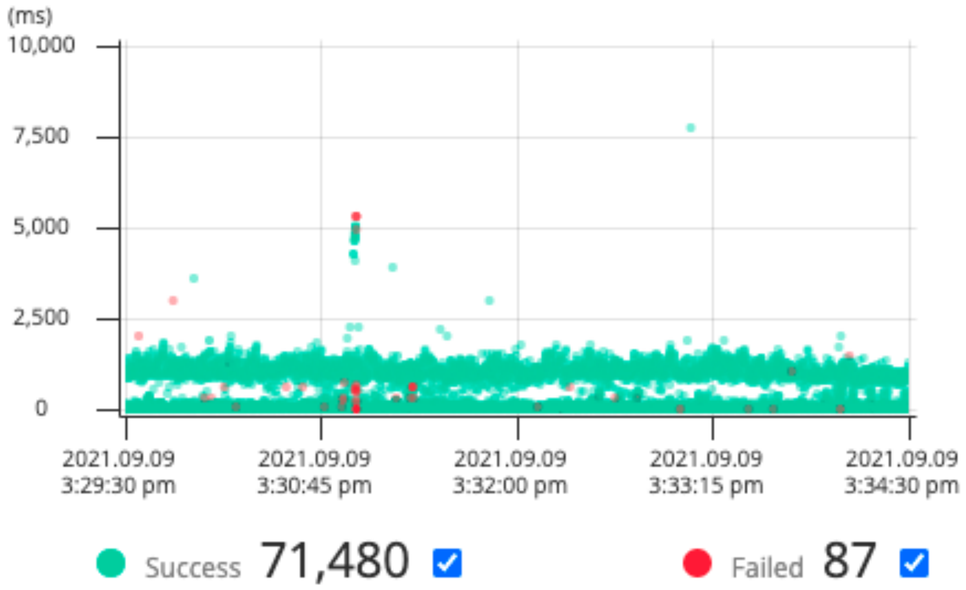



Sampled Scatter Chart

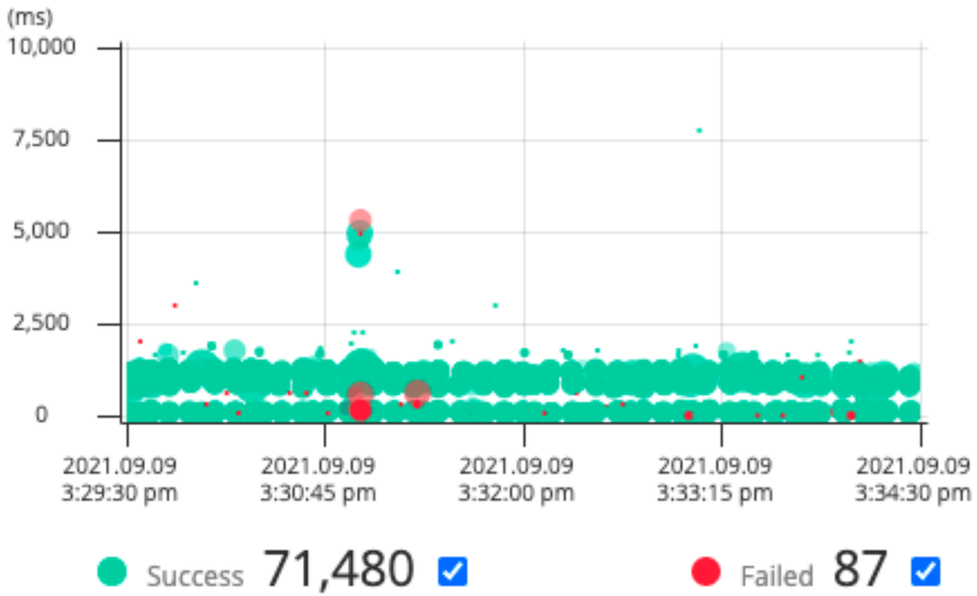
# 비교 및 개선 효과



- Canvas에 그리는 원의 개수 감소 (gridUnit 10px 기준, 약 92% 감소)
- 사용자의 인지에 도움




Sampling 전




Sampling 후

# 한눈에 알아보기 쉬운 Server Map

# Server Map이란

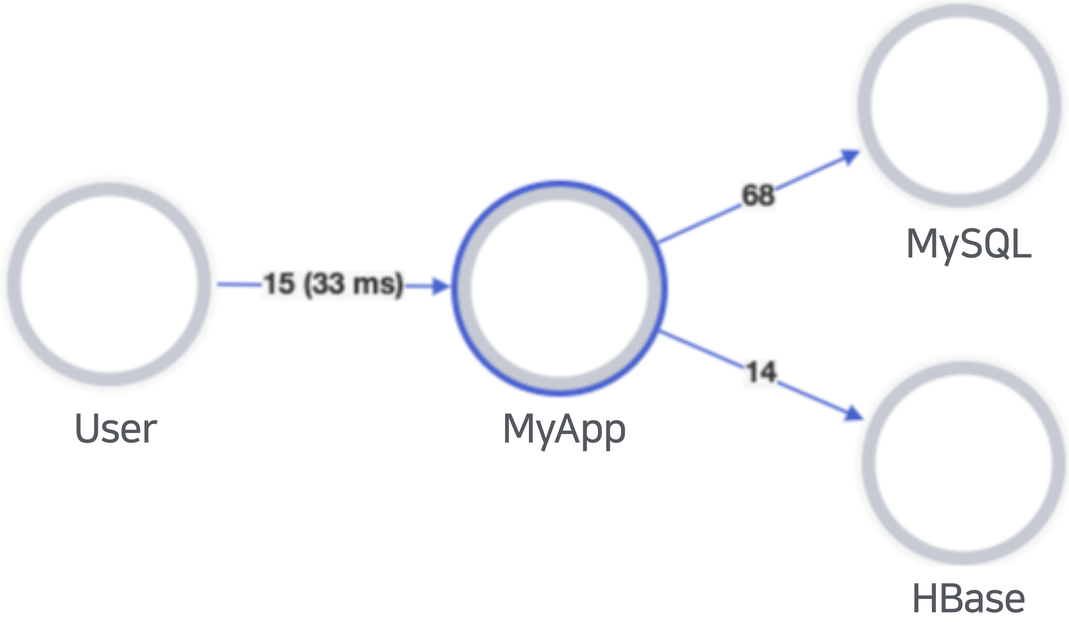



Server Map

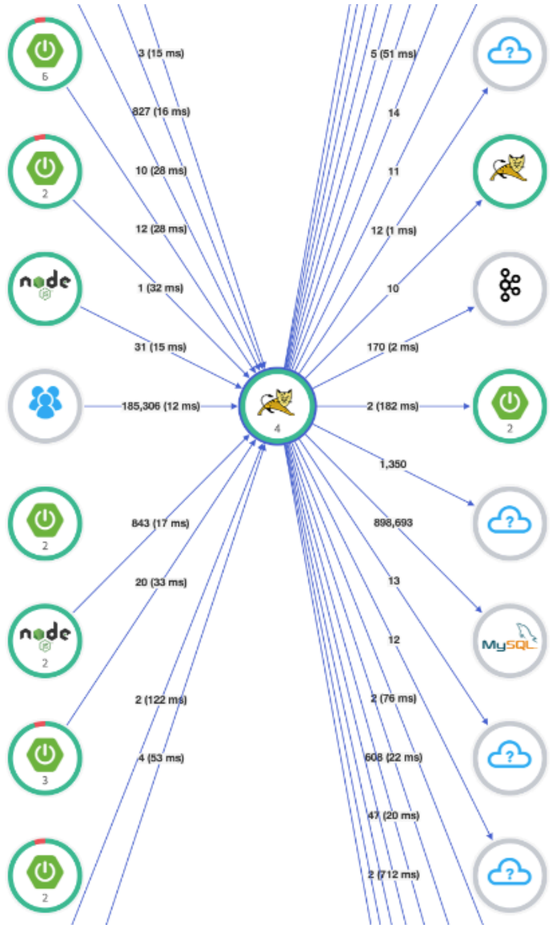
애플리케이션의 구성을
네트워크 맵의 형태로 시각화

# 고민

- 효과적 노드 표현
- Server Map 복잡도 개선




