

RENOBIT Tutorial (초안)

❶ ✓ 왜 모든 웹 기술을 다루지 않아도 되는가?

RENOBIT은 “웹 기반 시각화 저작 도구”

- 핵심은 시각화를 구성하고 작동시키는 과정이지, HTML/CSS/JS의 모든 사양을 배우는 게 아님
- 사용자가 익혀야 할 건 “RENOBIT이 어떻게 이 기술을 활용하는가?”이지 “웹 기술의 모든 것”이 아님

모든 걸 가르치려다 중심이 흐려질 수 있다

- 핵심 메시지: “RENOBIT을 쓰면 시각화를 만들 수 있다”
- 거기에 너무 많은 이론을 끼워 넣으면 학습자는 길을 잃음

이 문서는 웹 기술을 기반으로 하지만, 웹 교육 전체를 책임지지는 않습니다.

대신, 도구의 핵심 흐름을 중심으로 설명하면서,

기술적으로 연결되는 지점은 외부 문서로 안내하는 것이 가장 이상적인 학습 구조입니다.

▼ Introduction

RENOBIT은 시각화 도구입니다.

그렇다면 ‘시각화한다’는 것은 어떤 의미일까요? 이 개념에 대해 조금 더 자세히 이야기해보겠습니다.

무언가를 ‘본다’는 행위는 단순히 눈으로 보는 것을 넘어서, 다음의 요소들을 내포하고 있습니다:

1. **무엇을 통해서 볼 것인가** - Environment → Web browser
2. **무엇을 어디에 둘 것인가** – Placement → Layout
3. **보는 대상은 어떻게 구성되어 있는가** – Structuring → HTML (Structure), CSS (Style)"
4. **보는 대상은 무엇을 보여주는가** – Information → Data
5. **보는 대상이 변화할 수 있는가** - Interaction → JavaScript

이 다섯 가지 조건이 충족되어야 비로소 우리는 어떤 대상을 ‘시각화’할 수 있습니다.

RENOBIT은 이 모든 과정을 하나의 흐름으로 연결하고, 통합적으로 지원하는 시각화 도구입니다.

1. Environment – 무엇을 통해 볼 것인가

시각화는 정보를 단순히 데이터로 존재하게 두는 것이 아니라, 사용자가 시각적으로 인지할 수 있도록 표현하는 과정입니다.

이러한 표현이 이루어지기 위해서는 반드시 ‘보여지는 환경’, 즉 시각화가 실행되고 전달되는 플랫폼이 필요합니다.

웹 브라우저라는 시각화 환경

RENOBIT은 웹 브라우저를 기반으로 동작하는 시각화 도구입니다.

즉, 사용자가 구성한 시각화는 웹 브라우저 위에서 실행되며, 모든 구조와 데이터, 상호작용이 이 환경 안에서 구현됩니다.

웹 브라우저는 시각화에서 다음과 같은 중요한 역할을 합니다:

- **출력 매체:** HTML, CSS, JavaScript로 구성된 구조와 데이터를 눈에 보이도록 렌더링합니다.
- **표준 기반 플랫폼:** 다양한 기기와 운영체제에서도 일관된 방식으로 동작하며, 시각화 결과물을 공통된 방식으로 전달할 수 있게 합니다.
- **인터랙션의 장:** 사용자 입력(마우스, 터치, 키보드 등)에 대한 반응이 실시간으로 이루어져, 동적인 시각화가 가능해집니다.

왜 브라우저 기반인가?

시각화 도구는 다음과 같은 이유로 브라우저 기반 환경을 선택할 때 강력한 유연성을 확보할 수 있습니다:

- 설치가 필요 없는 접근성: 사용자는 웹 브라우저만 있으면 언제 어디서든 시각화에 접근하고 실행할 수 있습니다.
- 표준 기술에 기반한 확장성: HTML, CSS, JavaScript 등의 웹 표준 기술을 활용하여 다양한 형태의 시각화를 구성할 수 있습니다.
- 다양한 표현 수단과 연동 가능: Canvas, SVG, WebGL 등의 기술을 통해 2D부터 3D까지 다양한 표현 방식이 가능해집니다.

요약

Environment는 시각화의 출발점이며,

RENOBIT은 웹 브라우저라는 범용적이고 확장 가능한 환경을 기반으로 모든 시각화를 구성합니다.

이 환경 위에서 Placement, Structuring, Information, Interaction의 요소들이 차례로 쌓이며,

사용자는 복잡한 정보를 직관적으로 전달할 수 있게 됩니다.

2. Placement – 무엇을 어디에 들 것인가

시각화는 단순히 ‘무언가를 보이게 하는 것’이 아닙니다.

어디에, 어떻게 배치되었는가에 따라 정보의 의미와 전달 방식은 전혀 달라집니다.

요소의 위치, 크기, 간격, 정렬 방식은 시각화의 가독성과 효과성을 좌우하며,

이 모든 것을 정의하는 것이 바로 **배치(Placement)**입니다.

박스(Box)를 기반으로 한 구성

RENOBIT에서 사용자는 화면에 배치되는 모든 요소를 ‘박스(Box)’ 형태, 즉 HTML의 `div` 요소 단위로 다루게 됩니다.

텍스트, 이미지, 차트, 버튼, 3D 객체 등 다양한 콘텐츠는 모두 이 박스를 통해 시각적으로 표현됩니다.

이러한 박스 요소는 웹 표준인 **CSS Box Model**을 따르며, 기본적으로 다음과 같은 배치 속성을 가집니다:

- 위치 (Position)
- 크기 (Width, Height)
- 간격 (Margin, Padding)
- 정렬 (Flex, Grid 등)



참고: 웹에서의 Box Model 개념은 [MDN Box Model 문서](#)를 통해 확인할 수 있습니다.

모든 시각적 요소는 content → padding → border → margin의 구조로 이루어져 있으며,

이것이 RENOBIT의 시각적 배치 시스템의 이론적 기반이 됩니다.

Drag & Drop을 통한 직관적인 배치

RENOBIT의 가장 큰 장점 중 하나는 시각적 배치를 **Drag & Drop** 방식으로 수행할 수 있다는 점입니다.

사용자는 도구 패널에서 원하는 요소를 선택한 뒤, 페이지 상에 끌어다 놓음으로써 간편하게 배치 할 수 있습니다.

배치된 요소는 클릭 시 활성화되며, 이후 위치, 크기, 간격, 정렬 방식을 시각적 속성 패널을 통해 조정할 수 있습니다.

시각적 편집 vs. 코드 기반 편집

RENOBIT은 시각적 도구와 함께 코드 에디터도 내장하고 있어, HTML/CSS/JavaScript를 직접 편집할 수 있습니다.

사용자는 다음 두 가지 편집 방식 중 자유롭게 선택할 수 있습니다:

- **시각적 편집 패널:** 빠르고 직관적인 배치와 속성 조정
- **코드 편집기:** 세밀한 제어와 커스터마이징

이러한 **이중 편집 구조**는 초보자에게는 쉬운 접근성을,
고급 사용자에게는 세밀한 제어와 자유도를 동시에 제공합니다.

요약

RENOBIT의 Placement는 웹 표준 Box Model에 기반하여
박스 형태의 요소를 Drag & Drop으로 배치하고,
속성 패널 또는 코드 에디터를 통해 자유롭게 제어할 수 있도록 설계되어 있습니다.
이는 직관성과 정밀함을 동시에 갖춘 시각적 구성 방식을 가능하게 합니다.

3. Structuring – 보는 대상은 어떻게 구성되는가

시각화에서 대상이 단순히 존재하는 것만으로는 충분하지 않습니다.

우리가 정보를 인식하고 해석할 수 있으려면, 그 대상은 **구조화(structuring)** 되어 있어야 합니다.
즉, 무엇이 어디에 있고, 어떤 형태로 존재하는지를 정의하는 것이 시각화의 핵심입니다.

HTML과 CSS 기반의 시각 구조

RENOBIT에서 화면에 표현되는 모든 요소는 **HTML과 CSS**를 통해 구성됩니다.

- HTML은 시각화 대상의 **구조(Structure)**를 정의합니다.
- CSS는 구조화된 요소에 **스타일(Style)**을 부여하여, 시각적으로 명확하게 드러나도록 만듭니다.

사용자는 코드 에디터를 통해 직접 HTML/CSS를 수정할 수 있으며,
기본적으로 제공되는 Bootstrap 기반의 컴포넌트를 활용하여 빠르게 구조를 형성할 수도 있습니다.

이러한 **구조 + 스타일**의 조합은 우리가 보는 화면을 단순한 데이터 나열이 아닌, 의미 있는 시각적
체계로 만들어줍니다.

리소스(Resource)를 통한 시각 자산 구성

시각적 구조는 단지 코드로만 완성되지 않습니다.

이미지, 3D 모델(GLTF), 아이콘 등 다양한 시각 자산(resource)들이 함께 구성되어야 더 풍부한 시각화가 가능합니다.

RENOBIT은 이러한 자산들을 관리할 수 있도록 **Resource Manager** 기능을 제공합니다.

사용자는 다음과 같은 작업을 수행할 수 있습니다:

- 이미지나 3D 모델 파일(GLTF 등)을 업로드하여 저장
- 각 리소스에 이름을 지정하고 관리
- 필요한 시점에 컴포넌트에서 해당 리소스를 참조하여 구조에 삽입

이러한 방식은 시각적 구조의 재사용성과 일관성을 높이며,

특히 복잡한 3D 시각화나 아이콘 기반의 상태 표현 등에 효과적으로 활용됩니다.

구조화의 목적

‘구조화’는 단순히 예쁘게 배치하는 것이 아니라,

정보의 흐름을 시각적으로 설계하는 것입니다.

- 무엇이 먼저 보여야 하는가
- 어떤 관계가 있는가
- 어떤 구조로 구성되어야 이해가 쉬운가

이러한 질문에 대한 답을 구조로 표현하는 것이 시각화의 진정한 가치이며,

RENOBIT은 이 과정을 코드와 시각적 도구를 통해 동시에 수행할 수 있도록 설계되어 있습니다.

요약

RENOBIT은 HTML과 CSS를 기반으로 시각 구조를 정의하며,

Resource Manager를 통해 이미지나 3D 모델 같은 자산도 함께 구성할 수 있도록 지원합니다.

이로써 사용자는 시각화 대상에 대해 구조적 의미와 시각적 완성도를 동시에 부여할 수 있습니다.

4. Information – 무엇을 보여주는가

시각화에서 가장 중요한 질문은 바로 이것입니다:

“무엇을 보여줄 것인가?”

아무리 잘 구성된 레이아웃과 정교한 스타일을 갖추었더라도,
그 안에 담길 정보가 없다면 시각화는 의미를 잃습니다.

RENOBIT은 데이터를 중심으로 시각화를 구성할 수 있도록,
Dataset Manager와 명확한 데이터 연결 구조를 제공합니다.

데이터셋(Dataset)의 정의와 사용

RENOBIT에서는 데이터를 직접 다루는 대신,
“데이터셋(Dataset)”이라는 단위로 데이터를 정의하고 관리합니다.

