

Vue로 instagram 웹버전 만들기

1. 포스팅 보여주기
2. 이미지업로드
3. 필터선택
4. 글 작성
5. 발행
6. 기타 기능 (더보기, 데이터관리 등)

- 여기다 서버만 추가하면 되겠네!

새로운 프로젝트 생성하기 :
vue ui / vue create 어쩌구

우리가 만들 앱의 구조 :

App.vue

메인화면

무엇을
컴포넌트화 해서
첨부할것임?

vuestagram - App.vue 템플릿

```
<div class="header">
  <ul class="header-button-left">
    <li>Cancel</li>
  </ul>

  <ul class="header-button-right">
    <li>Next</li>
  </ul>

  
</div>

<Body />

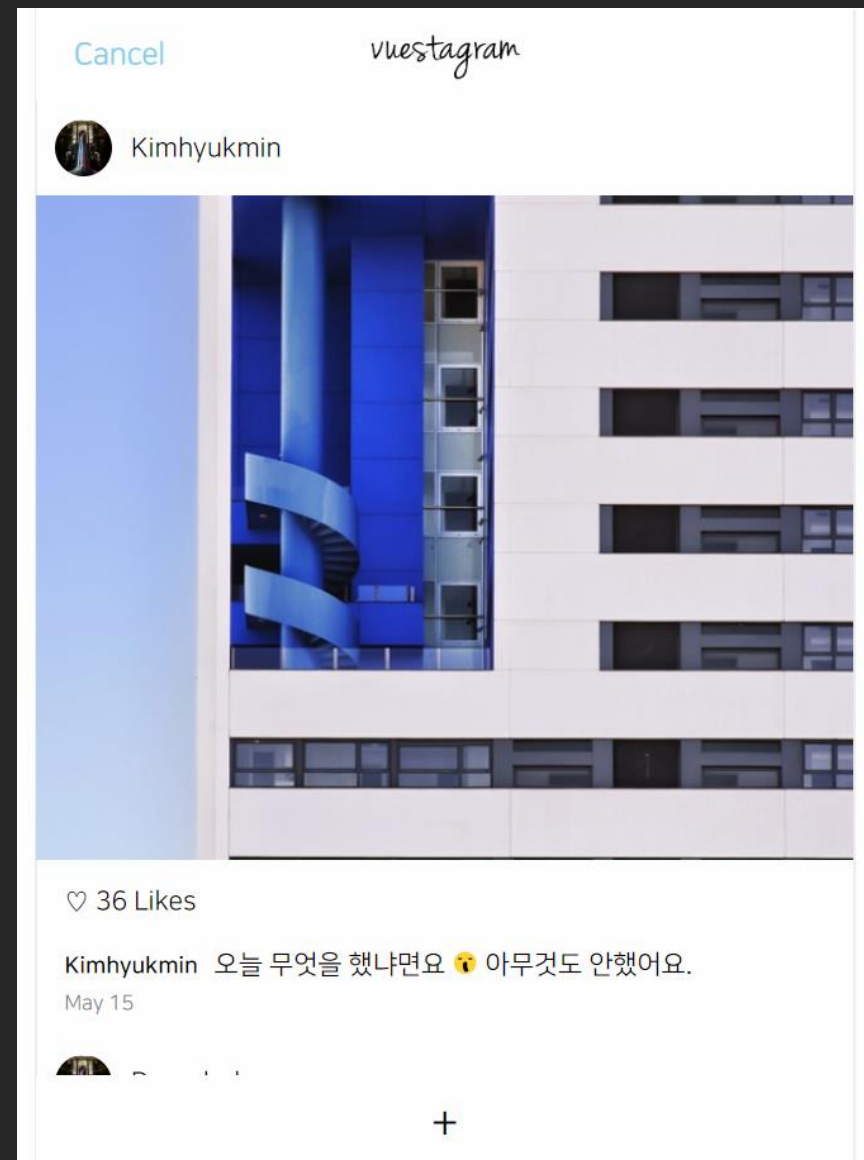
<div class="sample-box">임시 박스</div>

<div class="footer">
  <ul class="footer-button-plus">
    <li>+</li>
  </ul>
</div>
```

```
body {
  margin: 0;
}
ul {
  padding: 5px;
  list-style-type: none;
}
.logo {
  width: 22px;
  margin: auto;
  display: block;
  position: absolute;
  left: 0;
  right: 0;
  top: 13px;
}
.header {
  width: 100%;
  height: 40px;
  background-color: white;
  padding-bottom: 8px;
  position: sticky;
  top: 0;
}
.header-button-left {
  color: skyblue;
  float: left;
  width: 50px;
  padding-left: 20px;
  cursor: pointer;
  margin-top: 10px;
}
.header-button-right {
  color: skyblue;
  float: right;
  width: 50px;
  cursor: pointer;
  margin-top: 10px;
}
```

```
.footer {
  width: 100%;
  position: sticky;
  bottom: 0;
  padding-bottom: 10px;
  background-color: white;
}
.footer-button-plus {
  width: 80px;
  margin: auto;
  text-align: center;
  cursor: pointer;
  font-size: 24px;
  padding-top: 12px;
}
.sample-box {
  width: 100%;
  height: 600px;
  background-color: bisque;
}
.inputfile {
  display: none;
}
.input-plus {
  cursor: pointer;
}
#app {
  box-sizing: border-box;
  font-family: 'consolas';
  margin-top: 60px;
  width: 100%;
  max-width: 460px;
  margin: auto;
  position: relative;
  border-right: 1px solid #eee;
  border-left: 1px solid #eee;
}
```

우리가 만들 앱의 구조 :



vuestagram – Body.vue 템플릿

```
<template>

<div class="body">
  <Post />
  <Post />
  <Post />
</div>

</template>
```

vuestagram – Post.vue 템플릿

```
<template>

<div class="post">
  <div class="post-header">
    <div class="profile"></div>
    <span class="profile-name">ChanKim</span>
  </div>

  <div class="post-body"></div>

  <div class="post-content">
    <p>43 Likes</p>
    <p><strong>글쓴이아이디</strong> 임시내용이에요🐼</p>
    <p class="date">May 15</p>
  </div>
</div>

</template>
```

```
.post {
  width: 100%;
}
.profile {
  background-image: url('https://placeimg.com/100/100/arch');
  width: 30px;
  height: 30px;
  background-size: 100%;
  border-radius: 50%;
  float: left;
}
.profile-name {
  display: block;
  float: left;
  padding-left: 10px;
  padding-top: 7px;
  font-size: 14px;
}
.post-header {
  height: 30px;
  padding: 10px;
}
.post-body {
  background-image:
    url('https://placeimg.com/640/480/animals');
  height: 350px;
  background-position: center;
  background-size: cover;
}
.post-content {
  padding-left: 15px;
  padding-right: 15px;
  font-size: 14px;
}
.date {
  font-size: 11px;
  color: grey;
  margin-top: -8px;
}
```

[Post.vue에 실제 데이터 바인딩하기]

App.vue

Body.vue

Post.vue

0. 포스팅 데이터 3개는 `postdata.js`에 준비됨 `[{}, {}, {}]`

1. App.vue 데이터에 `postdata.js` 넣기

2. 데이터를 App.vue -> Body.vue에 전해주려면?