Unkind Node Expression

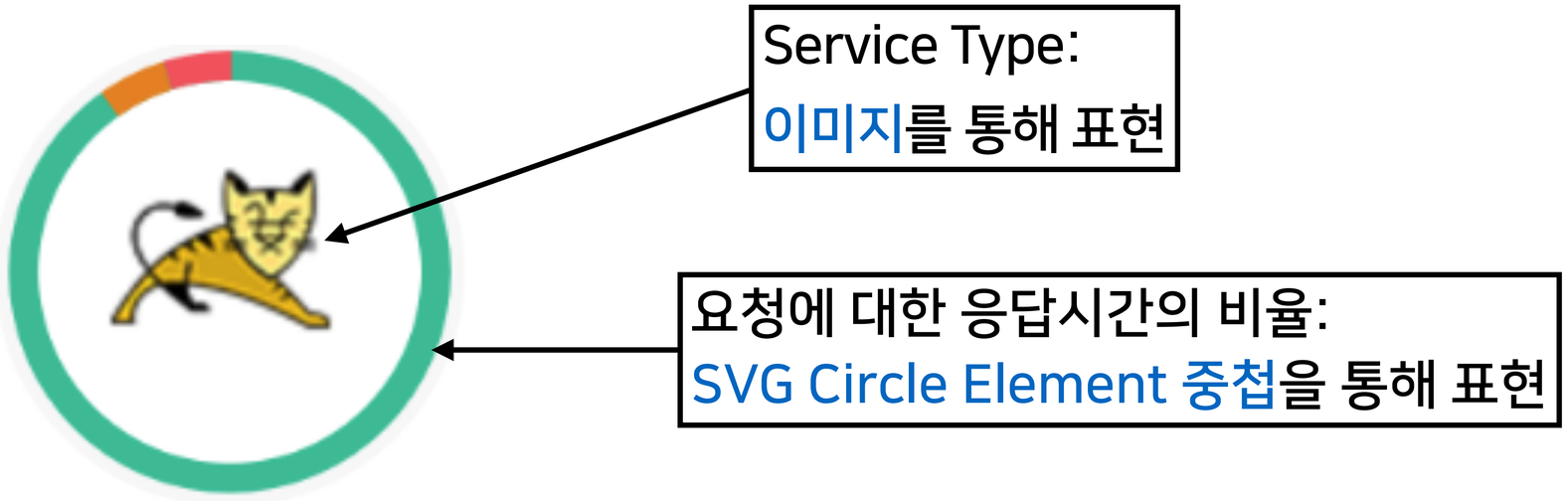



Complex Server Map

# 고민 해결하기 - 효과적 노드 표현

DEVIEW 2021

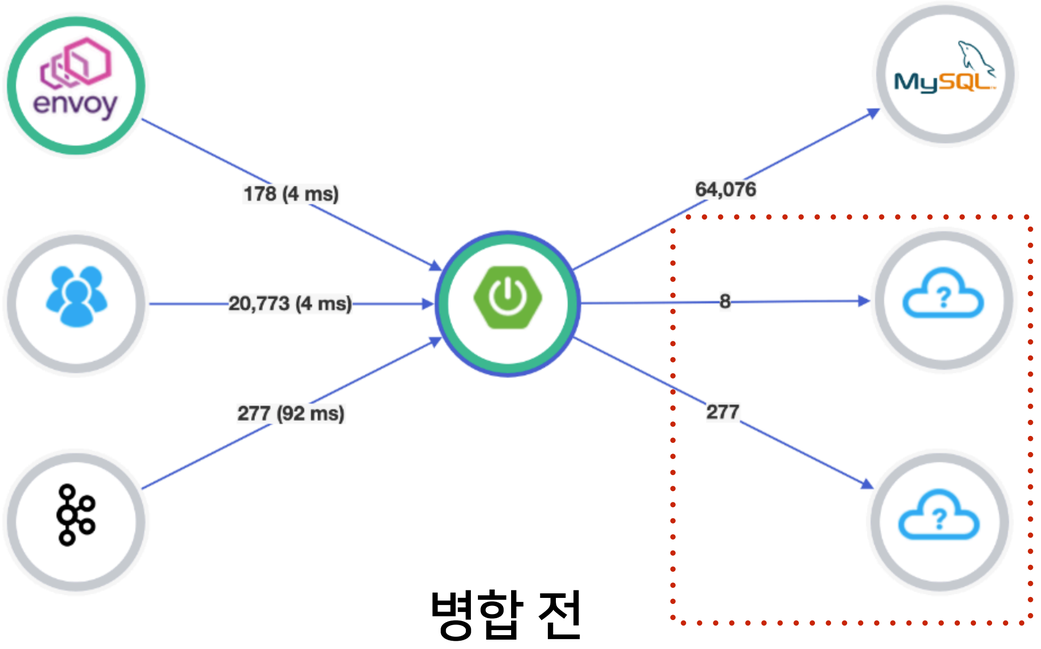
- 최소한으로 모니터링에 도움을 줄만한 정보만 제공
- Service Type: Node.js, Spring Boot, …
- 요청에 대한 응답시간의 비율: Good, Slow, Error



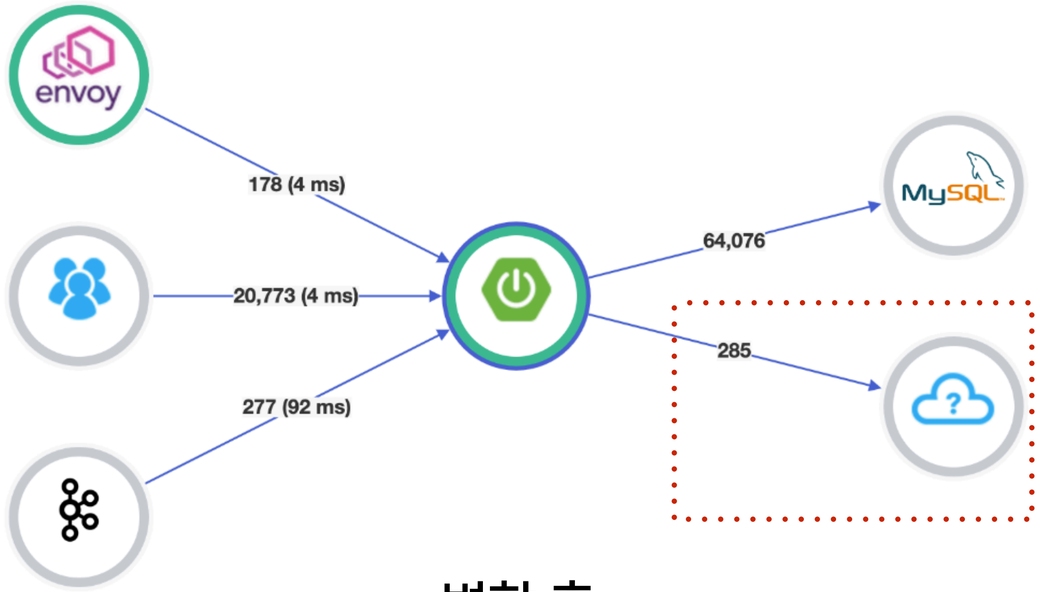
# 고민 해결하기 - Server Map 복잡도 개선

DEVIEW 2021

- 몇가지 조건에 따라 노드 및 링크를 병합하여 보여줌
- 조건 예시: Service Type이 같고, 같은 노드로부터 요청을 받으며, 종단 노드인 경우