사용자는 다음과 같은 방식으로 데이터셋을 만들 수 있습니다:

1. 데이터 소스 지정:

- SQL 쿼리 또는 REST API URL 입력

2. 이름 부여 및 저장:

- 예: `cpuStatusDataset`, `alarmList`, `temperatureTrend`

3. 호출 방식 정의:

- 필요한 파라미터를 함께 정의하여, 실행 시점에 동적으로 호출 가능

이렇게 생성된 데이터셋은, 이를 기반으로 호출하여 시각화에 재사용할 수 있습니다.

→ 마치 *"정보의 API를 정의하고 시각화에서 구독하는 구조"*를 갖는 셈입니다.

컴포넌트와 데이터의 연결 – 매핑 구조

데이터셋이 정의되었다면, 이제 그것을 어떤 시각화 컴포넌트와 연결할 것인지를 결정해야 합니다.

RENOBIT은 이를 위해 **dataMapping** 구조를 제공합니다.

이 구조는 다음을 명시합니다:

- 어떤 컴포넌트가 (`visualInstanceList`)
- 어떤 데이터셋을 사용할지 (`datasetName`)
- 어떤 파라미터를 전달할지 (`param`)

이 방식은 데이터를 단순히 전달하는 것이 아니라,
컴포넌트와 데이터 사이의 관계를 선언적으로 표현하는 방법입니다.

데이터의 재사용성과 일관성

RENOBIT의 데이터 연결 구조는 **하나의 데이터셋을 여러 컴포넌트에서 공유하거나, 동일한 컴포넌트가 다른 파라미터로 여러 번 데이터를 호출할 수 있게 설계되어 있습니다.**

이러한 구조는 다음과 같은 장점을 제공합니다:

- 데이터 소스와 시각화 로직의 **분리**
- 복잡한 대시보드 구성을 위한 **일관성 있는 데이터 흐름**
- 다양한 컴포넌트 간의 **정보 연결성과 응집력 강화**

요약

RENOBIT에서 시각화는 **무엇을 보여주는가**라는 질문에 정면으로 답합니다.

사용자는 **Dataset Manager**를 통해 데이터를 정의하고,

dataMapping 구조를 통해 시각화 컴포넌트와 정확하게 연결할 수 있습니다.

이러한 구조는 정보 중심의 시각화를 가능하게 하며,

RENOBIT이 단순한 화면 편집기를 넘어 **데이터 기반 시각화 도구**로 기능할 수 있게 합니다.

5. Interaction – 대상은 변화할 수 있는가

시각화가 진정한 의미를 가지려면, 보여지는 것만으로는 부족합니다.

사용자의 행동에 반응하고, 데이터 변화에 따라 스스로 갱신되는 시각화는 단순한 화면을 넘어 '**시스템**'이 됩니다.

이러한 반응성과 동작 흐름은 바로 ****인터랙션(Interaction)****으로 구현됩니다.

인터랙션은 코드로 정의된다

RENOBIT에서의 인터랙션은 모든 관계를 코드 에디터(**JavaScript**)를 통해 정의합니다.

즉, "이 버튼을 누르면 어떤 차트를 새로 고친다" 와 같은 상호작용은

사용자가 직접 작성한 자바스크립트 코드에 의해 구현됩니다.

시각화 화면에 배치된 각 요소는 고유한 **인스턴스 이름(instance name)**을 가지며,

이 이름을 통해 컴포넌트 간의 관계를 코드로 지정할 수 있습니다.

```
1 // 예시: 버튼 클릭 시 차트 데이터를 다시 불러오기
2 document.querySelector('#refreshButton').addEventListener('click', () => {
3   const targetChart = getInstanceByName('cpuChart1');
4   targetChart.refresh();
5 });


```

이처럼 인터랙션은 단순한 UI 연결이 아니라,
명확한 목적과 흐름을 가진 자바스크립트 로직으로 표현됩니다.

🧠 인스턴스 이름 기반의 구조적 관계

RENOBIT에서 배치된 모든 컴포넌트는 고유한 **인스턴스 이름**을 가집니다.
이는 인터랙션의 대상과 범위를 명확하게 지정하는 기준이 됩니다.

- `getInstanceByName('chart1')` 처럼 특정 요소를 찾고
- 그 인스턴스에 대해 동작을 수행 (`.show()`, `.refresh()`, `.toggle()` 등)

이러한 명명 기반 구조는 **시각적 요소와 인터랙션 로직을 명확하게 연결**시켜주며,
복잡한 시각화 구성에서도 구조적인 통제를 가능하게 합니다.

✓ 요약

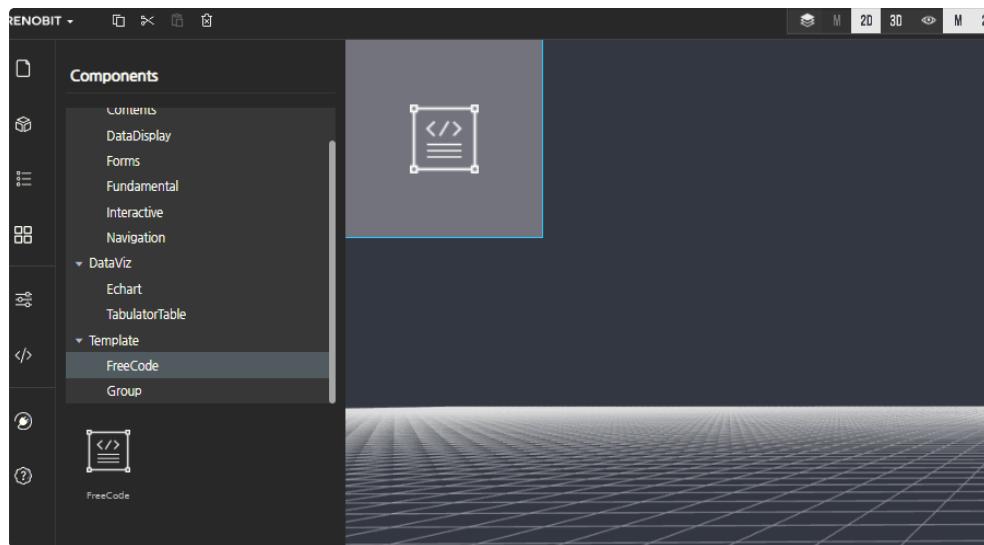
RENOBIT의 인터랙션은 모든 상호작용을 **JavaScript 코드로 정의**합니다.
배치된 컴포넌트는 고유한 인스턴스 이름을 가지며,
이름 기반으로 컴포넌트 간 동작을 연결하고, 사용자 행동에 반응하는 시각화 로직을 구성할 수 있습니다.

이로써 RENOBIT은 정적인 화면 구성에서 나아가,
사용자 주도의 동적인 시각화 설계 도구로 확장됩니다.

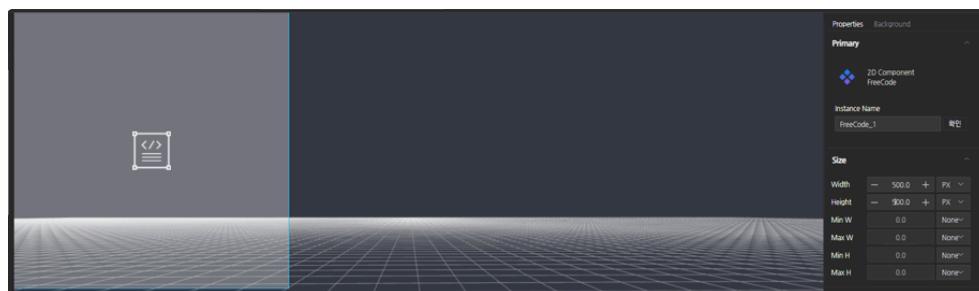
▼ Placement

“페이지에 박스를 배치하고, 위치와 크기를 조절해보세요.”

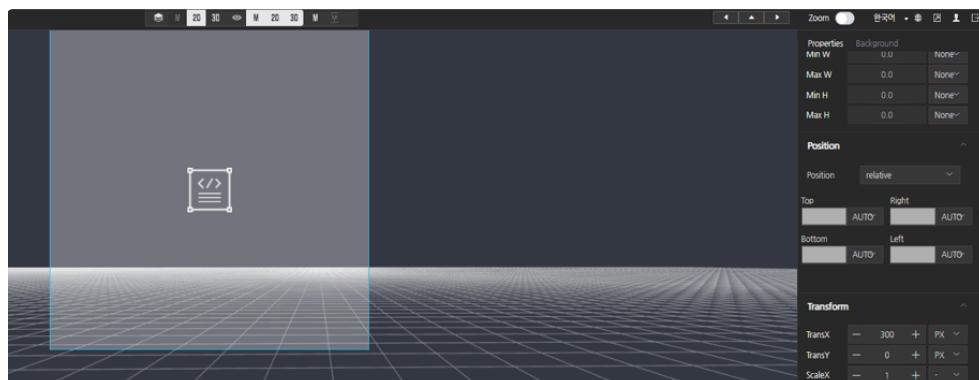
- 왼쪽 Toolbar에서 Component를 배치합니다. (Template > FreeCode)
- Outline으로 현재 컨테이너가 차지하고 있는 공간이 표시됩니다.



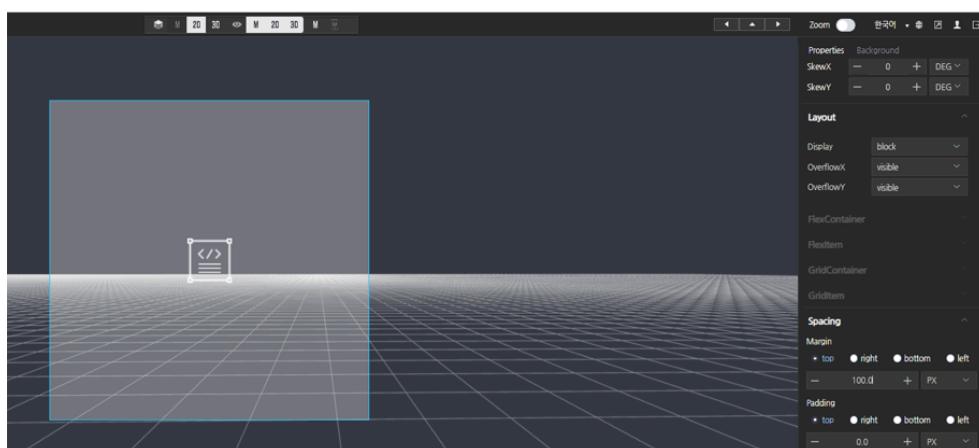
- 이 박스의 크기를 Width 500px, Height 500px로 변경해봅니다.