3. Body.vue 에선 `<Post/>`를 3개 만들어내야함.
각각의 Post에 `데이터[0],[1],[2]`를 전해줘야함

4. Post.vue 내부에선 받아온 object 데이터를 `{{바인딩}}`


postData.js

```
export default [
  {
    name: "Kimhyukmin",
    userImage: "https://placeimg.com/100/100/arch",
    postImage: "https://placeimg.com/640/480/arch",
    likes: 36,
    date: 'May 15',
    liked: false,
    caption: "오늘 무엇을 했냐면요 ☹ 아무것도 안했어요.",
    filter: "perpetua"
  },
  {
    name: "Dangdud",
    userImage: "https://placeimg.com/200/200/arch",
    postImage: "https://placeimg.com/640/480/people",
    likes: 20,
    date: 'Apr 20',
    liked: false,
    caption: "Do you even lift, bro?",
    filter: "clarendon"
  },
  {
    name: "Mungmung",
    userImage: "https://placeimg.com/100/100/animals",
    postImage: "https://placeimg.com/640/480/animals",
    likes: 49,
    date: 'Apr 4',
    liked: false,
    caption: "고양이 멍멍이",
    filter: "lofi"
  }
]
```

for 반복문으로 HTML 만드는 법

```
<ul>  
  <li v-for="상품 in 상품들">{{ 상품 }} </li>  
</ul>
```

```
export default {  
  data() {  
    return {  
      상품들: ['갤럭시', '아이폰', 'LG']  
    }  
  }  
}
```



CSS 혹은 스타일에 데이터(변수)를 집어넣는 법

```
<ul>  
  <li v-bind:style="{ fontSize: 사이즈 }"> 어쩌구 </li>  
</ul>
```

```
export default {  
  data() {  
    return {  
      사이즈: '16px'  
    }  
  }  
}
```

```
<List v-bind:내바지="아빠바지" />
```

1. v-bind:props이름="전해줄데이터"

```
import List from './components/List.vue'
```

```
export default {  
  data() {  
    아빠바지: '슬랙스'  
  },  
  components : {  
    List,  
  }  
}
```

```
<p> {{내바지}} </p>
```

2. 받아온 props 자유롭게 쓰기

```
export default {  
  props : {  
    내바지 : String  
  }  
}
```