병합 전




병합 후

# 정리

DEVIEW 2021

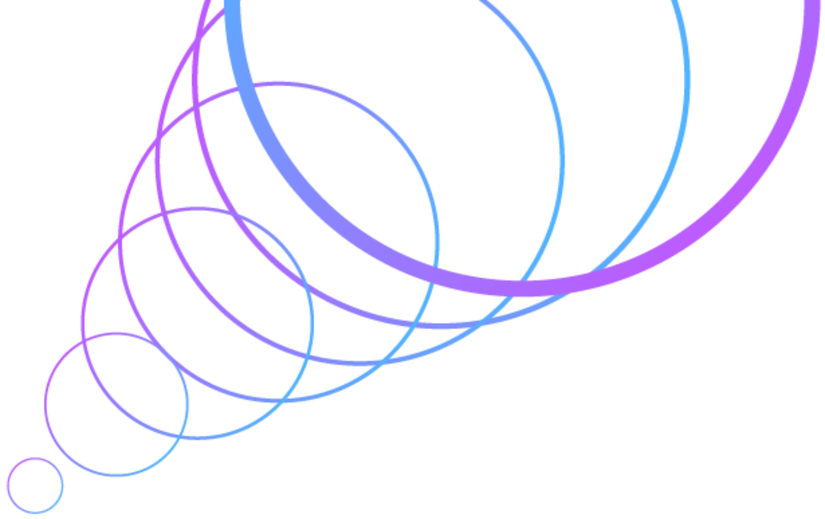
## Scatter Chart 성능 개선

- Canvas Optimization
- Data Sampling

## Server Map 직관성 개선

- 효과적 노드 표현
- 서버맵 복잡도 개선(feat. 노드 및 링크 병합 처리)

"좋은 모니터링 환경 제공하기"



DEVIEW 2021

# 데이터 시각화의 꽃 APM, FE 개발 이야기

김동빈, 김도윤 NAVER PLATFORM LABS

DEVIEW 2021

# UI/UX 개선 경험 공유

DEVIEW 2021

# 대규모 프로젝트에 Color System 적용하기

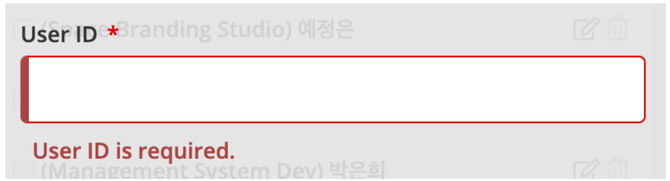
(디자이너 없이, 다크모드는 덤!)

# 배경

DEVIEW 2021

## 규칙없는 컬러 스타일..

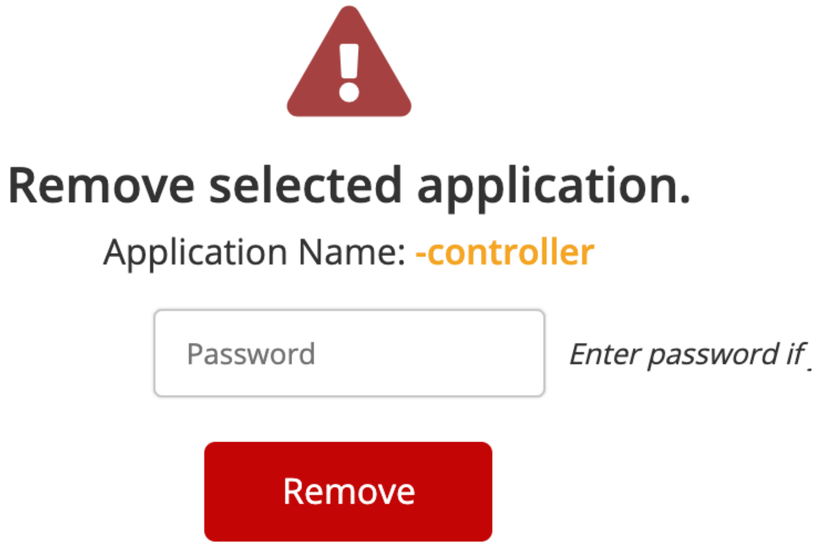
어떤 컬러를 사용해야 하지?
나는 개발자인가 디자이너인가?
(디자이너가 없는 팀의 상황)

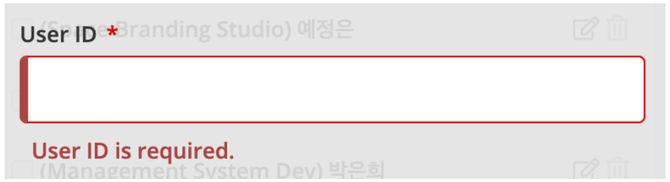

User ID *
User ID is required.


TESTds
Http failure response for /userGroup.pinpoint: 0 Unknown Error
test_header

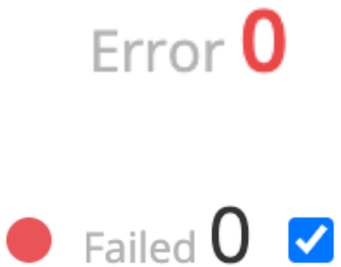

Error 0


Failed 0


Remove selected application.
Application Name: -controller
Password
Enter password if
Remove








#DD0000

#A94442

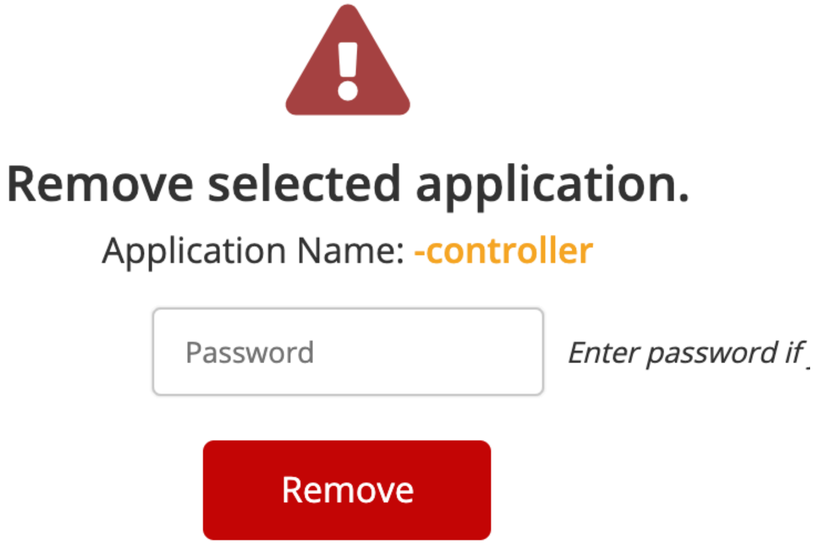
#FF8C00

#A94442

#C70606

#EB4747

#E95459




실패또는 에러 상태를 나타내는 공통된 목적을 가진 컴포넌트에 다른 컬러코드가 사용됨

# 기준 없는 컬러 사용의 문제점



- 일관성 없이 적용된 컬러 코드는 개발 생산성을 저해함
- 사용자의 직관적 인지 방해
- 서비스 통일감을 해치고 산만한 느낌을 줌

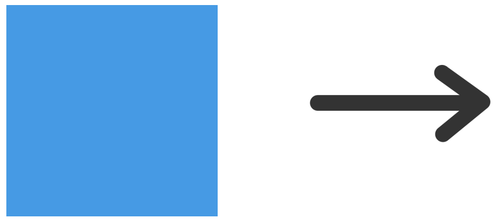

이것은 경고인가 에러인가?