- 이 박스의 위치를 변경해봅니다 (Transform → TransX 300px)

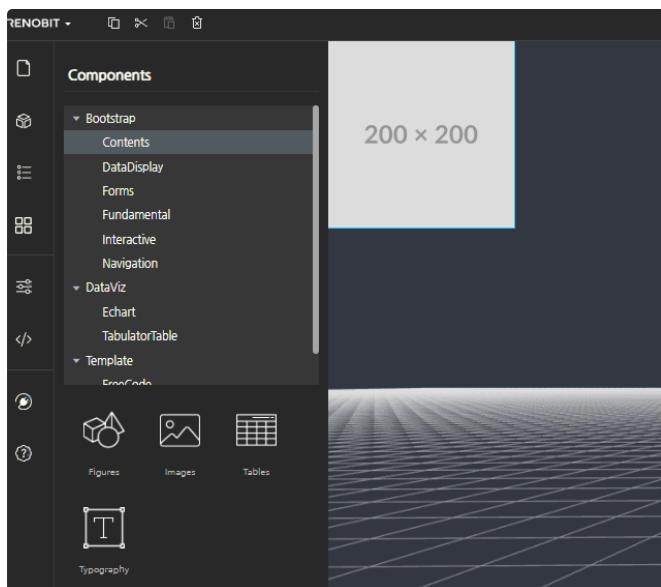


- 박스에 margin-top을 부여해봅니다.

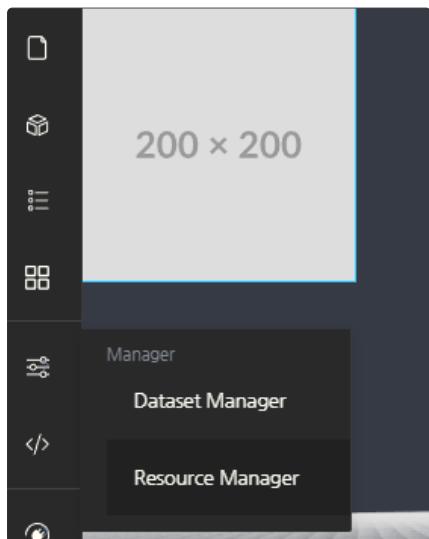


"이미지 컴포넌트를 배치하고 리소스 관리자와 HTML 편집을 통해 이미지를 삽입해보세요"

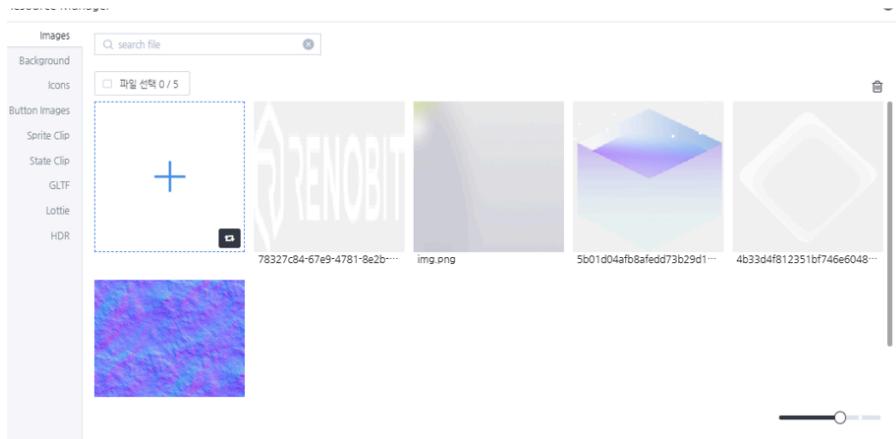
- 왼쪽 Toolbar에서 Component를 배치합니다. (Bootstrap > Contents > Images)



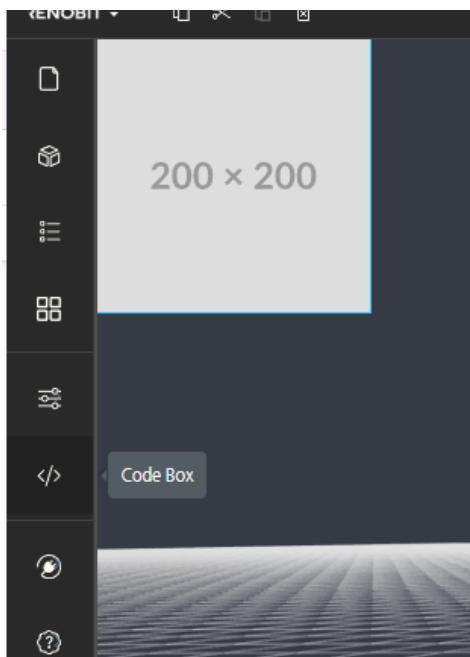
- 왼쪽 Toolbar에서 Resource Manager를 선택합니다.



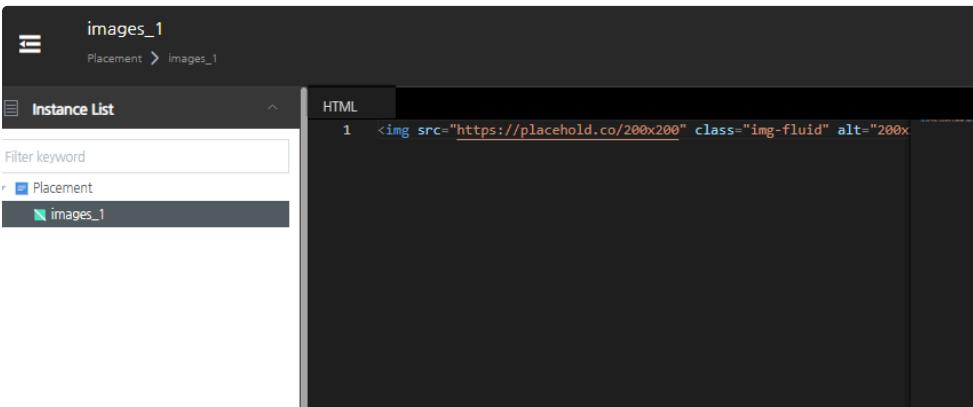
- "+" 영역을 선택하여 원하는 이미지를 업로드 합니다.



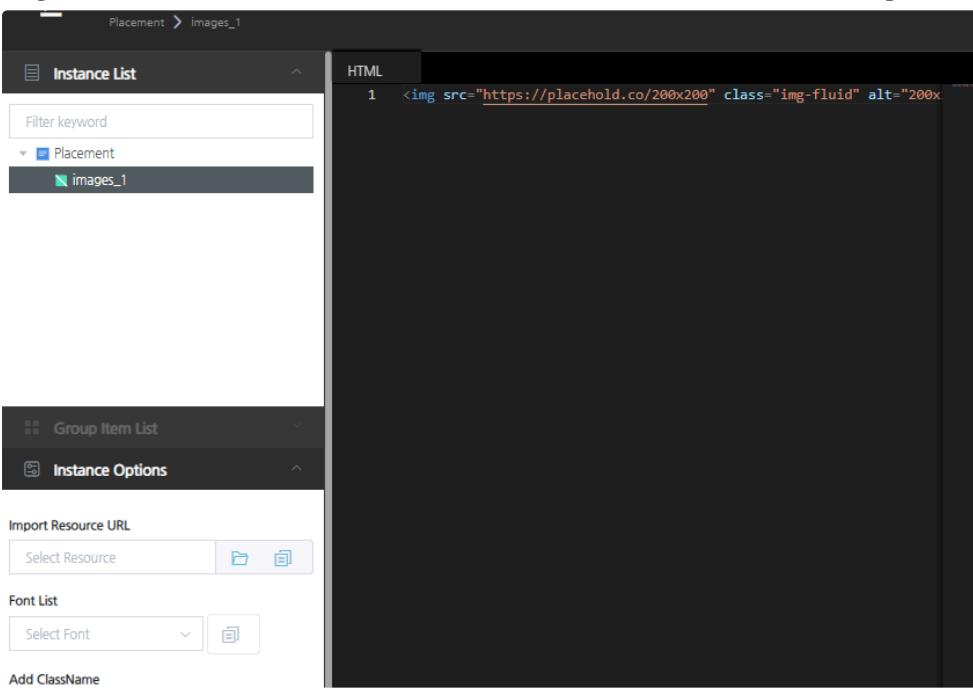
- 이미지 컴포넌트를 선택한 상태에서 Code box를 클릭합니다. 새 탭이 열립니다.



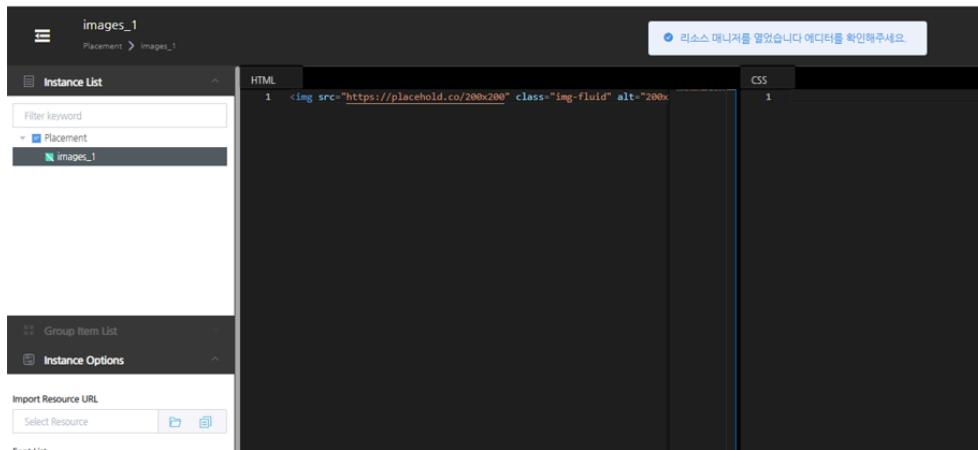
- 열린 창의 HTML 영역을 확인합니다. (컨테이너 역할을 하는 DIV 태그 내부에 포함된 HTML을 의미합니다.)



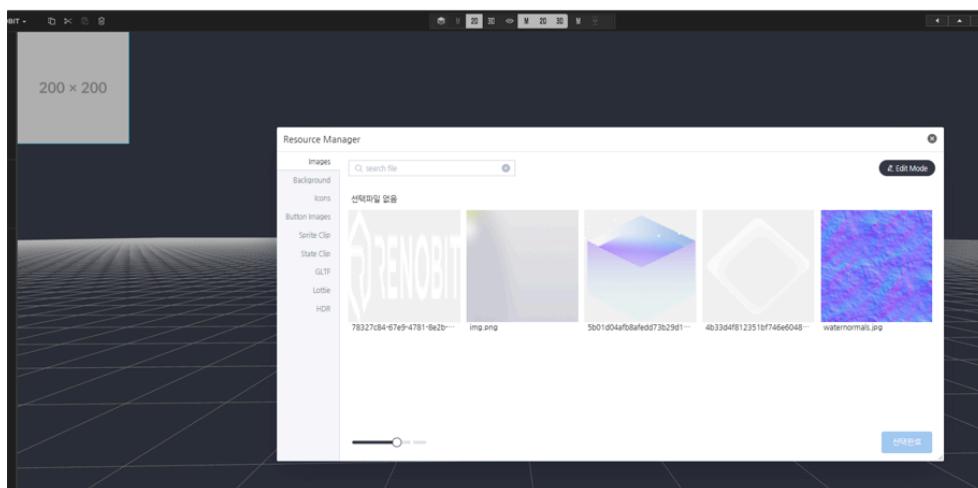
- img의 src를 업로드한 리소스의 주소로 교체합니다. 왼쪽 Instance Options를 클릭합니다.