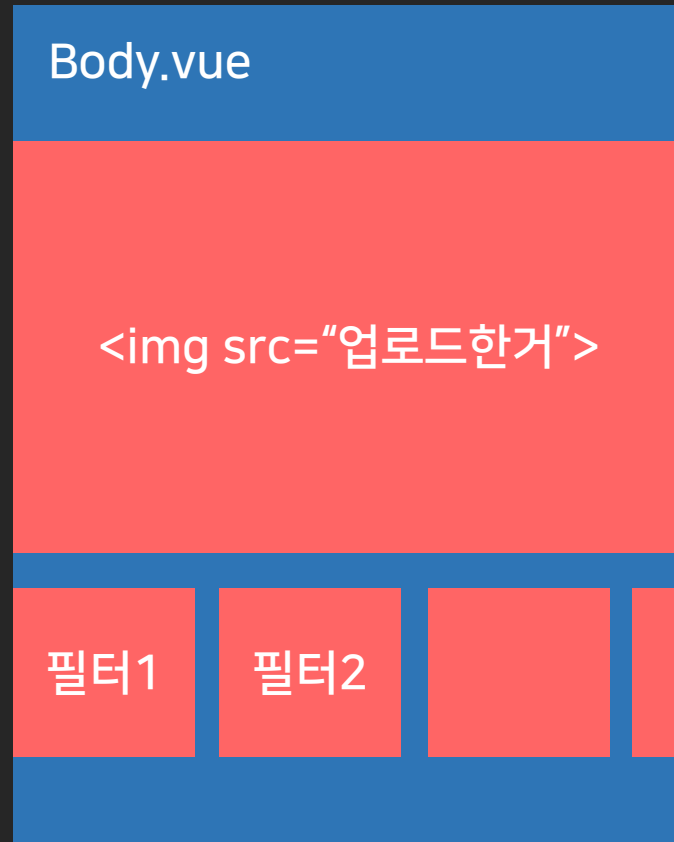
1. 받아올 props이름과 데이터형 명시해주기

Body.vue 구조

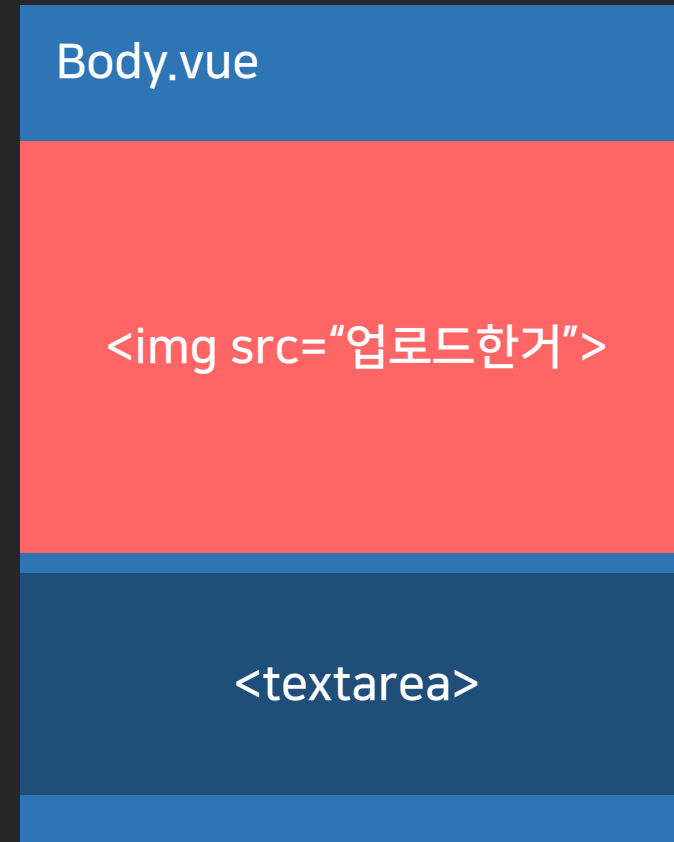
메인화면



업로드한 사진의 필터 선택



글 입력

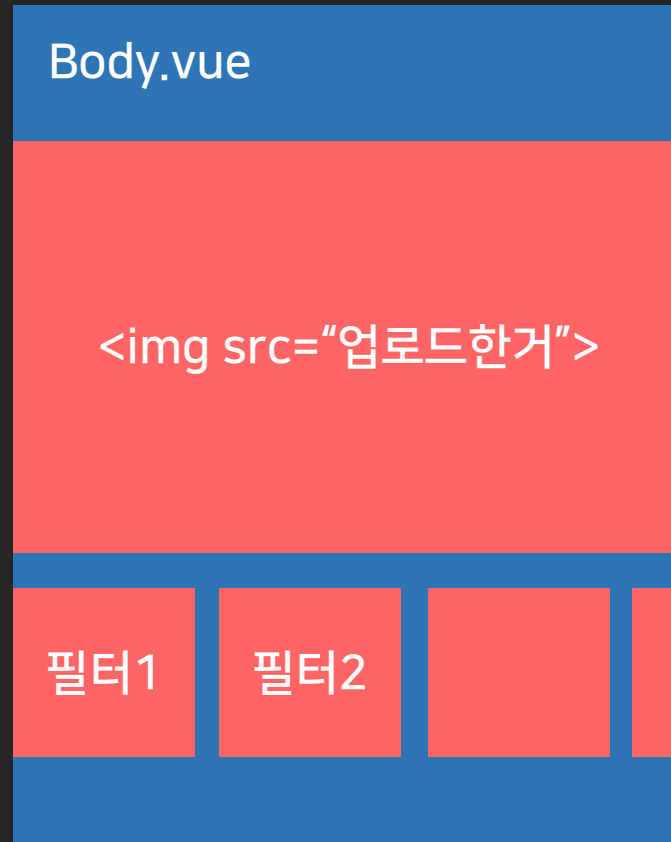


Body.vue 구조 : router 말고 v-if 에 따른 페이지 구분

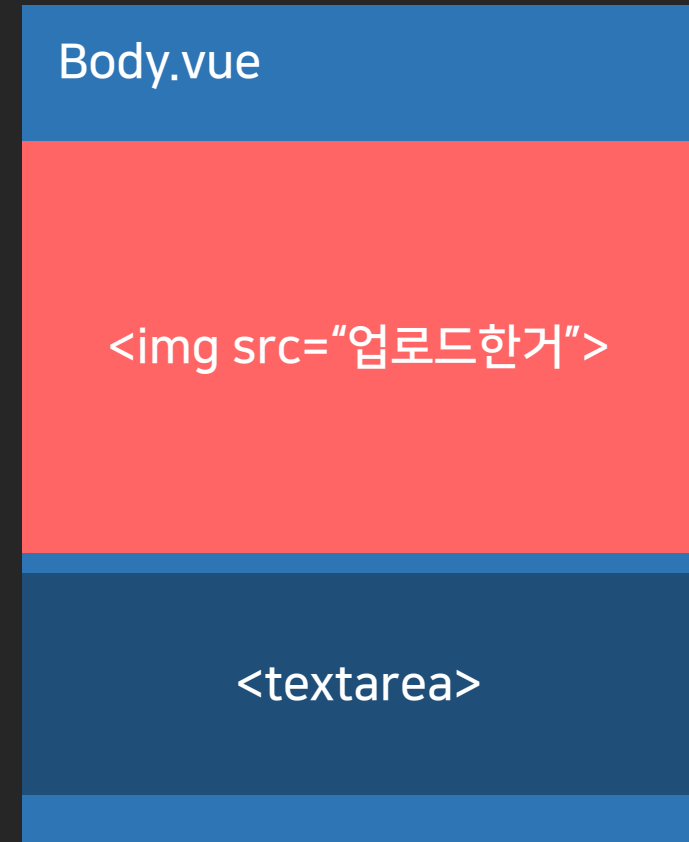
만약 step이 1일 경우



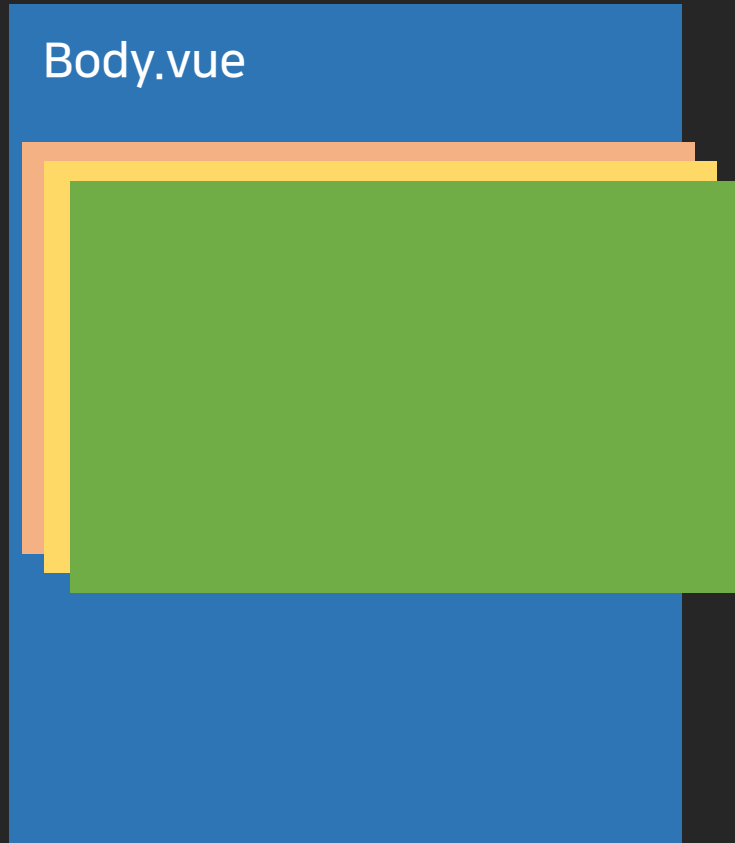
만약 step이 2일 경우



만약 step이 3일 경우



v-if를 이용한 페이지 나누기



만약에 step이 1이면.. Post보여줌
만약에 step이 2이면.. 이미지와 필터선택화면 보여줌
만약에 step이 3이면.. 이미지와 글쓰기 입력란 보여줌

Router/component 가 아니라 v-if를 이용해 페이지를 나눈 이유?

- 데이터 관리의 용이성 (입력받는 데이터가 3종류인데.. 각각 다른 컴포넌트에 있으면 관리가 어렵잖소)


```
<template>
<div class="body">
```

[<Post /> **포스트**

[<div class="upload-image"></div>
 <div class="filters">
 <div class="filter-1"></div>
 <div class="filter-1"></div>
 <div class="filter-1"></div>
 <div class="filter-1"></div>
 <div class="filter-1"></div>
 </div>

[<div class="upload-image"></div>
 <div class="write">
 <textarea class="write-box">write!</textarea>
 </div>

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

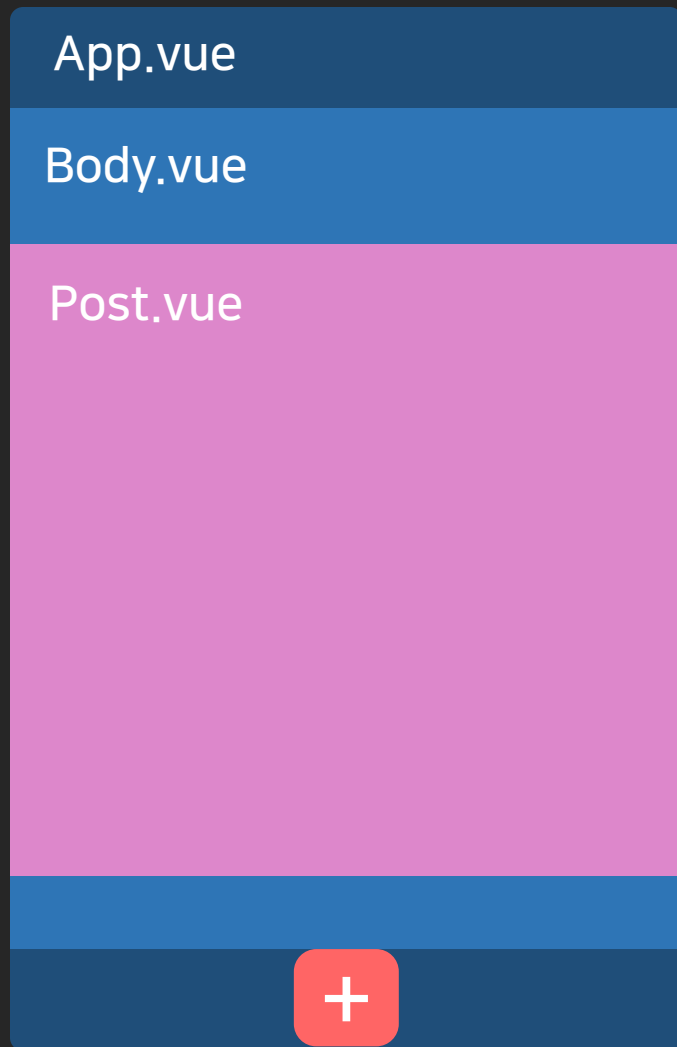
이미지 & 필터고르기화면

이미지 & 글입력

vuestagram – Body.vue 템플릿

```
.upload-image{
width: 100%;
height: 350px;
background: cornflowerblue;
}
.filters{
overflow-x:scroll;
white-space: nowrap;
}
.filter-1 {
width: 100px;
height: 100px;
background-color: cornflowerblue;
margin: 10px 10px 10px auto;
padding: 8px;
display: inline-block;
color : white;
background-size: cover;
}
.filters::-webkit-scrollbar {
height: 5px;
}
.filters::-webkit-scrollbar-track {
background: #f1f1f1;
}
.filters::-webkit-scrollbar-thumb {
background: #888;
border-radius: 5px;
}
.filters::-webkit-scrollbar-thumb:hover {
background: #555;
}
.write-box {
border: none;
width: 90%;
height: 100px;
padding: 15px;
margin: auto;
display: block;
outline: none;
}
```