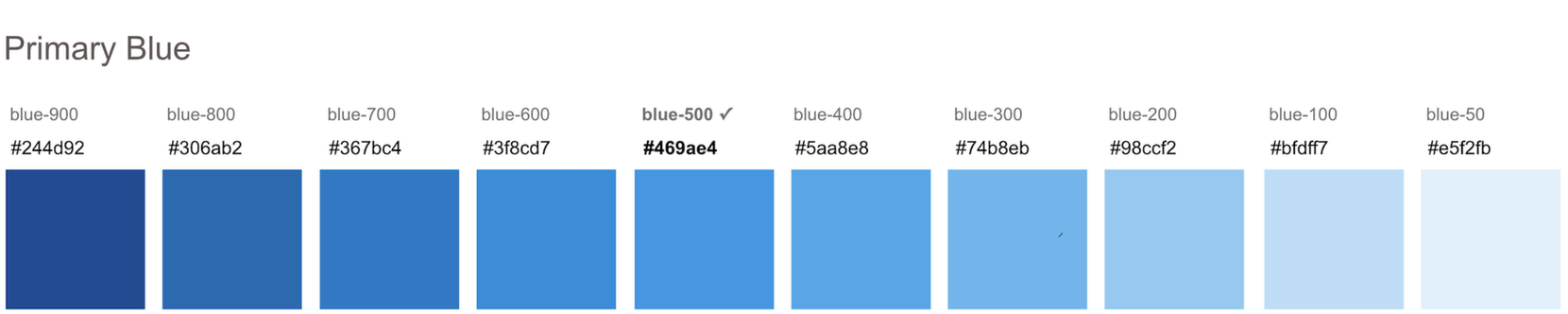
# Color System을 적용하자

DEVIEW 2021

- 말 그대로 서비스의 색상을 체계(System)화
- 서비스 디자인에 최적화된 컬러 팔레트(스펙트럼) 채용
- 서비스의 중심색(Identity)를 이외에 상태(성공, 실패, 경고)를 나타내는 색 포함
- 채용된 컬러 팔레트는 가시성(채도,명도의 적절한 차이)이 있어야 함

PINPOINT





# Color System을 적용하자

DEVIEW 2021

```css
body {
  /* primary blue */
  --blue-50: #e5f2fb;
  --blue-100: #bfdff7;
  --blue-200: #98ccf2;
  --blue-300: #74b8eb;
  --blue-400: #5aa8e8;
  --blue-500: #469ae4;
  --blue-600: #3f8cd7;
  --blue-700: #367bc4;
  --blue-800: #306ab2;
  --blue-900: #244d92;
}
```

```css
body {
 --primary: var(--blue-500);
}

.button-primary {
  background-color: var(--primary)
}
```

CSS Variable 사용

# 난관..대규모 프로젝트에서의 Color System 적용

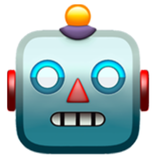
DEVIEW 2021

- 200개가 넘는 css 파일과 코드
- 150개의 색상이 600번 이상 하드코딩
- 무엇을 기준으로 컬러 팔레트 생성?
- 기존에 쓰인 컬러들을 새로운 컬러 팔레트에 어떻게 매칭?

```
ls -lR ./web/src/main/angular/src/**/*.css | wc -l

205
```

# Solution: 컬러 매칭 자동화





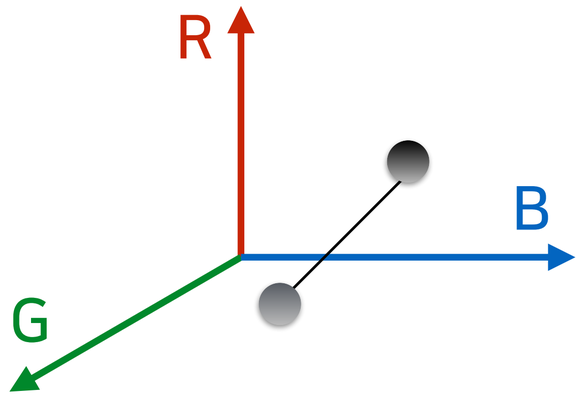
프로그래밍적 컬러 매칭

팔레트 생성

낮은 거리 기준에 따라

팔레트에
컬러 매칭

- 기준 컬러 RGB(팔레트 상의 컬러) 각각의 값을 X,Y,Z 좌표로 보고 비교 대상 RGB 값의 거리 차이 비교
- 낮은 거리의 컬러와 매칭
- 0 ~ 441 까지의 거리가 생김




$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

# 프로그래밍적 매칭의 한계

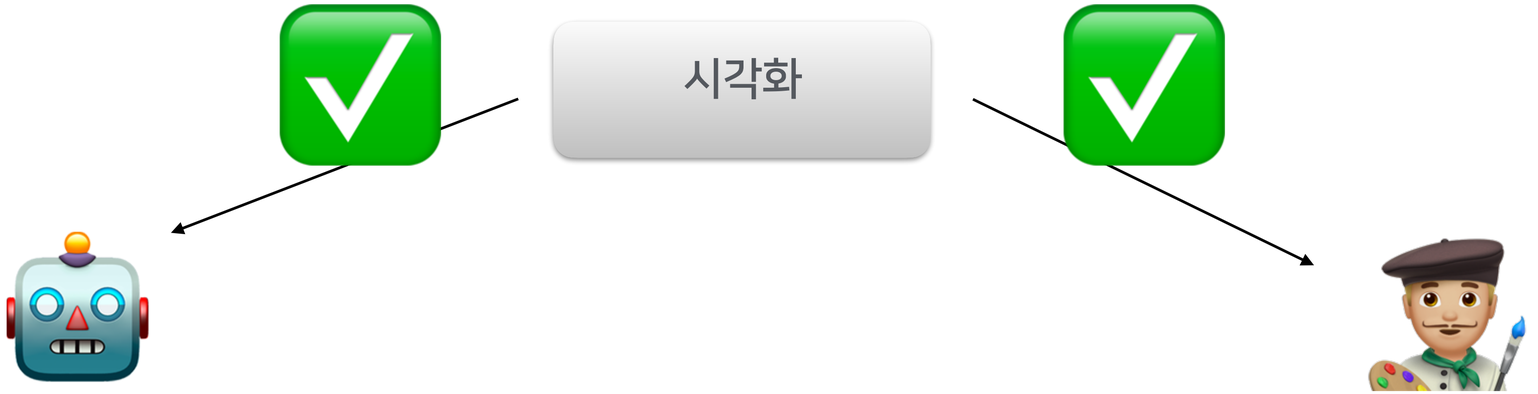
DEVIEW 2021

```
/*  스팩트럼    기존컬러     거리(점수)  */
--grey-100 #E9EBEC 6.164414002968976
--grey-200 #E5E8EA 8.831760866327848
--grey-50 #F8FAFC 5.916079783099616
--grey-50 #F6FAFE 7.681145747868608
--grey-100 #EAEEF4 7.211102550927978
--grey-50 #F6F8FB 4.242640687119285
--grey-50 #F1F3F7 7.211102550927978
--grey-50 #F8F9FB 4.58257569495584
--grey-50 #F9FAFC 6.164414002968976
--grey-400 #A8ACB5 9.848857801796104
--grey-200 #DBDEE6 9.899494936611665
--grey-100 #E7E8EC 9.433981132056603
--grey-50 #FAFAFC 6.557438524302
--grey-50 #F5F5F5 3.4641016151377544
--grey-50 #F3F3F3 6.928203230275509
--grey-100 #EEE 0
--grey-700 #565656 8.660254037844387
--grey-800 #333 0
```