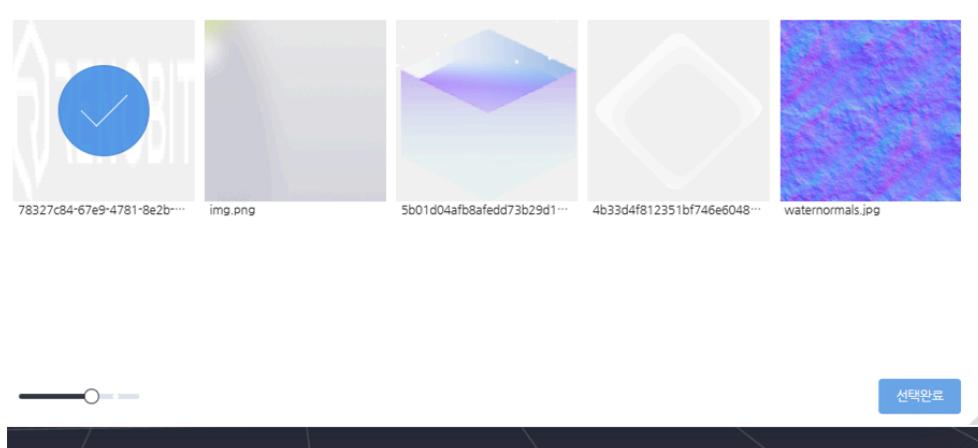
- Import Resource URL 항목 아래 풀더를 눌러서 리소스 매니저를 활성화 합니다.



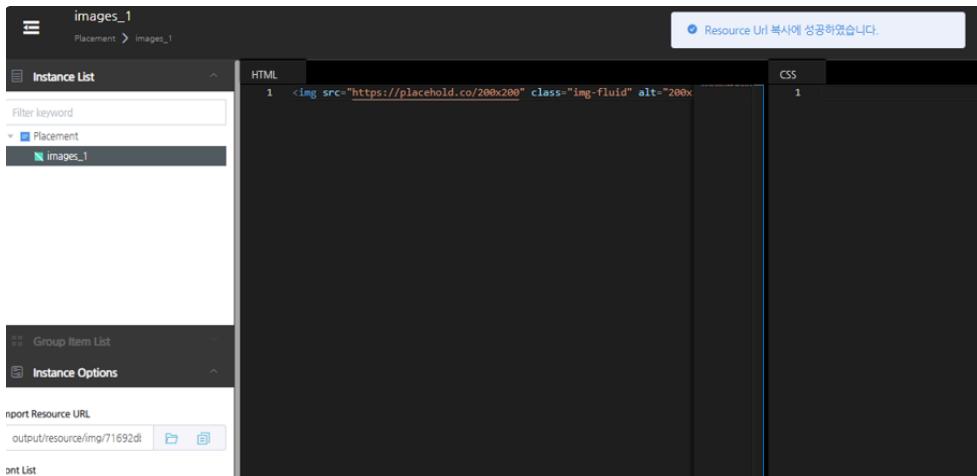
- 메인 에디터에서 리소스 매니저가 열린 것을 확인할 수 있습니다.



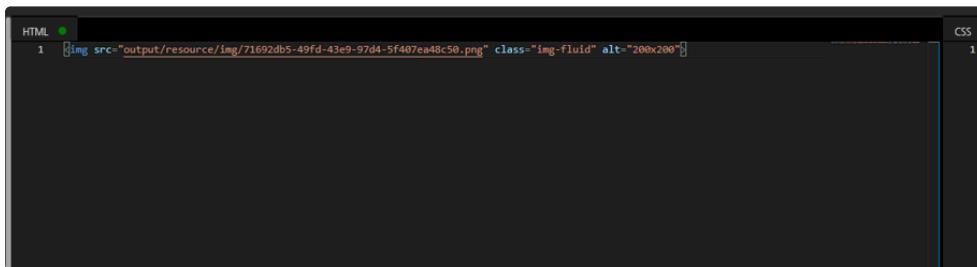
- 리소스를 선택하고 선택 완료 버튼을 누릅니다.



- Import Resource URL 항목 우측 버튼을 눌러서 URL을 Copy 합니다.



- Copy한 URL로 교체합니다.



- Apply 버튼을 클릭하여 HTML을 에디터에 적용합니다



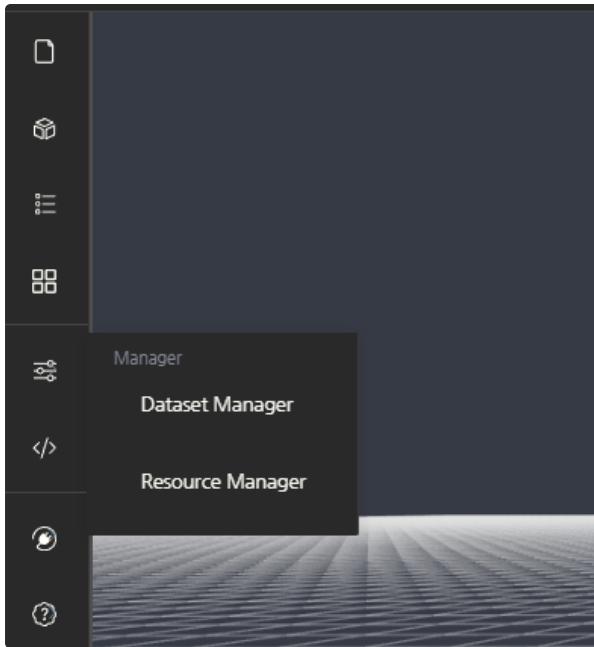
- 메인 에디터에서 변경된 결과를 확인합니다.



Information

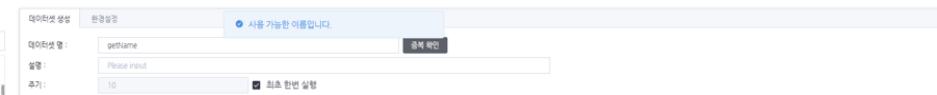
"목록 데이터를 호출하여 테이블에 표시해보세요."

- 데이터셋 매니저를 열어봅니다. 새 창이 열립니다.



- 데이터셋을 생성합니다. getName라는 이름으로 생성해봅니다.

- 중복 확인을 합니다.
- 최초 한번 실행을 체크합니다.



- 여기 예시에서는 HTTP API를 사용하도록 합니다. API 정보를 설정합니다.
 - 변수로 처리해야 할 부분은 → #{} 와 같은 형태로 지정합니다.
 - #{}를 사용할 경우 Create Parameter에서 선택할 수 있습니다
 - 파라미터에 기본값을 입력합니다.



- 미리보기를 통해 데이터를 확인합니다. 미리보기를 통해 데이터를 확인할 수 없는 데이터소스는 데이터셋으로 등록할 수 없습니다.

Credentials

Method :

URL :

Content-Type :

Body

```
[{"id": "1", "name": "John"}]
```

Create Parameter

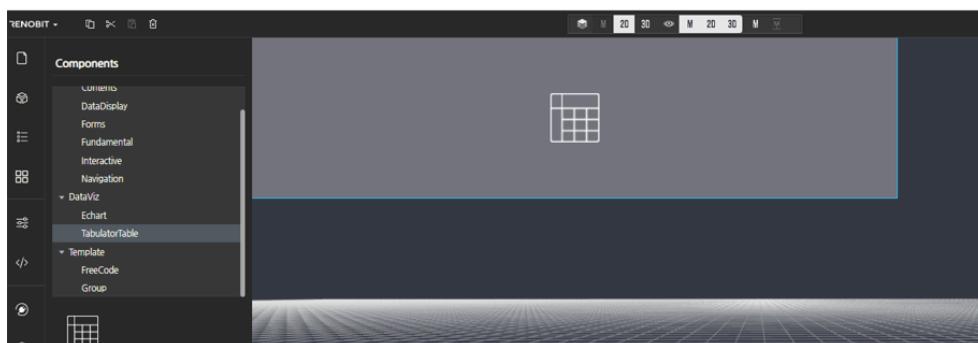
파라미터명 입력	직접 입력	Default	추가
변수명		기본값	
id	<input type="text" value="1"/>		

실행 결과 미리보기

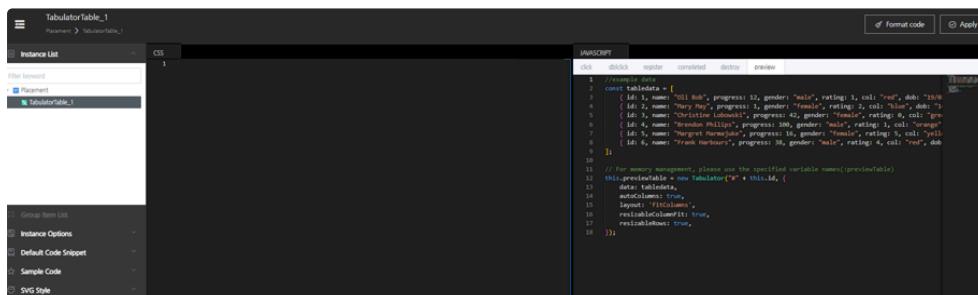
미리보기

```
[{"id": "1", "name": "John"}]
```

- 데이터를 표시할 테이블 컴포넌트를 페이지에 배치합니다. (DataViz > TabulatorTable)



- 컴포넌트를 클릭한 상태에서 Code box를 엽니다.



- 코드박스는 컴포넌트 라이프 사이클 + 유тиль 탭이 존재합니다.

- register > completed & destroy는 컴포넌트가 화면에 배치되고 사라지는 과정에 실행되는 영역입니다.
 - click, dblclick은 해당하는 이벤트에 반응하여 실행되는 코드 영역입니다.
 - preview는 뷰어와 별개로 에디터에서만 실행되는 코드 영역입니다. (차트, 테이블, FreeCode)

• 아무것도 수정하지 않은 상태에서 Apply를 누르면 다음과 같은 결과를 확인할 수 있습니다.