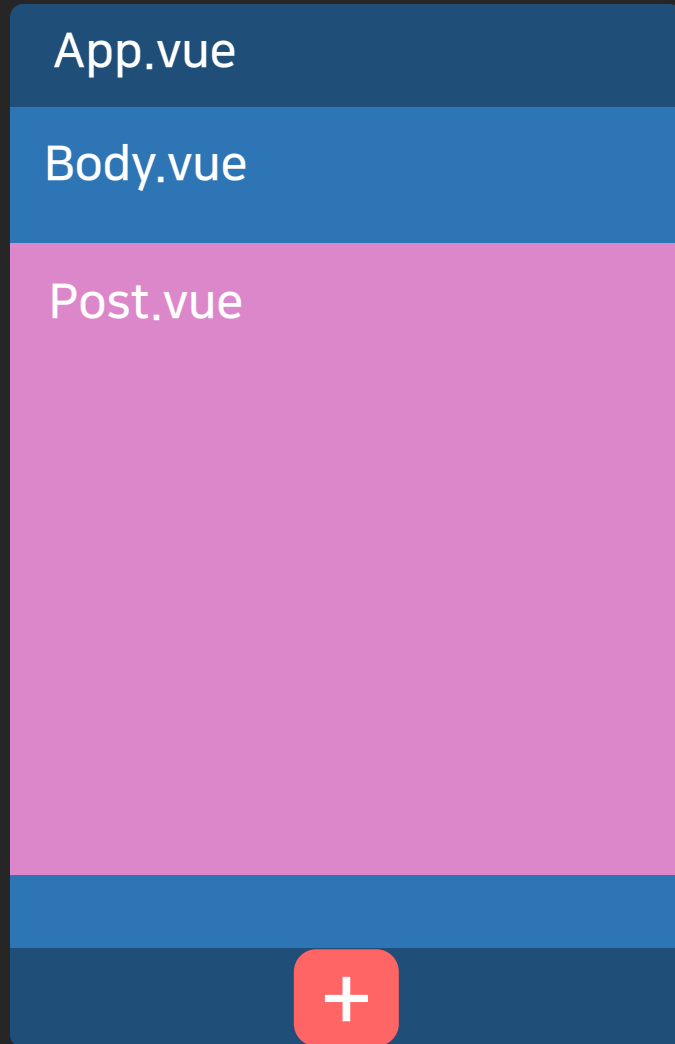
[App.vue 하단에 이미지 업로드 버튼 만들기]



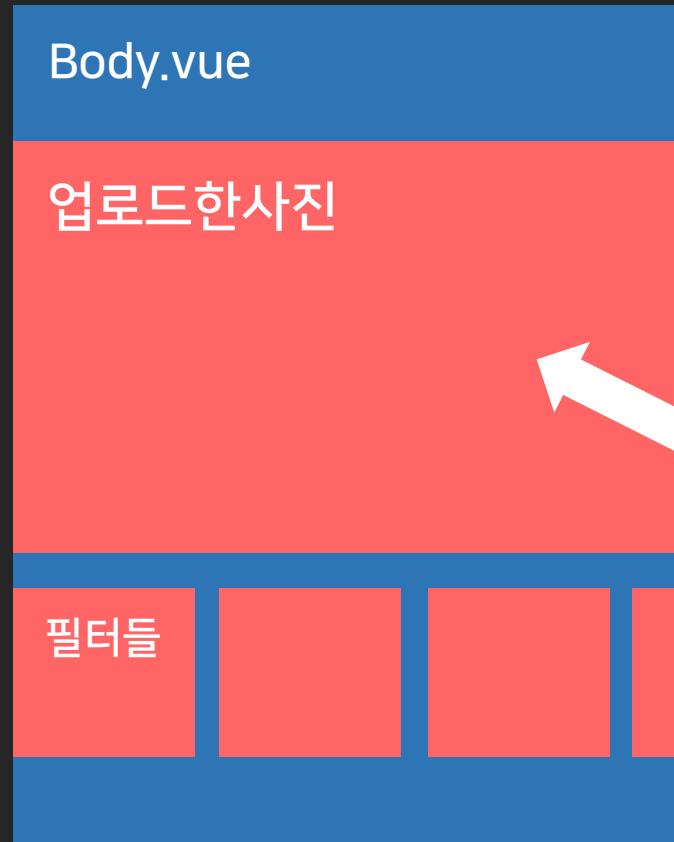
업로드 버튼을 누르면 어떤일이?

- 1.
- 2.
- 3.

[App.vue 하단에 이미지 업로드 버튼 만들기]



step 2




1. App.vue의 + 업로드 버튼을 누르면
2. 업로드한 이미지의 **경로**를 알아냄
3. **경로**를 App.vue의 data에 저장
4. 바로 step 2로 이동
5. 저장한 **경로** 데이터를 이용해 step2에서 사진 보여주기 (background-image로 첨부)

[App.vue 하단에 이미지 업로드 버튼 만들기]

1. 업로드 input 버튼을 만드는 법

```
<input type="file" id="file" class="inputfile" >  
<label for="file" class="input-plus">+</label>
```

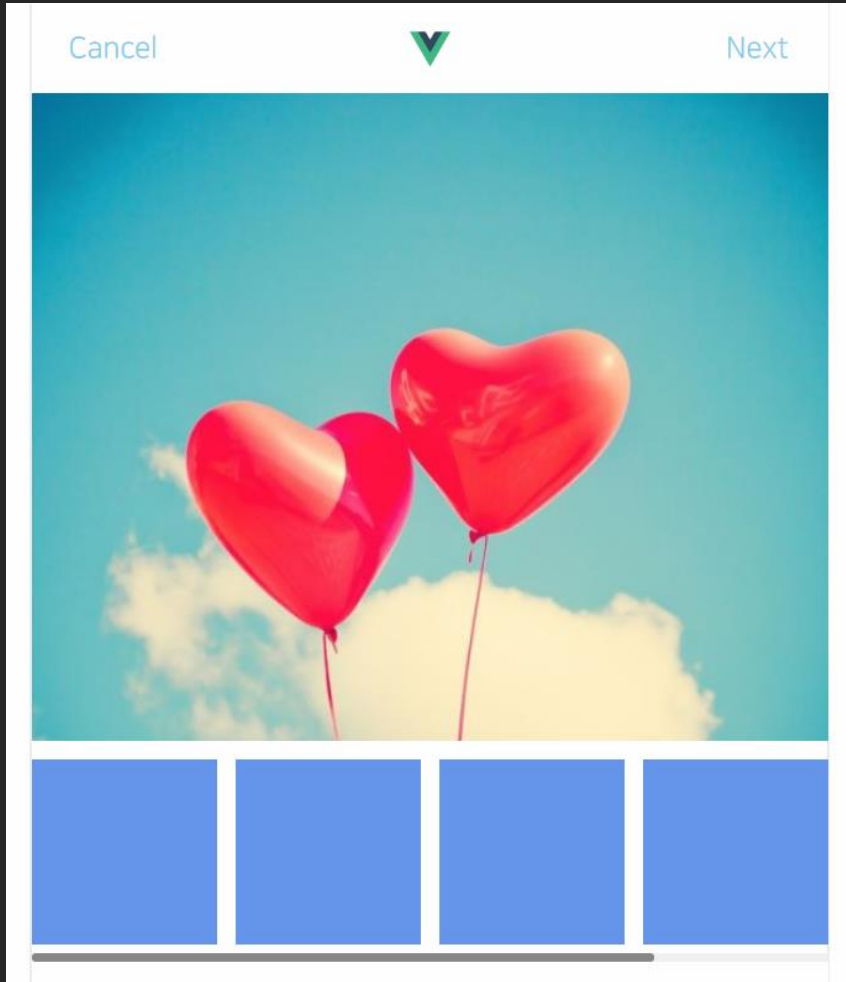
2. 업로드한 이미지의 경로를 알아내는 법

```
let file = e.target.files;  
let reader = new FileReader();  
reader.readAsDataURL(file[0]);  
reader.onload = e => {  
  console.log(e.target.result)  경로  
}
```

그 다음 3, 4, 5번은 배운 것이니 알아서해보자!

폼 이벤트는 `v-on:change="??"`

[이미지 업로드 누를 시 결과물]



```
v-bind:style="{ backgroundImage: 'url('이미지경로')' }"
```

[Next 버튼만들기]

next버튼 X

next버튼 O, 그리고 버튼의 기능은?