- 거리를 짧게 잡으면 매칭되는 컬러가 적어짐
- 거리가 길어지면 미스 매칭
- 적절한 기준점을 찾기 어려움

# Solution: 도구의 도움을 받아 직접 컬러매칭

DEVIEW 2021




## 프로그래밍적 컬러 매칭

팔레트 생성

↓

팔레트에
컬러 매칭

+

## 직접 컬러 매칭

팔레트 생성

↓

눈으로 보고
직접 매칭

# 시각화 도구 활용

DEVIEW 2021

## 🧑‍🎨 Color collector Report

### hex Color

spectrums

details

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #EB4747 1 | #F06C6B 1 | #C70606 1 | #C10404 1 | #D00 3 | #F00 7 | #FFF1F1 3 | #A94442 6 | #D9534F 1 | #F3E0DF 1 | #F78178 1 | #F27D70 1 | #C04E3F 1 | #E76F4B 1 | #FEA63E 1 | #FF8C00 1 | #F7A626 1 | #BDB76B 1 | #FFF0 1 | #E4F5E3 3 |
| #5CB85C 1 | #03A203 1 | #3C763D 1 | #42A948 3 | #E4F3EB 11 | #9DD4BB 1 | #B5E6D5 1 | #50CBA4 1 | #34B994 1 | #33B692 6 | #3DCFA8 3 | #38C2A2 2 | #00CCFF 1 | #E7F5F9 1 | #4AB0D2 1 | #B4DAE9 1 | #C1ECFF 1 | #E9EBEC 1 | #51AFDF 1 | #E5E8EA 1 |
| #428BCA 2 | #9EC2E4 1 | #4B99E3 2 | #777879 3 | #495057 1 | #566370 1 | #74879A 1 | #E0E7EE 1 | #F8FAFC 1 | #3E83C6 1 | #4A8FD2 3 | #C9DEF3 1 | #F6FAFE 1 | #D0D7DF 1 | #D7DDE4 16 | #8A939E 2 | #CFD7E1 4 | #EDF2F8 13 | #B4CCE9 1 | #BDC7D5 1 |
| #D0D7E1 2 | #D3DBE6 2 | #8F9CAF 1 | #506078 2 | #7185A2 1 | #3E506B 4 | #EAEEF4 5 | #F6F8FB 29 | #4C5C75 2 | #364660 2 | #303F59 1 | #B3B5B9 3 | #E6E8EC 10 | #F1F3F7 1 | #F8F9FB 3 | #F9FAFC 5 | #A8ACB5 15 | #DBDEE6 1 | #E5E8F0 37 | #B3B6BF 6 |
| #E7E8EC 1 | #B3B3B4 10 | #FAFAFC 1 | #609 1 | #D65B65 1 | #E95459 5 | #FFF 83 | #FFFFFF 4 | #F5F5F5 4 | #F3F3F3 1 | #EEE 10 | #E5E5E5 2 | #DDD 6 | #DFDFDF 1 | #D3D3D3 2 | #D0D0D0 1 | #CCC 16 | #CCCCCC 1 | #BEBEBE 3 | #BBB 2 |
| #999 9 | #979797 2 | #808080 3 | #757575 1 | #666 20 | #565656 1 | #333 40 | 2 | 1 | | | | | | | | | | | |

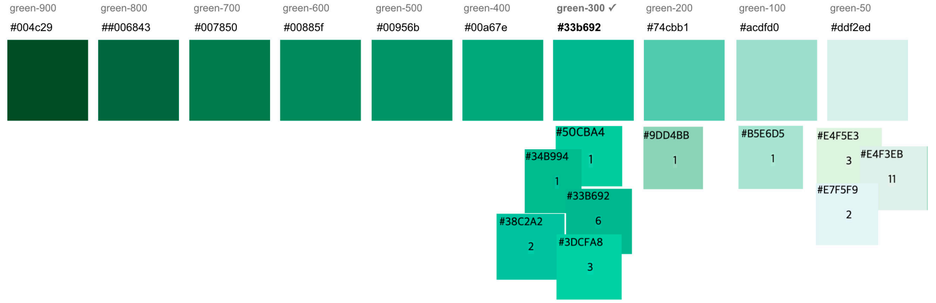
### rgb Color

spectrums

details

rgb(91, 192, 222) 1

- 모든 CSS 확장자 파일에서 컬러 코드들 (Hex, RGB, RGBA)을 정규표현식으로 추출 및 시각화
- 사용 횟수 및 사용된 파일 확인가능



# 적용 단계 설정 및 컬러 사용 실태 시각화 도구 개발



1. 메인 컬러 지정하기
2. 컬러 팔레트 생성하기
3. 컬러 팔레트에 기존 컬러코드들 매칭하기
4. 변수화 및 Semantic 하게 네이밍하기

+@ 변경하고 확인하고 고치기

# Step 1. 메인 컬러 지정하기

DEVIEW 2021

- 메인컬러를 바탕으로 컬러 팔레트 생성
- 사용 빈도수가 높거나 혹은 UI 에서  가장 많이 노출되는 위주로 메인 컬러 지정

👑 🙅

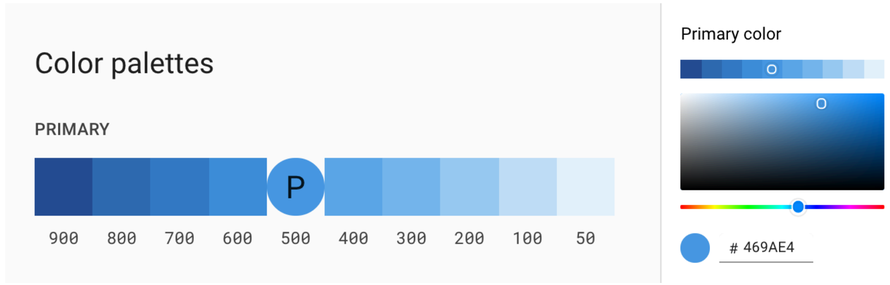
| #EB4747 | #f06c6b | #c70606 | #c10404 | #d00 | #F00 | #a94442 | #D9534F |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 3 | 7 | 6 | 1 |

| red-900 | red-800 | red-700 | red-600 | red-500 | **red-400 ✓** | red-300 | red-200 | red-100 | red-50 |
|---|---|---|---|---|---|---|---|---|---|
| #b20011 | #c2101e | #ce1b25 | #e0272b | #ef322b | **#eb4747** | #e26b6b | #ed9494 | #fecacf | #ffeaed |