ID	Name	Progress	Gender	Rating	Col	Dob	Car
1	Oli Bob	12	male	1	red	19/02/1984	1
2	Mary May	1	female	2	blue	14/05/1982	true
3	Christine Lobo...	42	female	0	green	22/05/1982	true
4	Brendon Philips	100	male	1	orange	01/08/1980	
5	Margret Marm...	16	female	5	yellow	31/01/1999	
6	Frank Harbours	38	male	4	red	12/05/1966	1

- 우리가 저장한 getName의 데이터에 맞도록 preview 코드를 수정해봅니다

```
JAVASCRIPT
click dblclick register completed destroy preview
1 //example data
2 const tabledata = [
3   { id: 1, name: "oli Bob" },
4
5 ];
6
7 this.previewTable = new Tabulator("#" + this.id, [
8   data: tabledata,
9   columns: [
10     { title: "ID", field: "id" },
11     { title: "Name", field: "name" },
12   ],
13 ]);
```

ID	Name
1	Oli Bob

- DataSet은 WKit 객체에서 fetchData 메소드를 활용해서 호출할 수 있습니다.
 - Completed 사이클에 다음과 같이 코드를 입력해봅니다.
 - 현재 개별 이름만 가져오는 API만 존재하는 상황을 가정하고, 목록을 만들어봅니다
 - 목록으로 테이블의 값을 설정합니다.
 - Tabulator에 대한 자세한 설명은 링크를 참고하세요

- [Tabulator - Setup Options](#)

```
1 const tabledata = [];
2 this.table = new Tabulator("#" + this.id, {
3   data: tabledata,
4   columns: [
5     { title: "ID", field: "id" },
6     { title: "Name", field: "name" },
7   ],
8 });
9 (async () => {
10   const idList = [1, 2, 3];
11   const getNamePromises = idList.map(id => WKit.fetchData(this.page, 'getName', { id }));
12   const responseLists = await Promise.all(getNamePromises).catch((err) => (console.error(err), []));
13   const tableData = responseLists.map(({ response }) => response);
14   this.table.setData(tableData);
15 })()
```

- 뷰어에서 결과를 확인할 수 있습니다.

ID	Name
1	John 1
2	John 2
3	John 3

interaction

"버튼을 클릭하여 차트를 그려보세요."

1. 버튼 인스턴스 이름: `refreshBtn` 으로 지정합니다
2. 차트 인스턴스 이름: `deviceChart1` 으로 지정합니다.
3. 코드 에디터에서 JavaScript 작성:
 - a. 버튼 컴포넌트의 HTML은 다음과 같이 입력합니다.

```
1 <button type="button" class="btn btn-primary">Render Chart</button>
```

- b. 버튼 컴포넌트의 register 사이클 JavaScript에 다음과 같이 입력하여 클릭 이벤트를 등록합니다

```
1 this.appendChild.addEventListener('click', () => {
2   const chartName = 'deviceChart1';
3   const iter = WKit.makeIterator(this.page);
4   const chartInstance = WKit.getInstanceByName(chartName, iter);
5 
6   chartInstance.render();
7 })
```

→ 버튼을 클릭하여(버튼을 포함하는 컨테이너를 클릭하여), 차트를 찾은 후 차트가 가진 render를 호출하여 그린다.

- c. 차트 컴포넌트의 register 사이클 JavaScript에 다음과 같이 입력하여 render 메소드를 정의합니다.

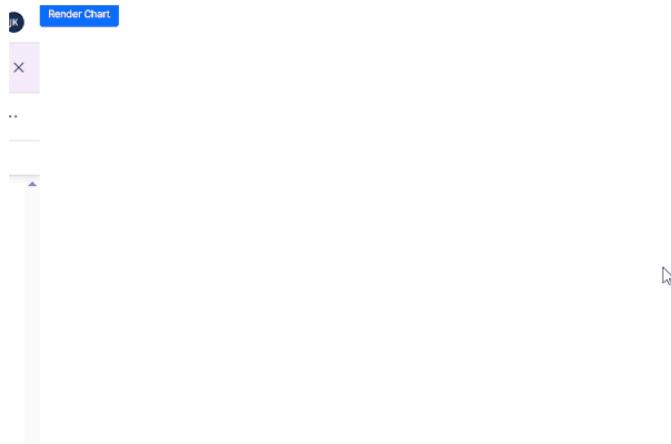
```
1 this.render = () => {
2   this.chart = echarts.init(this.element);
3   let option;
4   // example option
5   option = {
6     xAxis: {
7       type: 'category',
8       data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
9     },
10    yAxis: {
```

```

11     type: 'value'
12   },
13   series: [
14     {
15       data: [150, 230, 224, 218, 135, 147, 260],
16       type: 'line'
17     }
18   ]
19 };
20 option && this.chart.setOption(option);
21
22 }
23

```

→ render를 호출했을 경우 차트를 초기 설정하고, setOption을 통해 차트를 그린다.



Layer (stacked Layer)

Layer (Stacked Layer) – 2D와 3D의 공존

시각화 페이지에서 다양한 형태의 오브젝트를 배치하려면, 그 오브젝트가 어떤 차원에 속하는지를 구분하는 것이 중요합니다.

RENOBIT은 이를 위해 **2D Layer**, **3D Layer**, 그리고 **Master Layer**라는 세 개의 시각화 공간을 제공합니다.

왜 Layer가 필요할까?

- **2D 오브젝트**는 x, y 평면에서 작동합니다.
- **3D 오브젝트**는 x, y, z 공간의 깊이를 필요로 합니다.

같은 공간에 표현될 수 없기 때문에, 각 차원을 분리한 Layer 개념이 필요합니다.

도구모음을 통한 Layer 전환

상단의 **계층 아이콘**을 클릭하면,
어떤 Layer에 어떤 모양을 배치할 것인지 선택할 수 있습니다:

- **2D 선택 시**: 텍스트, 버튼, 도형 등의 2D 요소들이 보입니다.
- **3D 선택 시**: 장비, 공간 객체 등의 3D 요소들이 보입니다.

보기 모드 전환 – 눈 아이콘

오른쪽 상단의 **눈 모양 아이콘**을 통해 보기 모드를 전환할 수 있습니다:

-  2D만 보기
-  3D만 보기
-  2D + 3D 동시에 보기

Master Layer – 반복되는 요소를 위한 고정 레이어 (Component & CSS)

페이지를 구성할 때는 종종 다음과 같은 **반복되는 공통 UI**가 필요합니다:

- 네비게이션 바 (Navbar)
- 페이지 제목, 푸터
- 공통 알림 박스 등

이러한 공통 요소를 모든 페이지에 매번 새로 추가하는 것은 **비효율적**입니다.

그래서 **RENOBIT은 **Master Layer****라는 별도의 공간을 제공합니다.

Master Layer의 특징

-  2D 요소만 배치 가능
-  모든 페이지에서 공통으로 나타남
-  페이지 레이아웃과 독립적으로 유지
-  레이어 전환 UI에서 선택 가능

 즉, Master Layer는 사이트 전체의 공통 인터페이스를 관리하는 공간입니다.

Layer 구성 요약

레이어 종류	설명	주요 오브젝트 예시
2D Layer	페이지 전용 2D 시각 요소 배치	버튼, 텍스트, 이미지, 차트 등
3D Layer	페이지 전용 3D 시각 요소 배치	장비, 모델, 공간 구조체 등

Master Layer	모든 페이지에 공통으로 나타나는 2D 요소	Navbar, 페이지 헤더, 고정 안내 등
---------------------	-------------------------	-------------------------

▼ How to Use Code box

- HTML

- HTML Area (innerHTML Of Root Container (root div))

- Used to render HTML structure

- **data-*** attributes are recommended for storing metadata used in JavaScript or CSS

→ Useful for runtime logic, component targeting, or conditional styling

- Reference: [Use data attributes - HTML | MDN](#)

- Dynamic template (With JavaScript)

- Use the **<template>** tag to define reusable HTML fragments.

- Reference: [Using templates and slots - Web APIs | MDN](#)

▼ Dynamic Template Example

```

1 <h1>유저 목록</h1>
2 \<ul id="userList"></ul>
3
4 <!-- 템플릿 정의 -->
5 <template id="userItemTemplate">
6   <li class="user-item">
7     <span class="name"></span>
8   </li>
9 </template>
10
11
12 <!-- In JavaScript Area
13
14   const users = ['Alice', 'Bob', 'Charlie'];
15   const list = document.getElementById('userList');
16   const template = document.getElementById('userItemTemplate');
17
18   users.forEach(name => {
19     // 템플릿의 내용을 복제
20     const clone = template.content.cloneNode(true);
21     // 복제한 템플릿의 내부 내용 수정
22     clone.querySelector('.name').textContent = name;
23     // DOM에 삽입
24     list.appendChild(clone);
25   });
26
27
28 -->

```

- CSS
 - Panel Style
 - Applied to the root container (`<div>`)
 - Inline styles have higher priority
 - `!important` is used in specific scenarios where necessary
 - css specificity, selector
 - [CSS selectors - CSS | MDN](#)
 - [CSS selectors and combinators - CSS | MDN](#)
 - copy ID
 - Click the component name in the code box to copy its ID.
- JavaScript (“this” mean , page or component)
 - “this” Context
 - Page Script → Refers to the **page**
 - Component Script
 - Refers to the **component itself**,
 - **this.page** → Refers to the **parent page** from within a component
 - Page
 - life cycle
 - **beforeLoad**
 - Runs when only the page metadata is set.
 - **Component data is not accessible** yet.
 - Typically used for initial preparation before anything is rendered.
 - **ready**
 - Component files are loaded, and **instance information is now available** (excluding resources).
 - Resource loading for each component starts at this stage.
 - **loaded**
 - All components and their **resources have been fully loaded**.
 - The entire screen is ready for user interaction.
 - **beforeUnload**
 - Executes right **before a new page is opened**.
 - **Existing component information is still accessible**, but layers are hidden.

- **unload**

- The previous page and its components are **fully removed from memory**.

- wemb.mainPageComponent → page

- In Page Script, “this” means page

- Component Iterator (**WKit** → **will be available as of version 3.1.0**)

- **WKit.makeIterator(page)** → returns a list of **all components across all layers**

- You can specify target layers as parameters:

- e.g. **'twoLayer'** , **'threeLayer'** , **'masterLayer'**

- 1 | WKit.makeIterator(page, 'twoLayer'); // Two Layer only
 - 2 | WKit.makeIterator(page, 'twoLayer', 'threeLayer'); // Two + Three Layer

- wemb.threeElements (For Three Layer)

- scene

-  [three.js docs](#)

- camera

-  [three.js docs](#)

- renderer

-  [three.js docs](#)

- wemb.mainControls

- target

- The focus point of the controls, the object orbits around this.