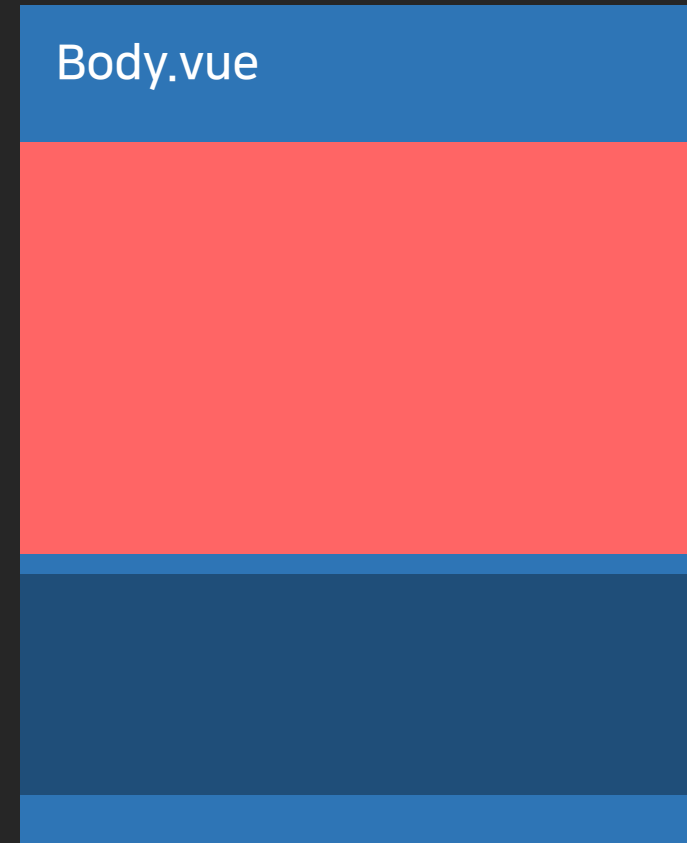
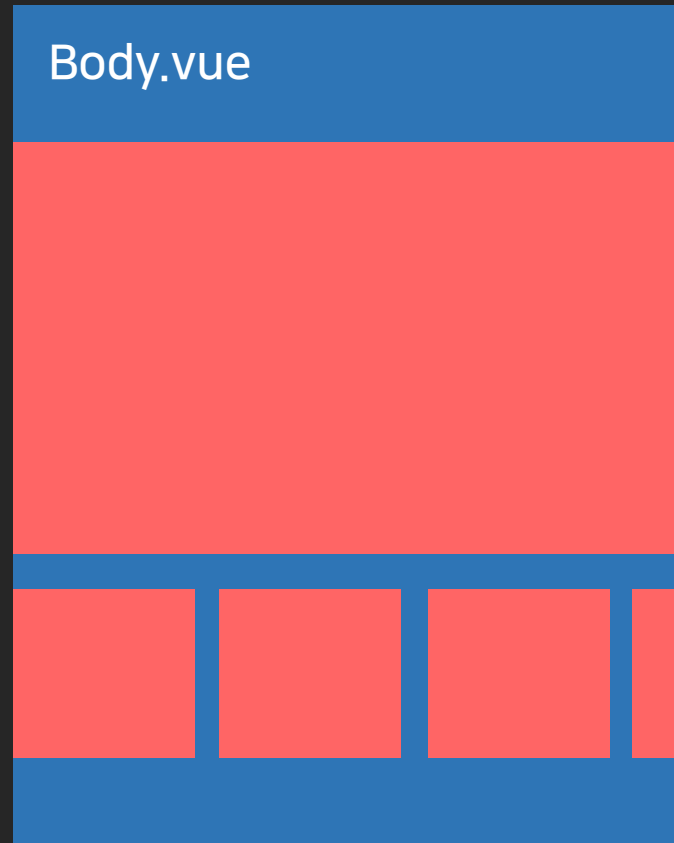
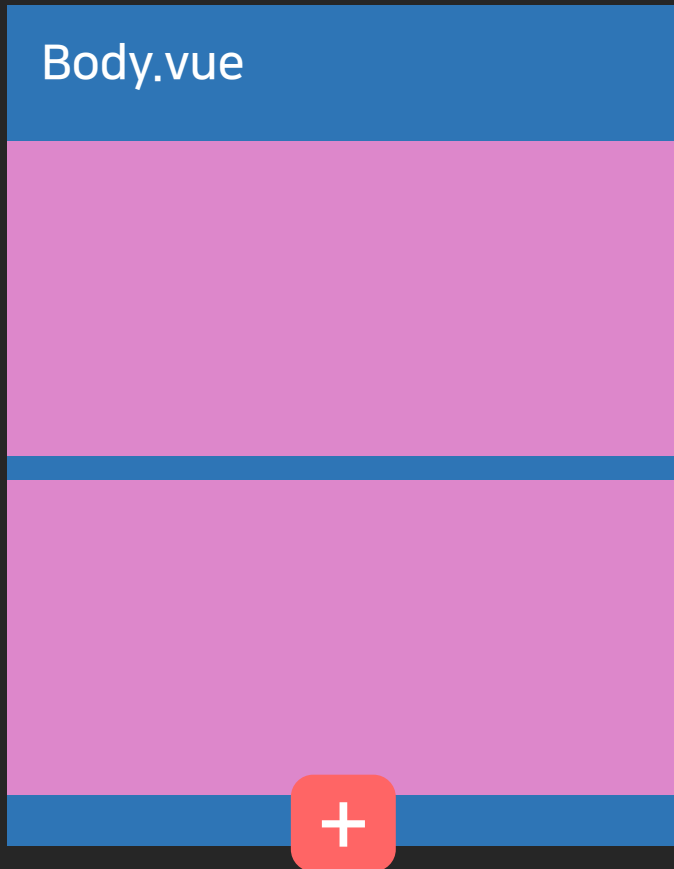
만약 step 1일 경우