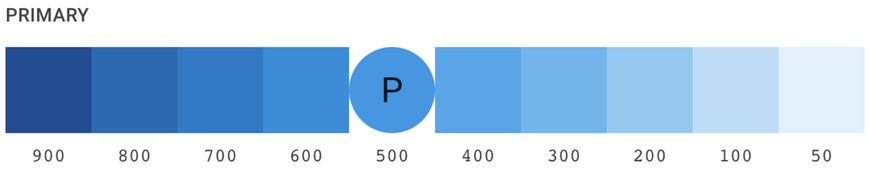
# Step 2. 컬러 팔레트 생성하기

DEVIEW 2021

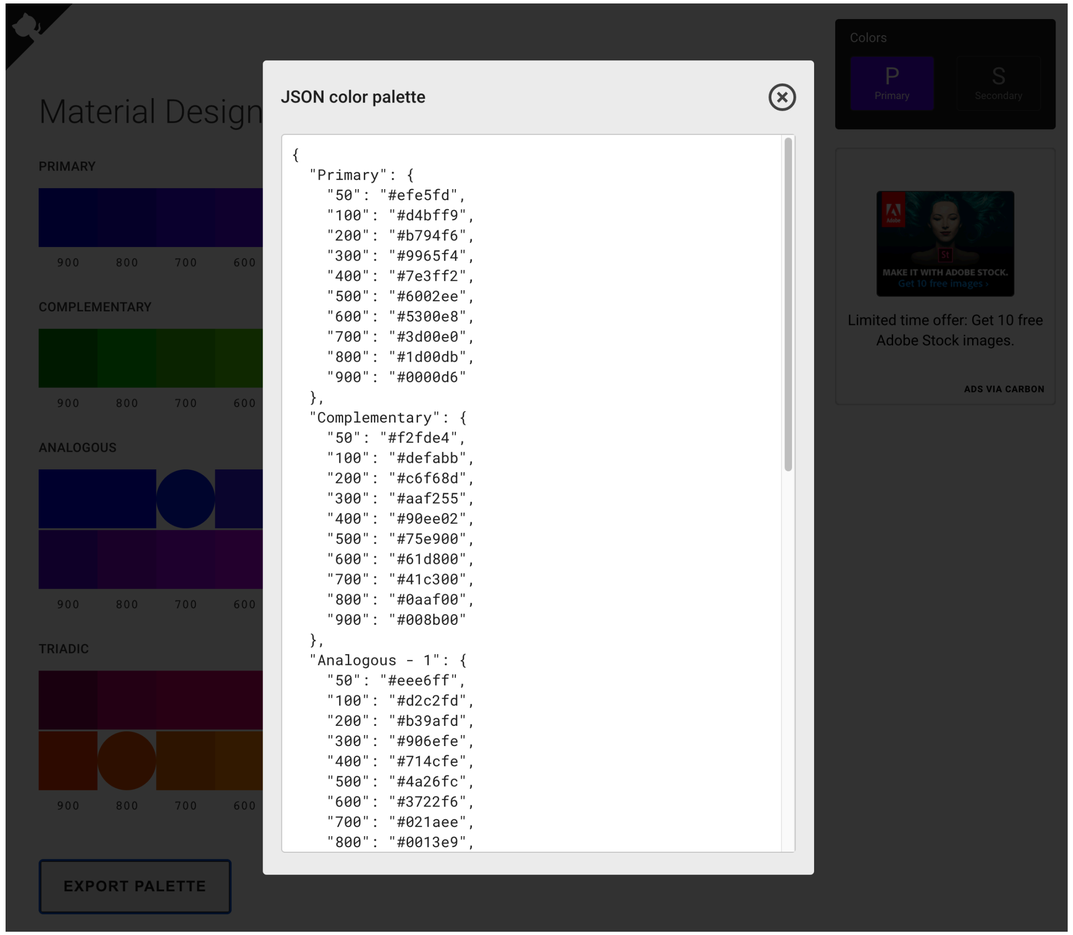
- 팔레트 생성 도구: https://github.com/edelstone/material-palette-generator
- Google Material palette generator를 fork한 도구
- JSON type export 지원



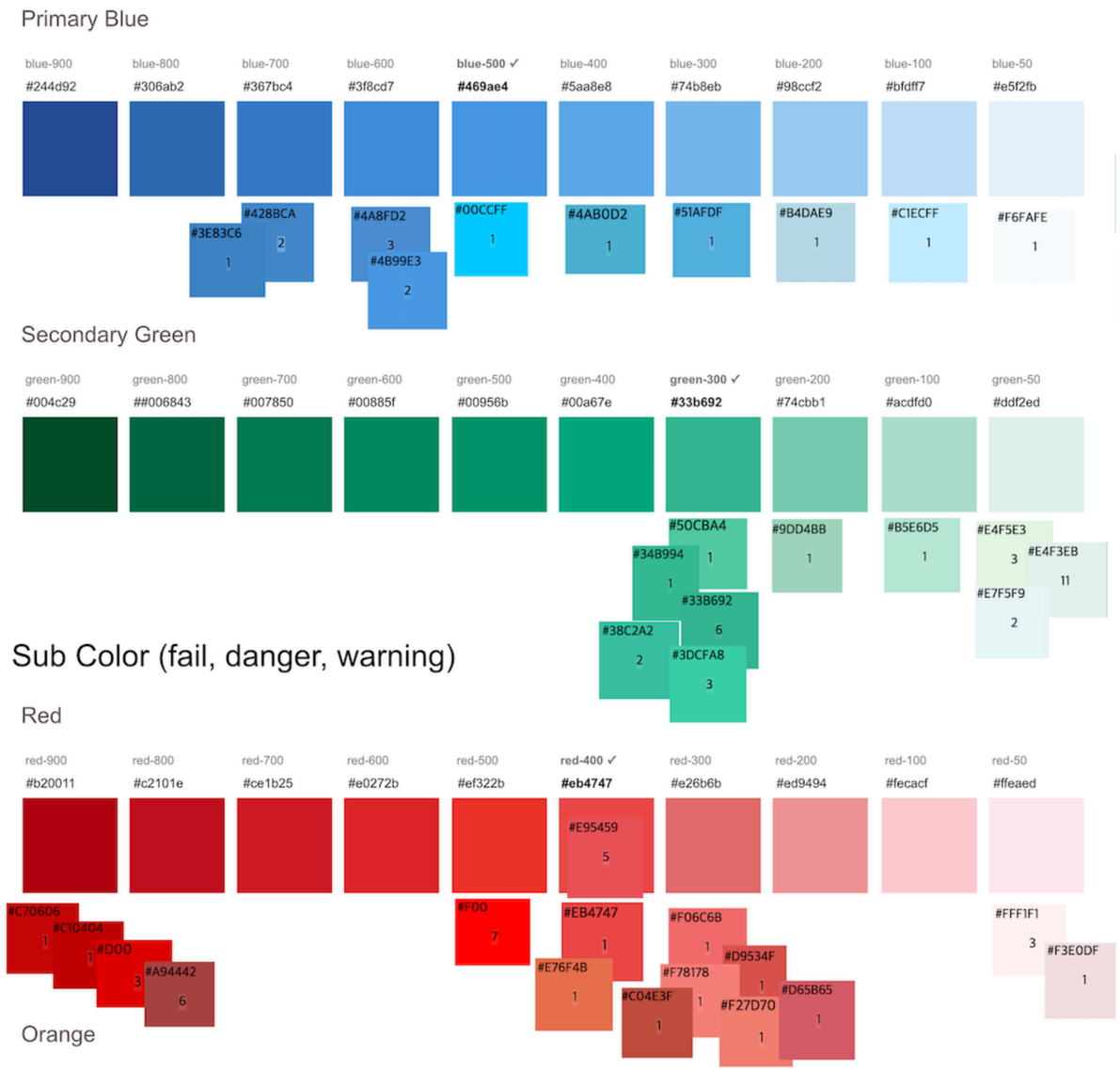

https://material.io/design/color/the-color-system.html#tools-for-picking-colors




https://github.com/edelstone/material-palette-generator



# Step 3. 컬러 팔레트에 기존 컬러 매칭하기




- 시각화 도구를 활용해 기존 컬러를 팔레트에 직접 매칭
- 매칭이 어렵다면 2번단계로 돌아가 다시 컬러 팔레트 생성 및 3번 재 실행
- 프로그래밍적 매칭 기준 거리에 벗어 나지 않는 범위 안에서