- Component

- life cycle

- register → completed → destroy

- rootContainer

- this.appendChild (2D DOM → div, 3D → Object3D Group)

- In Component Script, “this” means components itself

- this.page → Refers to the **parent page** from within a component

- Preview

- TabulatorTable , Echarts

- table, chart name → should be “previewTable“ or “previewChart“ **for memory management**

- Others tab → doesn't matter

- Instance Option
 - Import Resource URL
 - Select Resource and Copy URL
 - Font List
 - Select and Copy font family string → font-family: “NotoSans“

▼ Popup (RENOBIT Page Popup, Two Layer Only) & Navigate To a Page

```
# Popup ( RENOBIT Page Popup, Two Layer Only )
```

- Open Popup
 - 1 wemb.popupManager.open(pageName, options);
- Close Popup
 - 1 wemb.popupManager.closed(pageName); // Close a specific popup
 - 2 wemb.popupManager.clear() // Close all popups
- Configure Popup Options

- The **options** object supports the following properties:

```
o 1 let options = {
  2   // Percentage units (e.g., '50%') are calculated relative to the page size
  3   width: '300px', // Accepts px, %, rem, em, etc. If number only, treated as px
  4   height: 300, // Number only – interpreted as '300px'
  5   x: '50%',
  6   y: 0,
  7   title: 'This is Title', // Title displayed in the header
  8   className: 'customClass',
  9   params: {
 10     // Parameters to be passed to the page
 11     customKey: 'custom Value'
 12   }
 13 }
```

- Access params in Popup
 - Inside a Popup page, during the **loaded** lifecycle:

- 1 console.log(this.params)

- (In Popup Page) Custom Style Popup (Header , Close)
 - Using Component as a Dragable Header or Close Button
- (In Popup Page) Set Popup Background
 - Popup background is customizable via the page's own styling

- Page Background → Use And Setting Background

Navigate to Another Page

- Navigate to Another Page

- ```
1 let pageName = "myCustomPage";
2 wemb.pageManager.openPageByName(pageName, {
3 customKeyOfPage: 'custom Value of Page'
4 });
```

- Access Parameters in Page ( Same as in the Popup )

- Inside a page, during the **loaded** lifecycle:

- ```
1 // You can access the parameters in the same way as in the popup page
2 console.log(this.params)
```

```
↳ {customKeyOfPage: 'custom Value of Page'} '-----'
```

▪

▼ Memory Management

Cycle (when to clean)

- Page's before unLoad
- Component's destroy

Clear Memory

- Bootstrap

▼ Initialize

- Initialized By JavaScript (Alert, Toasts, Tooltips, Popovers)

- - B Alerts**
 - B Toasts**
 - B Tooltips**
 - B Popovers**

- ```
1 // RENOBIT Tooltip Component completed cycle
2 const tooltipTriggerList = this.element.querySelectorAll('[data-bs-toggle="tooltip"]');
3 this.tooltipList = [...tooltipTriggerList].map(tooltipTriggerEl => new Bootstrap.Tooltip(tooltipTriggerEl));
```

### ▼ Clean

- B JavaScript**

## Bootstrap 컴포넌트별 메모리 정리 방법

### 1. Modal, Tooltip, Popover 등: `dispose()` 사용

Bootstrap의 주요 컴포넌트는 인스턴스를 생성하며, 명시적으로 제거할 수 있는 메서드를 제공합니다.

#### 예시 – Tooltip ( Initialized via JavaScript)

```
1 // RENOBIT Tooltip Component destroy cycle
2 this.tooltipList.forEach(toolTipIns => toolTipIns.dispose());
3 this.tooltipList = []
```

#### 예시 – Modal ( Initialized via Data Attribute )

```
1 const modalEl = document.querySelector('.modal'); // Modal 루트 요소
2
3 const modalInstance = Bootstrap.Modal.getInstance(modalEl);
4 modalInstance?.dispose();
```

- Tabulator

- Clean

-  Tabulator - Events

#### 1. `destroy()` 메서드 사용

Tabulator 인스턴스를 제거하려면 반드시 `.destroy()` 를 호출해야 합니다.

```
1 const table = new Tabulator("#example-table", {
2 data: [...],
3 columns: [...]
4 });
5
6 // 정리 시
7 table.destroy();
```

`.destroy()` 는 다음을 수행합니다:

- 내부 데이터 구조 정리
- 이벤트 리스너 제거
- DOM 요소 해제
- 글로벌 참조 제거

#### 예시 – TabulatorTable Component 정리 코드

```
1 // RENOBIT TabulatorTable Component destroy cycle
2 this.resizeObserver instanceof ResizeObserver && this.resizeObserver.disconnect();
3 this.resizeObserver = null;
4 this.table instanceof Tabulator && this.table.destroy();
5 this.table = null;
```

- Echarts

▫ Clean

- [Documentation - Apache ECharts](#)

- [ECharts 공식 dispose 문서](#)

- 캔버스 기반이므로 브라우저가 자동으로 정리할 거라 생각할 수 있지만, 이벤트와 내부 상태는 명시적으로 정리해야 안전합니다.

## ECharts 메모리 정리 방법

### 1. dispose() 호출

```
1 const chart = echarts.init(document.getElementById('myChart'));
2 // ...
3 chart.dispose();
4
```

`.dispose()` 는 다음을 정리합니다:

- DOM에 바인딩된 캔버스/그래픽 인스턴스 제거
- 내부 이벤트 리스너 제거
- 애니메이션 루프 정리
- 메모리 캐시 해제

**주의:** `.dispose()` 는 `echartsInstance`를 통해 호출해야 하며, DOM 요소만 제거 한다고 해서 메모리 정리가 되지 않습니다.

### 예시 – Echarts Component 정리 코드

```
1 // RENOBIT Echarts Component destroy cycle
2 this.resizeObserver instanceof ResizeObserver && this.resizeObserver.disconnect();
3 this.resizeObserver = null;
4
5 this.isEchartsInstance(this.chart) && this.chart.dispose();
6 this.chart = null;
```

- Three.js

▫ Clean

- [three.js manual](#)

```
1 // fx.go, WKit.dispose3DTree will be available as of version 3.1.0
2 const { scene } = wemb.threeElements;
3 fx.go(
4 WKit.makeIterator(this, 'threeLayer'),
```

```
5 map(({ appendElement }) => WKit.dispose3DTree(appendElement))
6)
```

∨ Architecture Case

## 🔗 EventBus Pattern: Event-Driven Communication in JS

### w Publish–subscribe pattern

## # Utils

∨ fx.js

```
1 // source --> https://github.com/indongyoo/functional-javascript-01/blob/master/fx.js
2 const fx = {};
3
4 const curry =
5 (f) =>
6 (a, ...) =>
7 _length ? f(a, ...) : (..._) => f(a, ..._);
8
9 fx.curry = curry;
10
11 const isIterable = (a) => a && a[Symbol.iterator];
12
13 fx.isIterable = isIterable;
14
15 const go1 = (a, f) => (a instanceof Promise ? a.then(f) : f(a));
16
17 fx.go1 = go1;
18
19 const reduceF = (acc, a, f) =>
20 a instanceof Promise
21 ? a.then(
22 (a) => f(acc, a),
23 (e) => (e == nop ? acc : Promise.reject(e))
24)
25 : f(acc, a);
26
27 fx.reduceF = reduceF;
28
29 const head = (iter) => go1(take(1, iter), ([h]) => h);
30
31 fx.head = head;
32
33 const reduce = curry((f, acc, iter) => {
34 if (!iter) return reduce(f, head((iter = acc[Symbol.iterator]())));
35
36 iter = iter[Symbol.iterator]();
37 return go1(acc, function recur(acc) {
38 let cur;
39 while (!(cur = iter.next()).done) {
40 acc = reduceF(acc, cur.value, f);
41 if (acc instanceof Promise) return acc.then(recur);
42 }
43 return acc;
44 });
45 });

});
```

```
46 fx.reduce = reduce;
47
48
49 const go = (...args) => reduce((a, f) => f(a), args);
50
51 fx.go = go;
52
53 const pipe =
54 (f, ...fs) =>
55 (...as) =>
56 go(f(...as), ...fs);
57
58 fx.pipe = pipe;
59
60 const take = curry((l, iter) => {
61 let res = [];
62 iter = iter[Symbol.iterator]();
63 return (function recur() {
64 let cur;
65 while (!(cur = iter.next()).done) {
66 const a = cur.value;
67 if (a instanceof Promise) {
68 return a
69 .then((a) => ((res.push(a), res).length == l ? res : recur()))
70 .catch((e) => (e == nop ? recur() : Promise.reject(e)));
71 }
72 res.push(a);
73 if (res.length == l) return res;
74 }
75 return res;
76 })();
77 });
78
79 fx.take = take;
80
81 const each = curry((f, iter) =>
82 go1(
83 reduce((_, a) => f(a), null, iter),
84 (_) => iter
85)
86);
87
88 fx.each = each;
89
90 const takeAll = take(Infinity);
91
92 fx.takeAll = takeAll;
93
94 const L = {};
95
96 fx.L = L;
97
98 L.range = function* (l) {
99 let i = -1;
100 while (++i < l) yield i;
101 };
102
103 L.map = curry(function* (f, iter) {
```

```

104 for (const a of iter) {
105 yield go1(a, f);
106 }
107 });
108
109 const nop = Symbol('nop');
110
111 L.filter = curry(function* (f, iter) {
112 for (const a of iter) {
113 const b = go1(a, f);
114 if (b instanceof Promise) yield b.then((b) => (b ? a : Promise.reject(nop)));
115 else if (b) yield a;
116 }
117 });
118
119 L.entries = function* (obj) {
120 for (const k in obj) yield [k, obj[k]];
121 };
122
123 L.flatten = function* (iter) {
124 for (const a of iter) {
125 if (isIterable(a)) yield* a;
126 else yield a;
127 }
128 };
129
130 L.deepFlat = function* f(iter) {
131 for (const a of iter) {
132 if (isIterable(a)) yield* f(a);
133 else yield a;
134 }
135 };
136
137 L.flatMap = curry(pipe(L.map, L.flatten));
138
139 const map = curry(pipe(L.map, takeAll));
140
141 fx.map = map;
142
143 const filter = curry(pipe(L.filter, takeAll));
144
145 fx.filter = filter;
146
147 const find = curry((f, iter) => go(iter, L.filter(f), take(1), ([a]) => a));
148
149 fx.find = find;
150
151 const flatten = pipe(L.flatten, takeAll);
152
153 fx.flatten = flatten;
154
155 const flatMap = curry(pipe(L.map, flatten));
156
157 fx.flatMap = flatMap;
158
159 const range = (l) => {
160 let i = -1;
161 let res = [];

```

```

162 while (++i < l) {
163 res.push(i);
164 }
165 return res;
166 };
167
168 fx.range = range;
169
170 const C = {};
171
172 fx.C = C;
173
174 function noop() {}
175
176 const catchNoop = ([...arr]) => (
177 arr.forEach((a) => (a instanceof Promise ? a.catch(noop) : a)), arr
178);
179
180 C.reduce = curry((f, acc, iter) =>
181 iter ? reduce(f, acc, catchNoop(iter)) : reduce(f, catchNoop(acc))
182);
183
184 C.take = curry((l, iter) => take(l, catchNoop(iter)));
185
186 C.takeAll = C.take(Infinity);
187
188 C.map = curry(pipe(L.map, C.takeAll));
189
190 C.filter = curry(pipe(L.filter, C.takeAll));
191

```

▽ GlobalDataPublisher

```

1 const GlobalDataPublisher = () => {
2 const mappingTable = new Map();
3 const subscriberTable = new Map();
4
5 return {
6 registerMapping({ topic, datasetInfo }) {
7 mappingTable.set(topic, datasetInfo);
8 return {
9 topic,
10 datasetInfo,
11 };
12 },
13
14 unregisterMapping(topic) {
15 mappingTable.delete(topic);
16 },
17
18 async fetchAndPublish(topic, page) {
19 const datasetInfo = mappingTable.get(topic);
20 if (!datasetInfo) {
21 console.warn(`[GlobalDataPublisher] 등록되지 않은 topic: ${topic}`);
22 return;
23 }
24
25 const data = await WKit.fetchData(page, datasetInfo.datasetName, datasetInfo.param);