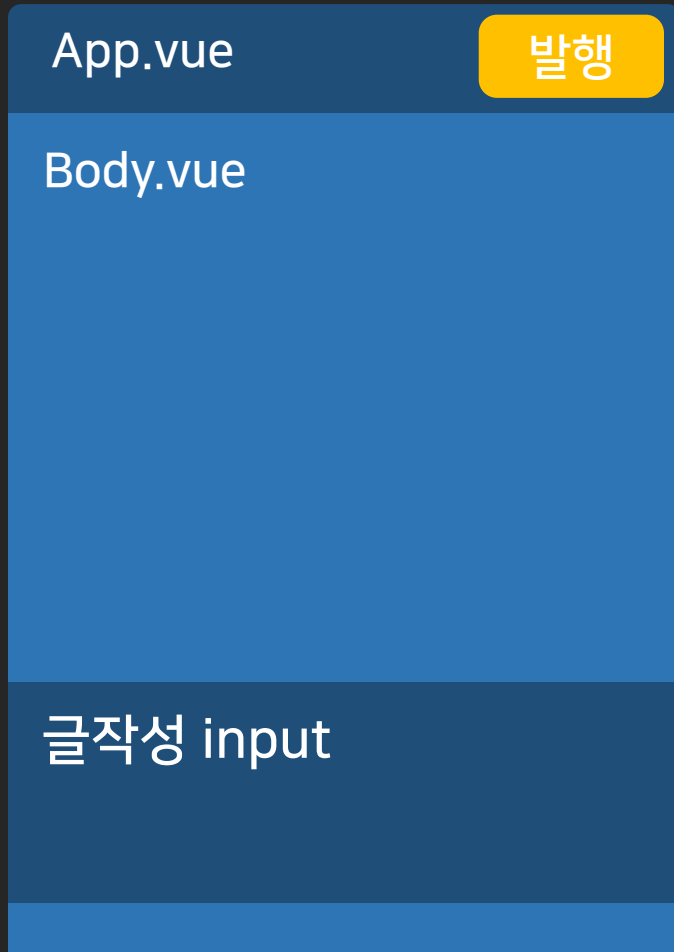
만약 step 2일 경우



만약 step 3일 경우



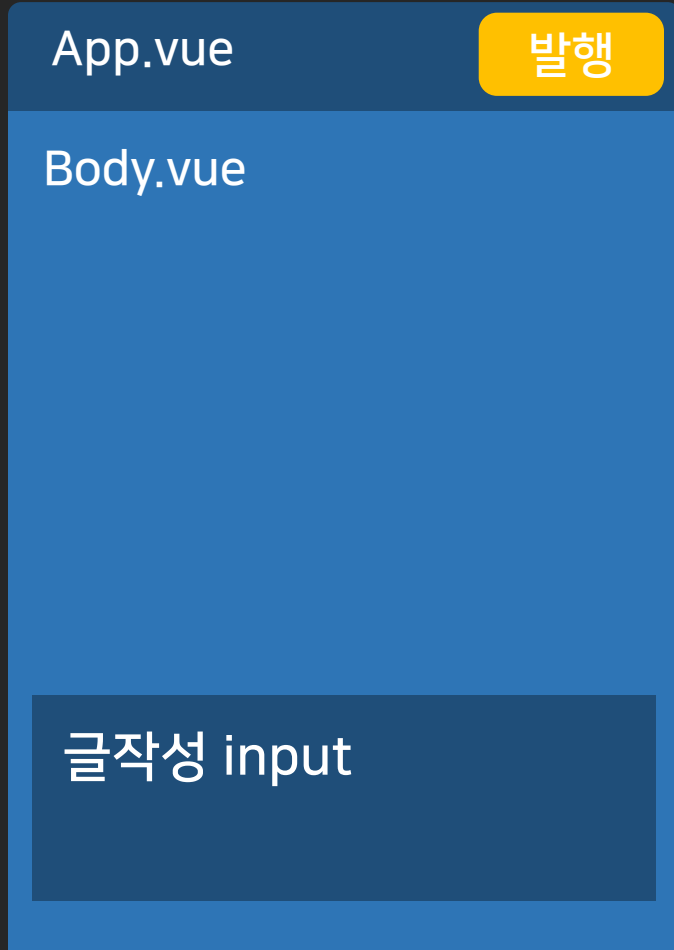
[글 발행 버튼 & 기능만들기]



글 발행기능을 만들려면?

- 1.
- 2.
- 3.
- 4.

[글 발행 버튼 & 기능만들기]



1. step3일 경우 '발행' 버튼이 보여야함.

- 발행을 누르면

2. 나의 글과 이미지가 담긴 이쁜 {포스트데이터} 만들기

3. App.vue의 posts 데이터에 나의 {포스트데이터}를 추가함

- 글은 어디서 나는것?

4. Body.vue에서 작성한 글 데이터가 App.vue 까지 전해져야함

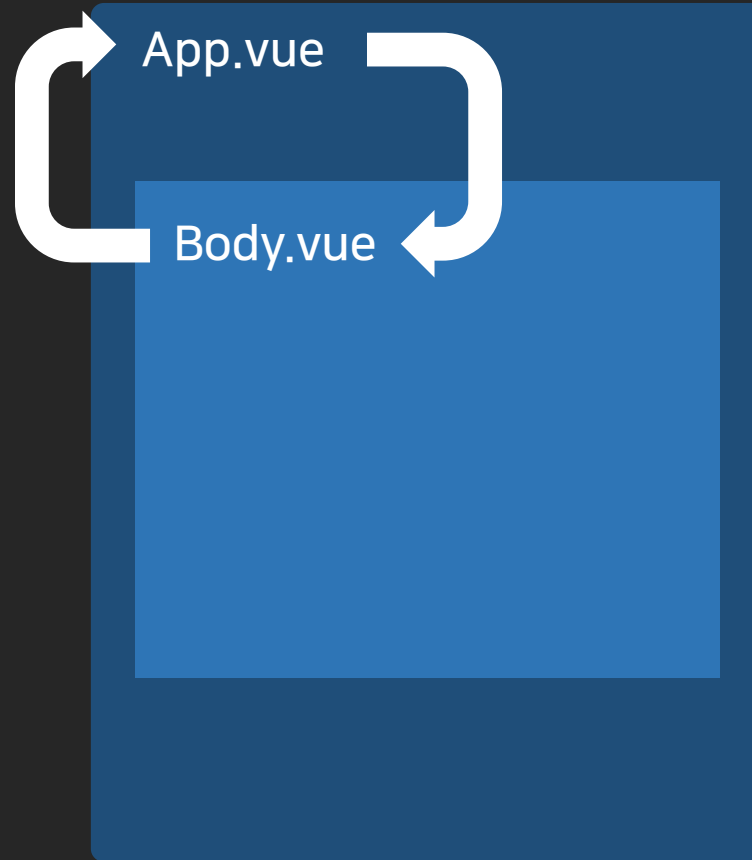
- 이미지는 어디서 나는것?

5. 전에 이미지 경로 어딘가에 저장하지 않았습니까

꼭 알아야할

데이터 전달하는 궁극의 법칙

부모에게 데이터 전달은
커스텀 event로



자식에게 데이터 전달은
Props로

Props 사용법 :

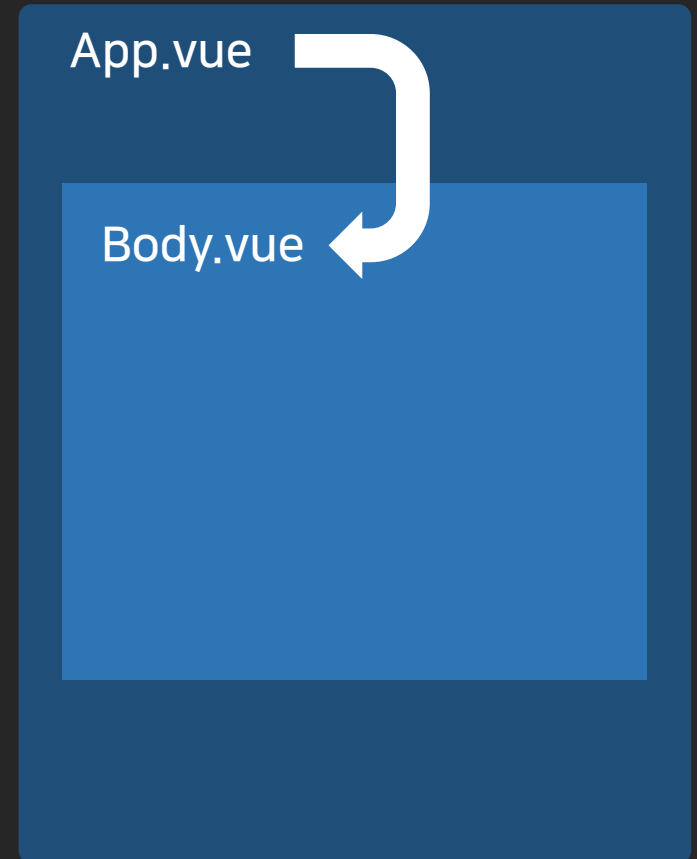
1. 부모가 컴포넌트 첨부할 때 속성란에
<v-bind:props이름 = 데이터>

```
<child v-bind:height="우월한키" />
```

2. 자식은 받아올 'props이름'과 '자료형' 명시해주기,
그리고 데이터처럼 활용

```
props: {  
  height : String  
},
```

자식에게 데이터 전달은
Props로



커스텀 event 사용법 :

1. 자식은 v-on:click="\$emit('이벤트이름', 전해줄값)"

```
<button v-on:click="$emit('hyodo', '50000')">
```