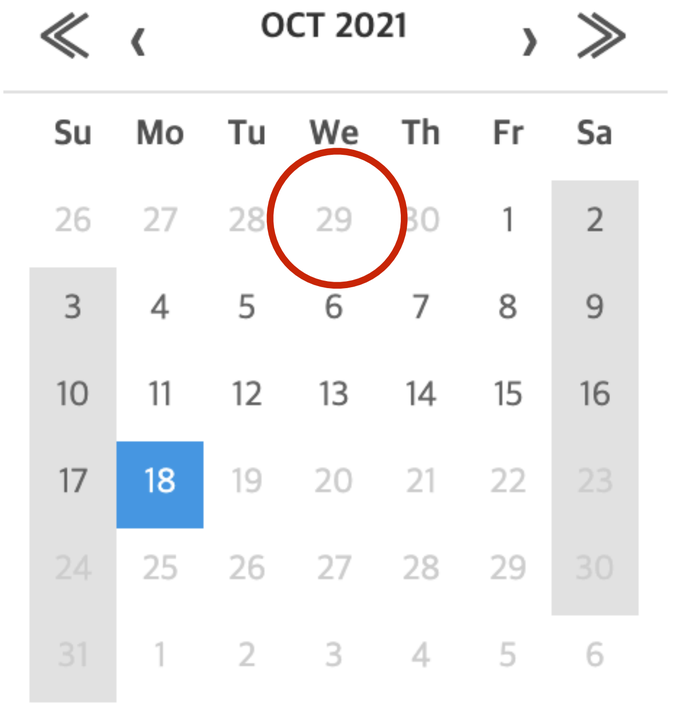
# Step 4. 변수화 및 Semantic 하게 네이밍하기

DEVIEW 2021

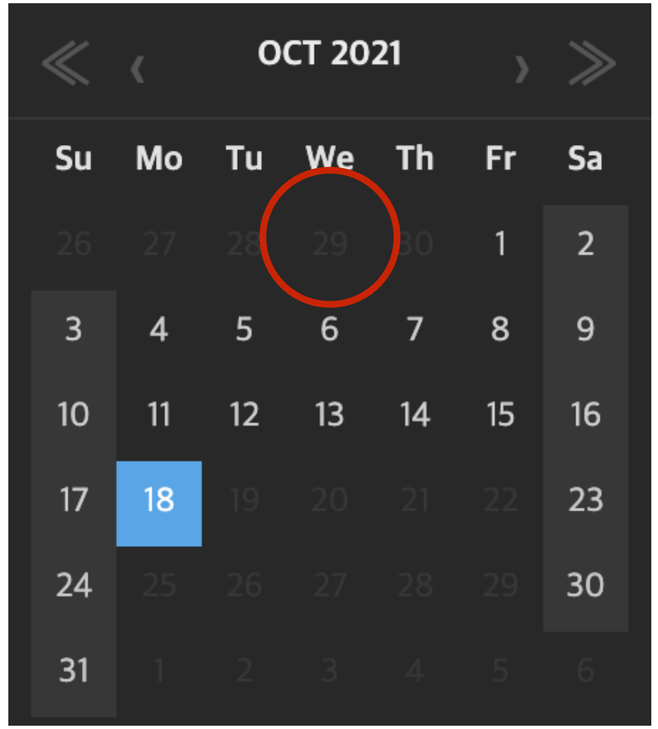
```css
body {
  ...
  --grey-200: #E3E3E3;
  --grey-300: #D1D1D1;
  ...
}


.day.disable {
  color: var(--grey-200);
}
```

# Step 4. 변수화 및 Semantic 하게 네이밍하기

DEVIEW 2021




```
body {
  ...
  --grey-200: #E3E3E3;
  --grey-300: #D1D1D1;
  ...
}

body.dark {
  ...
  --grey-200: #515151;
  --grey-300: #646464;
  ...
}

.day.disable {
  color: var(--grey-200);
}
```




—grey-200 컬러가 다크모드에선 잘 식별이 되지 않음

# Step 4. 변수화 및 Semantic 하게 네이밍하기

DEVIEW 2021

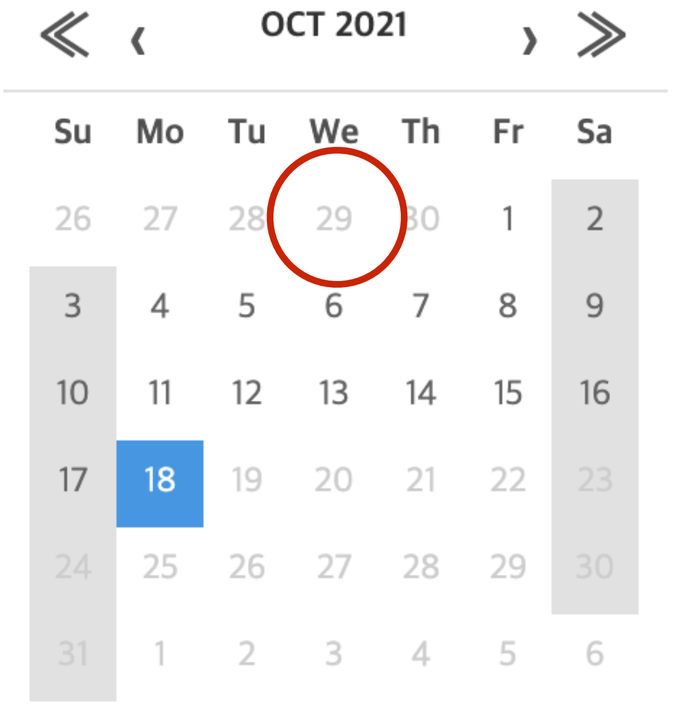
- 의미론적 작명법
- 값이아닌 목적에 따라 명명
- 다크모드 등 테마에 따라 1대1 매칭이 되지 않기 때문
- 테마를 지원하지 않더라도 좀 더 효율적으로 스타일 관리 가능

```
body {
  ...
  --text-default: var(--black-default);
  --text-primary: var(--grey-800);
  --text-primary-lighter: var(--grey-700);
  --text-primary-lightest: var(--grey-600);
  --text-secondary: var(--grey-500);
  --text-secondary-lighter: var(--grey-400);
  --text-secondary-lightest: var(--grey-300);
  --text-knockout: var(--white-default);
  --text-shadow: var(--grey-100);
  --text-disable: var(--grey-300);
  ...
}
```