```

```

26 const subs = subscriberTable.get(topic) || new Set();
27
28 for (const { instance, handler } of subs) {
29 handler.call(instance, data);
30 }
31 },
32
33 subscribe(topic, instance, handler) {
34 if (!subscriberTable.has(topic)) subscriberTable.set(topic, new Set());
35 subscriberTable.get(topic).add({ instance, handler });
36 },
37
38 unsubscribe(topic, instance) {
39 const subs = subscriberTable.get(topic);
40 if (!subs) return;
41 for (const sub of subs) {
42 if (sub.instance === instance) subs.delete(sub);
43 }
44 },
45 getGlobalMappingSchema({
46 topic = 'weather',
47 datasetInfo = {
48 datasetName: 'dummyjson',
49 param: { dataType: 'weather', id: 'default' },
50 },
51 } = {}) {
52 return {
53 topic,
54 datasetInfo,
55 };
56 },
57 };
58 })();
59

```

#### ✓ WEventBus.js

```

1 const WEventBus = () => {
2 // controller
3 const listeners = new Map();
4
5 return {
6 on(event, callback) {
7 if (!listeners.has(event)) {
8 listeners.set(event, []);
9 }
10 listeners.get(event).push(callback);
11 },
12
13 off(event, callback) {
14 if (!listeners.has(event)) return;
15 const newList = listeners.get(event).filter((cb) => cb !== callback);
16 listeners.set(event, newList);
17 },
18
19 emit(event, data) {
20 if (!listeners.has(event)) return;
21 for (const callback of listeners.get(event)) {

```

```
22 callback(data);
23 }
24 },
25
26 once(event, callback) {
27 const wrapper = (data) => {
28 callback(data);
29 this.off(event, wrapper);
30 };
31 this.on(event, wrapper);
32 },
33 };
34 })();
35
```

▽ WKit.js

```
1 const WKit = {};
2
3 /* Public API: data mapping */
4 WKit.pipeForDataMapping = function (targetInstance) {
5 const currentPage = wemb.isPage(targetInstance) ? targetInstance : targetInstance.page;
6 return new Promise((res, rej) => {
7 fx.go(resolveMappingInfo(targetInstance), fx.map(getDataFromMapping.bind(currentPage)))
8 .then(res)
9 .catch(rej);
10 });
11 };
12
13 /* Public API: 2D event binding */
14 WKit.bindEvents = function (instance, customEvents) {
15 fx.go(
16 Object.entries(customEvents),
17 fx.map(([eventName, selectorList]) => {
18 fx.map((selector) => {
19 const handler = makeHandler(instance, selector);
20 delegate(instance, eventName, selector, handler);
21 }, Object.keys(selectorList));
22 })
23);
24 };
25
26 WKit.removeCustomEvents = function (instance, customEvents) {
27 fx.go(
28 Object.entries(customEvents),
29 fx.map(([eventName, selectorList]) => {
30 fx.map((selector) => {
31 const handler = instance.userHandlerList?.[eventName]?.[selector];
32 if (handler) {
33 instance.element.removeEventListener(eventName, handler);
34 }
35 }, Object.keys(selectorList));
36 })
37);
38 };
39
40 /* Public API: 3D event binding */
41
```

```

42 WKit.initThreeRaycasting = function (target, eventName) {
43 const raycaster = new THREE.Raycaster();
44 const mouse = new THREE.Vector2();
45 const { scene, camera } = wemb.threeElements;
46 const onRaycasting = makeRaycastingFn(target, raycaster, mouse, scene, camera);
47 target.addEventListener(eventName, onRaycasting);
48 return onRaycasting;
49 };
50
51 WKit.bind3DEvents = function (instance, customEvents) {
52 instance.appendChild.eventListener = {};
53 fx.map((browserEvent) => {
54 const eventHandler = make3DHandler(instance);
55 instance.appendChild.eventListener[browserEvent] = eventHandler;
56 }, Object.keys(customEvents));
57 };
58
59 /* Public API: 3D dispose */
60 WKit.dispose3DTree = function (rootContainer) {
61 rootContainer.traverse((obj) => {
62 // 1. geometry
63 if (obj.geometry) {
64 obj.geometry.dispose?();
65 }
66
67 // 2. material(s)
68 if (obj.material) {
69 if (Array.isArray(obj.material)) {
70 obj.material.forEach((mat) => {
71 disposeMaterial(mat);
72 });
73 } else {
74 disposeMaterial(obj.material);
75 }
76 }
77
78 // 3. textures (in material, handled inside disposeMaterial)
79
80 // 4. eventListener (custom-defined on your side)
81 if (obj.eventListener) {
82 Object.keys(obj.eventListener).forEach((eventType) => {
83 obj.eventListener[eventType] = undefined;
84 });
85 obj.eventListener = undefined;
86 }
87
88 // 5. 기타 사용자 정의 데이터
89 if (obj.userData) {
90 obj.userData = {};
91 }
92 });
93
94 // 부모로 부터 detach
95 if (rootContainer.parent) {
96 rootContainer.parent.remove(rootContainer);
97 }
98 };
99

```

```

100 WKit.clearSceneBackground = function (scene) {
101 const bg = scene.background;
102
103 if (bg && bg.dispose) {
104 bg.dispose(); // Texture & CubeTexture 일 경우만 dispose 존재
105 }
106
107 scene.background = null;
108 };
109
110 /* Public API: helper */
111 WKit.makeIterator = function (page, ...layerList) {
112 layerList = layerList.length ? layerList : ['masterLayer', 'twoLayer', 'threeLayer'];
113 const mapName = {
114 masterLayer: 'componentInstanceListMap',
115 twoLayer: 'componentInstanceListMap',
116 threeLayer: '_appendElementListMap',
117 };
118 return combineIterators(
119 fx.go(
120 layerList,
121 fx.map((layer) => page?[layer]?.[mapName[layer]]?.values())
122)
123);
124 };
125
126 WKit.getInstanceByName = function (instanceName, iter) {
127 return fx.find((ins) => ins.name === instanceName, iter);
128 };
129
130 WKit.getInstanceById = function (targetId, iter) {
131 return fx.find((ins) => ins.id === targetId, iter);
132 };
133
134 WKit.fetchData = function (page, datasetName, param) {
135 return new Promise((res, rej) => {
136 page.dataService
137 .call(datasetName, { param })
138 .on('success', (data) => res(data))
139 .on('error', (err) => rej(err));
140 });
141 };
142
143 WKit.emitEvent = function (eventName, targetInstance) {
144 console.log('[WKit:EmitByCode]', eventName, targetInstance);
145 WEventBus.emit(eventName, {
146 targetInstance,
147 });
148 };
149
150 WKit.triggerEventToTargetInstance = function (
151 targetInstanceName,
152 eventName,
153 iter = WKit.makeIterator(wemb.mainPageComponent)
154) {
155 fx.go(
156 fx.range(1),
157 (_)> WKit.getInstanceByName(targetInstanceName, iter),

```

```
158 fx.curry(WKit.emitEvent)(eventName)
159);
160 };
161
162 /* Public API: event bus on / off */
163 WKit.onEventBusHandlers = function (eventBusHandlers) {
164 fx.go(
165 Object.entries(eventBusHandlers),
166 fx.map(([eventName, handler]) => WEventBus.on(eventName, handler))
167);
168 };
169
170 WKit.offEventBusHandlers = function (eventBusHandlers) {
171 fx.go(
172 Object.entries(eventBusHandlers),
173 fx.map(([eventName, handler]) => WEventBus.off(eventName, handler))
174);
175 };
176
177 /* Public API: schema utility */
178 WKit.getDataMappingSchema = function () {
179 return [
180 {
181 ownerId: 'ownerId',
182 visualInstanceList: ['Chart_for_specific_model'],
183 datasetInfo: {
184 datasetName: 'dummyjson',
185 param: {
186 dataType: 'carts',
187 id: 'ownerId',
188 },
189 },
190 },
191];
192 };
193
194 WKit.getGlobalMappingSchema = function () {
195 return [
196 {
197 topic: 'users',
198 datasetInfo: {
199 datasetName: 'dummyjson',
200 param: { dataType: 'users', id: 'default' },
201 },
202 },
203 {
204 topic: 'comments',
205 datasetInfo: {
206 datasetName: 'dummyjson',
207 param: { dataType: 'comments', id: 'default' },
208 },
209 },
210];
211 };
212
213 WKit.getCustomEventsSchema = function () {
214 return {
215 click: {
```

```

216 '.navbar-brand': '@triggerNavbarTitle',
217 '.nav-link': '@triggerNavLink',
218 '.dropdown-item': '@triggerDropDownItem',
219 },
220 submit: {
221 form: '@submitForm',
222 },
223 };
224 };
225
226 WKit.getCustomEventsSchemaFor3D = function () {
227 return {
228 click: '@triggerClick',
229 };
230 };
231
232 WKit.getSubscriptionSchema = function () {
233 return {
234 users: ['method1', 'method2'],
235 comments: ['method3', 'method4'],
236 };
237 };
238
239 /*Internal only: utils for data mapping */
240 async function getDataFromMapping(
241 ownerId,
242 visualInstanceList,
243 datasetInfo: { datasetName, param },
244) {
245 return {
246 ownerId,
247 visualInstanceList: fx.map(
248 (visualInstanceName) => WKit.getInstanceByName(visualInstanceName, WKit.makeIterator(this)),
249 visualInstanceList
250),
251 data: await WKit.fetchData(this, datasetName, param).catch((err) => (console.error(err), [])),
252 };
253 }
254
255 function resolveMappingInfo(targetInstance) {
256 let dataMapping = [];
257
258 if (!dataMapping.length && targetInstance.dataMapping) {
259 dataMapping = targetInstance.dataMapping;
260 console.info('[Fallback] instance.dataMapping 사용됨');
261 }
262
263 if (!dataMapping.length) {
264 throw new Error(`매핑 정보가 없습니다. instanceId: ${targetInstance.id}`);
265 }
266
267 return dataMapping;
268 }
269
270 /*Internal only: utils for 2D event */
271
272 function makeHandler(targetInstance, selector) {
273 return function (event) {