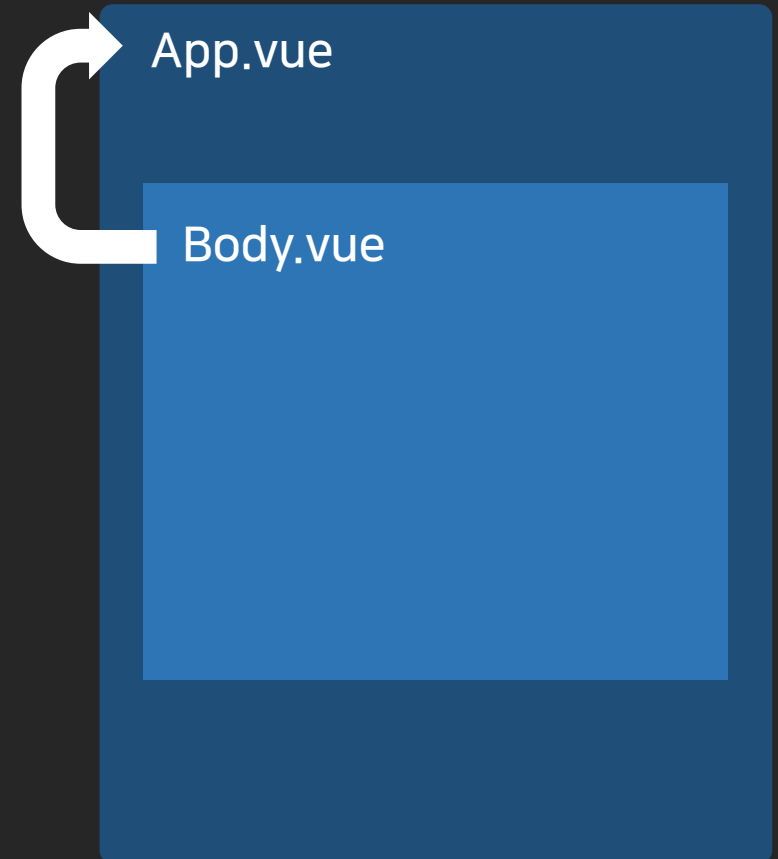
- 부모에게 전달할 나만의 예쁜 이벤트 이름, 그리고 값을 정해주면 됩니다
- event 작성시 kebab-case

2. 부모는 자식 컴포넌트 첨부하는 곳에서 :
커스텀 이벤트 수신시 실행할 코드 적어놓기

```
<child v-on:hyodo="지갑 = $event"/>
```

- 전해준 값(5만원)은 \$event라고 쓰면 수신 가능합니다.
- 수신한 데이터는 data(){}에 저장하고 이런 식이면 되겠죠?

부모에게 데이터 전달은
커스텀 event로



커스텀 event 사용법 2 : input에 입력한 값을 전해주려면?

1. 자식은 `v-on:click="$emit('이벤트이름', 전해줄값)"`

```
<input v-on:input="$emit('hyodo', $event.target.value)" >
```

input에 입력된 값은 `$event.target.value`라고 쓰면 가져올 수 있습니다.

2. 부모가 자식 첨부하는 곳에서 :
커스텀 이벤트 수신시 실행할 코드 적어놓기

`v-on:hyodo="어쩌구"` 여긴 뭐 똑같음

CSSgram을 이용한 필터고르기 기능

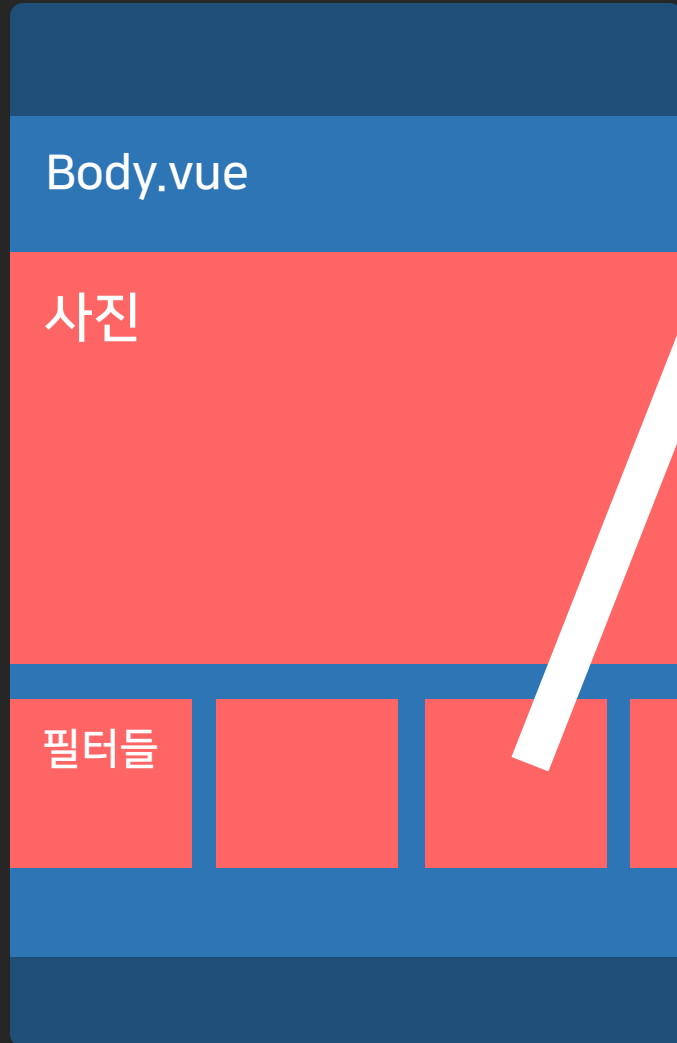


누군가 만들어놓은 CSSgram 라이브러리를 이용해서
인스타그램과 똑같은 필터효과 적용가능

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/cssgram/0.1.10
/cssgram.min.css">
(index.html에 복사)
```

CSSgram을 이용한 필터고르기 기능

step 2



필터들을 FilterBox.vue를 이용해서
<FilterBox />로 첨부합시다

```
<div class="filter-item"> </div>
```

```
.filter-item {  
  width: 100px;  
  height: 100px;  
  margin: 10px 10px 10px auto;  
  padding: 8px;  
  display: inline-block;  
  color : white;  
  background-size: cover;  
  background-position : center;  
}
```

CSSgram을 이용한 필터고르기 기능

▼App.vue의 데이터로 저장하세요