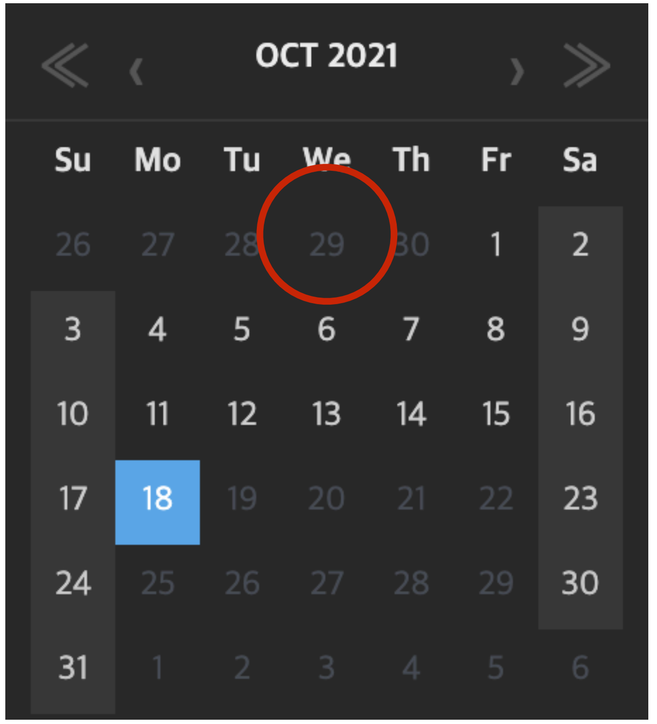
Semantic 네이밍의 예

# Step 4. 변수화 및 Semantic 하게 네이밍하기

DEVIEW 2021



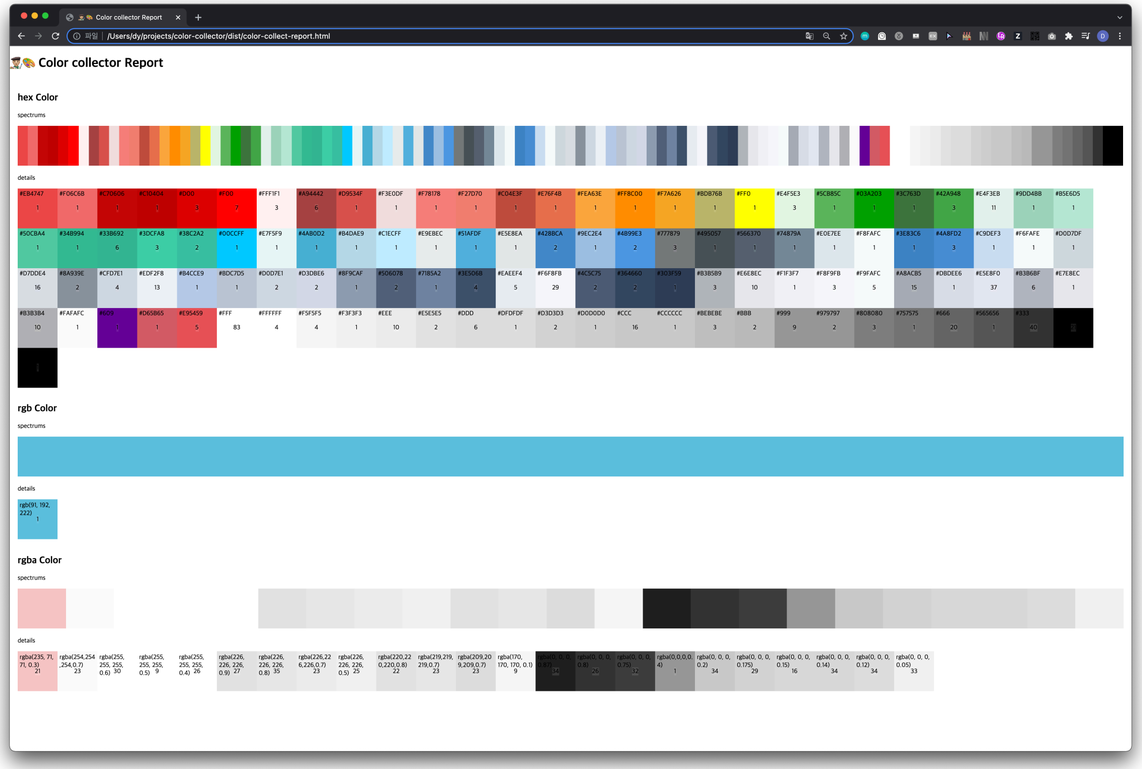

```css
body {
  ...
  --grey-200: #E3E3E3;
  --grey-300: #D1D1D1;
  ...
  --text-disable: var(--grey-200);
}

body.dark {
  ...
  --grey-200: #515151;
  --grey-300: #646464;
  ...
  --text-disable: var(--grey-300);
}

.day.disable {
  color: var(--text-disable);
}
```



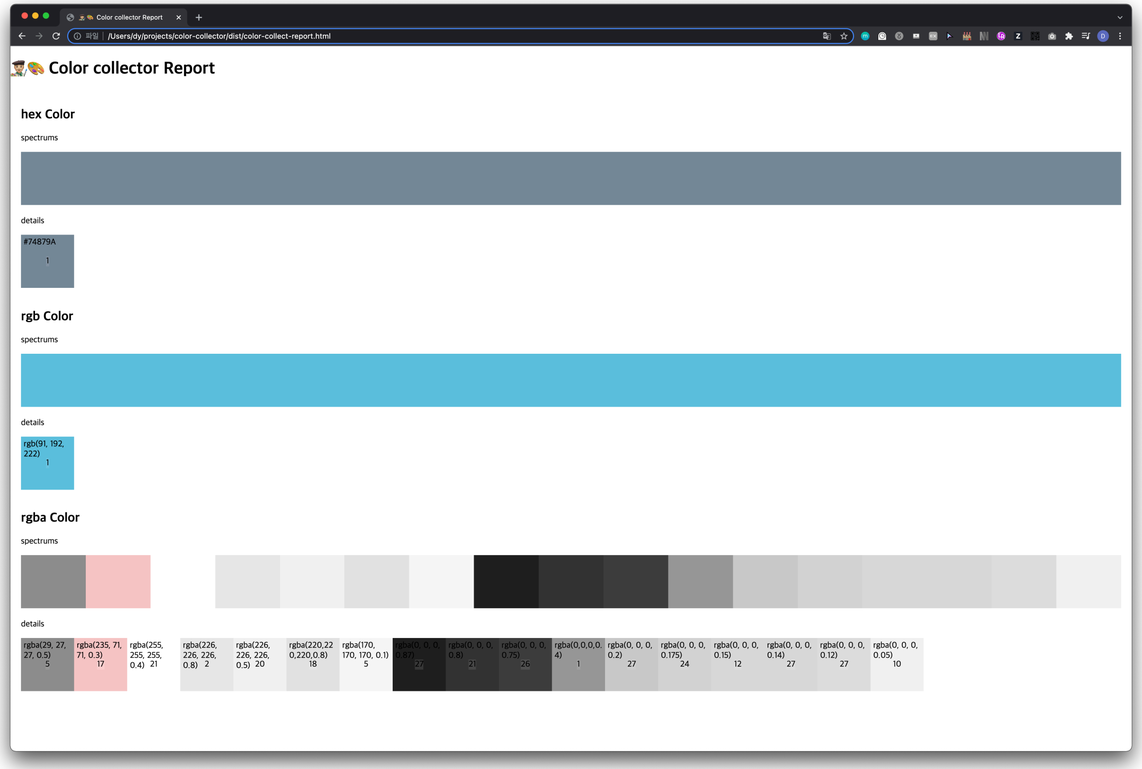
# 개발자도 Color System 안전하게 적용 할 수 있어!

DEVIEW 2021

```
[nodemon] starting `ts-node main.ts`
collect color done.. (211 files)
Complete report creation..
```




적용 전 600여개의 하드코딩된 컬러




적용 후 19개의 하드코딩된 컬러

# 적용 결과

DEVIEW 2021

- 개발 생산성 향상(더 이상의 컬러 고민은 없다!)
- 서비스의 컬러 통일감 높임
- 다크모드도 쉽게 적용
- 디자인 시스템을 위한 기반 마련

# Thank You