```

```
274 event.preventDefault();
275 const { customEvents } = targetInstance;
276 const triggerEvent = customEvents?.[event.type]?.[selector];
277 if (triggerEvent) {
278 console.log('@eventHandler', customEvents[event.type][selector]);
279 WEventBus.emit(triggerEvent, {
280 event,
281 targetInstance,
282 });
283 }
284 };
285 }
286
287 /*Internal only: utils for 3D */
288
289 function makeRaycastingFn(rootElement, raycaster, mouse, scene, camera) {
290 return function (event) {
291 mouse.x = (event.offsetX / rootElement.clientWidth) * 2 - 1;
292 mouse.y = -(event.offsetY / rootElement.clientHeight) * 2 + 1;
293 raycaster.setFromCamera(mouse, camera);
294 const intersects = raycaster.intersectObjects(scene.children, true);
295 fx.go(
296 intersects,
297 fx.L.map((inter) => inter.object),
298 fx.L.map((obj) => {
299 let current = obj;
300 while (current && !current.eventListener) {
301 current = current.parent;
302 }
303 return current;
304 }),
305 fx.L.filter(Boolean),
306 fx.take(1),
307 ([target]) => target?.eventListener?.[event.type]?(Object.assign(event, { intersects })))
308);
309 };
310 }
311
312 function make3DHandler(targetInstance) {
313 return function (event) {
314 const { customEvents } = targetInstance;
315 console.log('@eventHandler', customEvents[event.type]);
316 WEventBus.emit(customEvents[event.type], {
317 event,
318 targetInstance,
319 });
320 };
321 }
322
323 function disposeMaterial(material) {
324 // dispose texture in known slots
325 const slots = [
326 'map',
327 'lightMap',
328 'aoMap',
329 'emissiveMap',
330 'bumpMap',
331 'normalMap',
```

```

332 'displacementMap',
333 'roughnessMap',
334 'metalnessMap',
335 'alphaMap',
336 'envMap',
337 'specularMap',
338 'gradientMap',
339];
340
341 slots.forEach((key) => {
342 const tex = material[key];
343 if (tex && tex.dispose) {
344 tex.dispose();
345 material[key] = null;
346 }
347 });
348
349 material.dispose?.();
350 }
351
352 /*Internal only: utils for general */
353 function* combineIterators(iterables) {
354 for (const iterable of iterables) {
355 yield* iterable;
356 }
357 }
358
359 function qsAll(selector, scope = document) {
360 if (!selector) throw 'no selector';
361
362 return Array.from(scope.querySelectorAll(selector));
363 }
364
365 function delegate(instance, eventName, selector, handler) {
366 const emitEvent = (event) => {
367 const potentialElements = qsAll(selector, instance.element);
368 for (const potentialElement of potentialElements) {
369 if (potentialElement === event.target) {
370 return handler.call(event.target, event);
371 }
372 }
373 };
374
375 instance.userHandlerList = instance.userHandlerList || {};
376 instance.userHandlerList[eventName] = instance.userHandlerList[eventName] || {};
377 instance.userHandlerList[eventName][selector] = emitEvent;
378
379 instance.element.addEventListener(eventName, emitEvent);
380 }
381

```

## # Runtime Code ( RENOBIT Code )

- Runtime code Page Before load ( EventBus, Init Raycasting )

```
1 const { onEventBusHandlers, initThreeRaycasting, pipeForDataMapping } = WKit;
```

```

2
3 initPageController.call(this);
4
5 function initPageController() {
6 this.eventBusHandlers = getEventBusHandlers.call(this);
7 onEventBusHandlers.call(this, this.eventBusHandlers);
8
9 this.raycastingEventType = 'click';
10 this.raycastingEventHandler = initThreeRaycasting(this.element, this.raycastingEventType);
11 };
12
13 function getEventBusHandlers() {
14 return {
15 '@myClickEvent']: async ({ event, targetInstance }) => {
16 const dataFromMapping = await pipeForDataMapping(targetInstance);
17 console.log('@myClickEvent', event);
18 console.log('@targetInstance', targetInstance)
19 console.log('@Data From Mapping', dataFromMapping)
20 }
21 }
22 };
23
24

```

▽ Runtime code Page Loaded ( DataPublisher )

```

1 const { go, each, C } = fx
2
3 initPageDataPublisher.call(this);
4
5 function initPageDataPublisher() {
6 this.globalDataMappings = getGlobalDataMappings();
7 go(
8 this.globalDataMappings,
9 each(GlobalDataPublisher.registerMapping),
10 each(({ topic }) => GlobalDataPublisher.fetchAndPublish(topic, this))
11)
12
13 };
14
15 function getGlobalDataMappings() {
16 return [
17 {
18 topic: 'users',
19 datasetInfo: {
20 datasetName: 'dummyjson',
21 param: { dataType: 'users', id: 'default' },
22 },
23 },
24 {
25 topic: 'comments',
26 datasetInfo: {
27 datasetName: 'dummyjson',
28 param: { dataType: 'comments', id: 'default' },
29 },
30 },
31];
32 };

```

33  
34  
35  
36

✓ Runtime code Page Before UnLoad ( clear event, data publisher, three elements)

```
1 const { go, map } = fx;
2 const { makeIterator, dispose3DTree, clearSceneBackground, offEventBusHandlers } = WKit;
3
4 onPageUnLoad.call(this);
5
6 function onPageUnLoad() {
7 clearEventBus.call(this);
8 clearDataPublisher.call(this);
9 clearThreeInstances.call(this);
10};
11
12 function clearEventBus() {
13 offEventBusHandlers.call(this, this.eventBusHandlers);
14 this.eventBusHandlers = null;
15};
16
17 function clearDataPublisher() {
18 go(
19 this.globalDataMappings,
20 map(({ topic }) => topic),
21 each((GlobalDataPublisher.unregisterMapping))
22)
23 };
24
25
26 function clearThreeInstances() {
27 const { scene } = wemb.threeElements;
28 go(
29 makeIterator(this, 'threeLayer'),
30 map(({ appendElement }) => dispose3DTree(appendElement))
31)
32
33 clearSceneBackground(scene);
34 this.element.removeEventListener(this.raycastingEventType, this.raycastingEventHandler);
35 this.raycastingEventHandler = null;
36 };
37
38
```

✓ Runtime code Component\_3D register ( Data Mapping , Event Binding)

```
1
2 const { bind3DEvents } = WKit;
3 const { go, L } = fx;
4
5 this.customEvents = getCustomEvents();
6 this.dataMapping = getDataMapping.call(this);
7 init.call(this);
8
9 function init() {
10 bind3DEvents(this, this.customEvents);
11};
```

```

12
13 function getCustomEvents() {
14 return {
15 click: '@myClickEvent'
16 };
17 };
18
19 function getDataMapping() {
20 return [
21 {
22 ownerId: this.id,
23 visualInstanceList: ['ComponentDataVisualizer'],
24 datasetInfo: {
25 datasetName: 'dummyjson',
26 param: {
27 dataType: 'carts',
28 id: this.id
29 }
30 }
31 },
32];
33 };
34

```

▽ Runtime code Component\_2D register default ( Property, Method, Event Binding )

```

1 const { go, L } = fx;
2 const { bindEvents } = WKit;
3
4 initComponent.call(this);
5
6 function initComponent() {
7 this.customEvents = getCustomEvents.call(this);
8 this.myMethod = myMethod.bind(this);
9 bindEvents(this, this.customEvents);
10 };
11
12 function getCustomEvents() {
13 return {
14 click: {
15 [`selector`]: '@myEvent'
16 }
17 }
18 };
19
20 function myMethod(data) {
21 console.log(`[myMethod] ${this.name}`, data);
22 };
23
24

```

▽ Runtime code Component\_2D destroy default ( Remove Event )

```

1 const { removeCustomEvents } = WKit;
2
3 onInstanceUnLoad.call(this);
4
5 function onInstanceUnLoad() {
6 removeCustomEvents(this, this.customEvents);

```

```
7 };
```

✓ Runtime code Component\_2D register ( Subscribe Page )

```
1 const { L, go, each } = fx;
2 const { subscribe } = GlobalDataPublisher;
3
4 initComponent.call(this);
5
6 function initComponent() {
7 this.subscriptions = getSubscriptions();
8 this.renderTable = renderTable.bind(this);
9 go(
10 Object.entries(this.subscriptions),
11 each(([topic, fnList]) =>
12 each(fn => this[fn] && subscribe(topic, this, this[fn]), fnList)
13)
14);
15 }
16
17 function getSubscriptions() {
18 return {
19 users: ['renderTable'],
20 }
21 }; 1
22
23 function renderTable(data) {
24 console.log(`[Render Table] ${this.name}`, data, 'subscription result');
25 };
26
27
```

✓ Runtime code Component\_2D destroy ( UnSubscribe Page )

```
1 const { unsubscribe } = GlobalDataPublisher;
2
3 onInstanceUnLoad.call(this);
4
5 function onInstanceUnLoad() {
6 clearSubscribe(this);
7 }
8
9 function clearSubscribe(instance) {
10 go(
11 Object.entries(instance.subscriptions),
12 each(([topic, _]) => unsubscribe(topic, instance))
13);
14 }
```

✓ Runtime code Component\_2D completed ( trigger Specific 3D Component )

```
1 const { triggerEventToTargetInstance } = WKit;
2 const { targetInstanceName, eventName } = getDefaultEventTarget();
3
4 triggerEventToTargetInstance(targetInstanceName, eventName);
5
6 function getDefaultEventTarget() {
7 return {
8 targetInstanceName: 'DataMappedComponent',
```

```
9 eventName: '@myClickEvent'
10 }
11);
```

▼ Further Study

# frontend essential

 [MDN Curriculum | MDN Curriculum](#)

 [HTTP: Hypertext Transfer Protocol | MDN](#)

 [Using the Fetch API - Web APIs | MDN](#)

 [Using promises - JavaScript | MDN](#)

 [The box model - Learn web development | MDN](#)

 [Specificity - CSS | MDN](#)

 [Using templates and slots - Web APIs | MDN](#)

 [Using CSS nesting - CSS | MDN](#)

# Library

 [Tabulator - Interactive JavaScript Tables](#)

 [Apache ECharts](#)

 [Bootstrap](#)

 [Three.js – JavaScript 3D library](#)

 [GitHub - marpple/FxJS: Functional Extensions Library for JavaScript](#)

# Responsive Web Design

 [Learn Responsive Design | web.dev](#)

 [\\* CSS Flexbox Layout Guide | CSS-Tricks](#)

 [\\* CSS Grid Layout Guide | CSS-Tricks](#)

 이번에야말로 CSS Flex를 익혀보자

 이번에야말로 CSS Grid를 익혀보자

 [프론트엔드 필수 반응형 CSS 단위 총정리 \(EM과 REM\) | Responsive CSS Units](#)

 [CSS Flexbox 완전 정리. 포트폴리오 만드는 날까지! | 프론트엔드 개발자 입문편: HTML, CSS, JavaScript](#)

 [avascript](#)

 CSS Grid 완전 정리 끝판왕 😎

 ResizeObserver - Web APIs | MDN

# Memory

 JavaScript data types and data structures - JavaScript | MDN

 Object references and copying

 Memory management - JavaScript | MDN

 Garbage collection

 Fix memory problems | Chrome DevTools | Chrome for Developers

 Chrome DevTools | Chrome for Developers

# Figma

 Introducing our MCP server: Bringing Figma into your workflow | Figma Blog

 Guide to the Figma MCP server

 GitHub - GLips/Figma-Context-MCP: MCP server to provide Figma layout information to AI coding agents like Cursor

 Figma MCP 사용법과 코드 변환 테스트 후기