```
[ "normal", "clarendon", "gingham", "moon", "lark", "reyes", "juno",  
  "slumber", "aden", "perpetua", "mayfair", "rise", "hudson", "valencia",  
  "xpro2", "willow", "lofi", "inkwell", "nashville"]
```

그리고 이 필터 개수만큼 <FilterBox />를 생성해야하는데 어떻게 하죠?

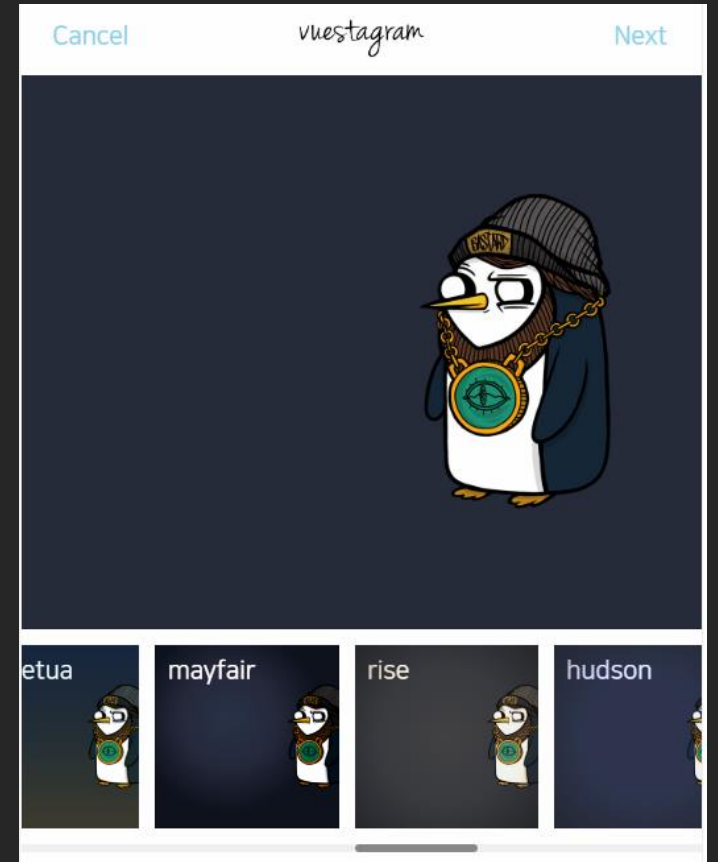
```
<div class="filters">  
  <FilterBox />  
</div>
```

CSSgram을 이용한 필터고르기 기능

19개의 **필터이름**을 각각의 FilterBox.vue 안에 전달해서

1. 네모박스안의 글씨를 **필터이름**으로 {{데이터바인딩}}

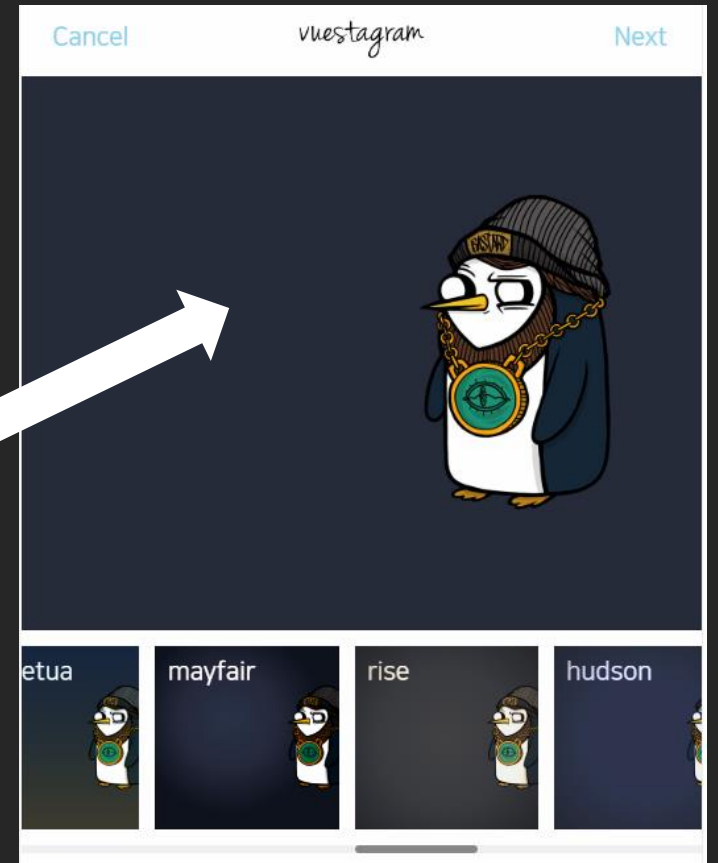
2. class이름을 **필터이름**으로 {{데이터바인딩}}



선택한 필터로 업로드 사진 꾸미기

1. FilterBox안의 div박스를 누르면,
현재 누른 필터 이름이 **App.vue**까지 전해져야함.
(Bus 태우기)

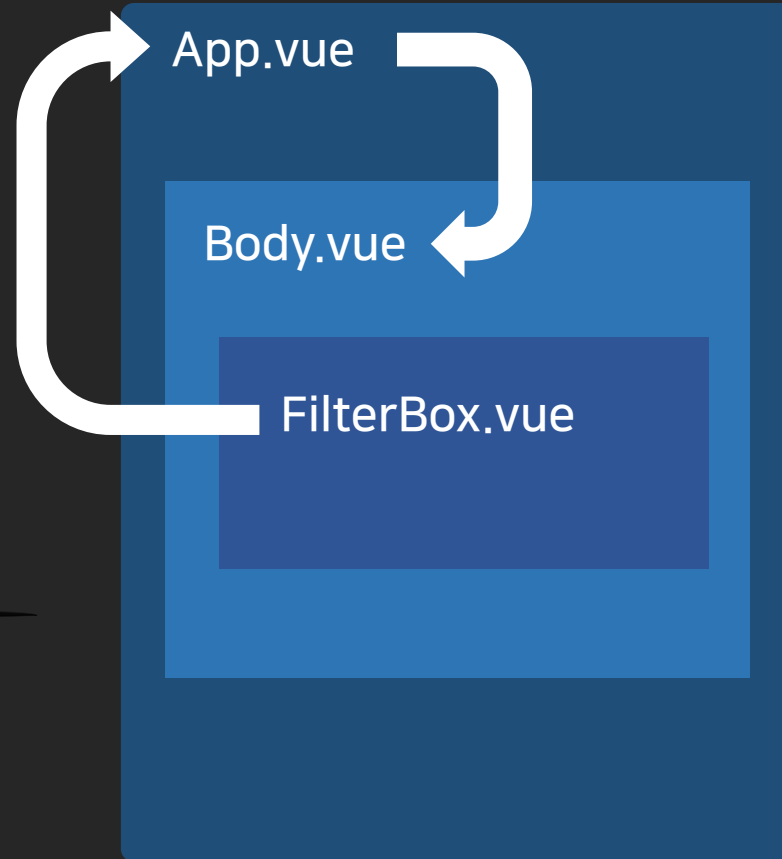
2. 그리고 Body.vue는 이걸 props로 전달받아서
업로드한 이미지를 꾸밈 (class를 추가함)



알아둬야할

데이터 전달하는 궁극의 법칙2

조부모 이상에게 데이터 전달은
EventBus로



자식에게 데이터 전달은
Props로

이벤트버스로 커스텀 이벤트를 만드는법 :

EventBus.js

파일 아무데나 만들고 (src폴더 추천)

```
import Vue from "vue";  
export default new Vue()
```



자식.vue

```
EventBus.$emit('select-filter', 보낼자료);
```

EventBus.js를 import한 뒤에
보낼자료를 select-filter 이벤트에 실어보냄

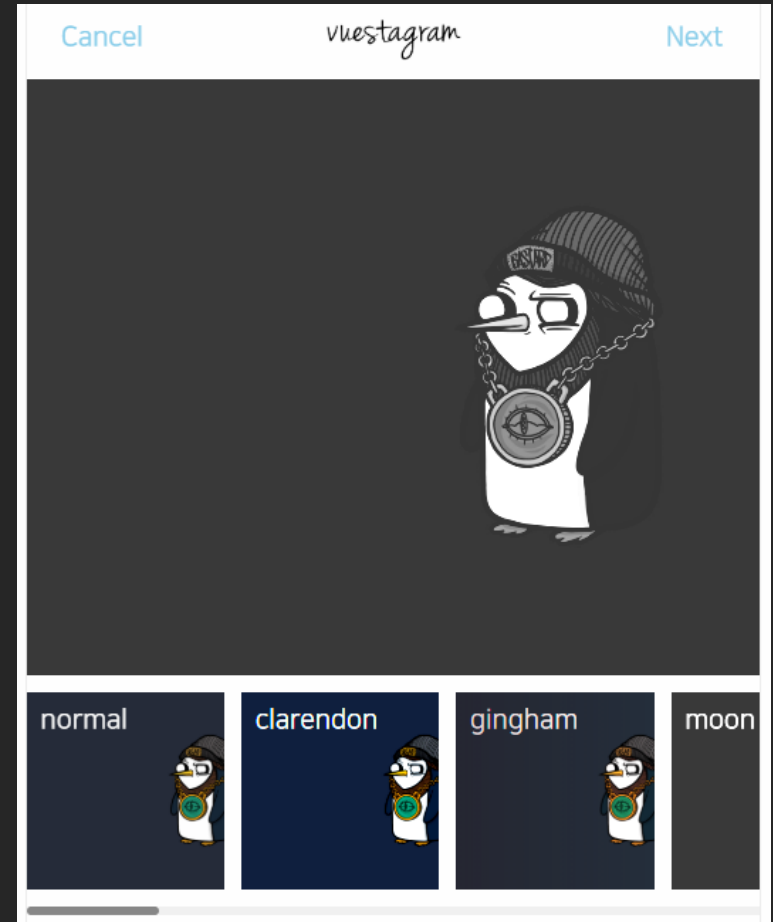
부모.vue

```
EventBus.$on('select-filter', (받은자료) => {  
  console.log(받은자료)  
});
```

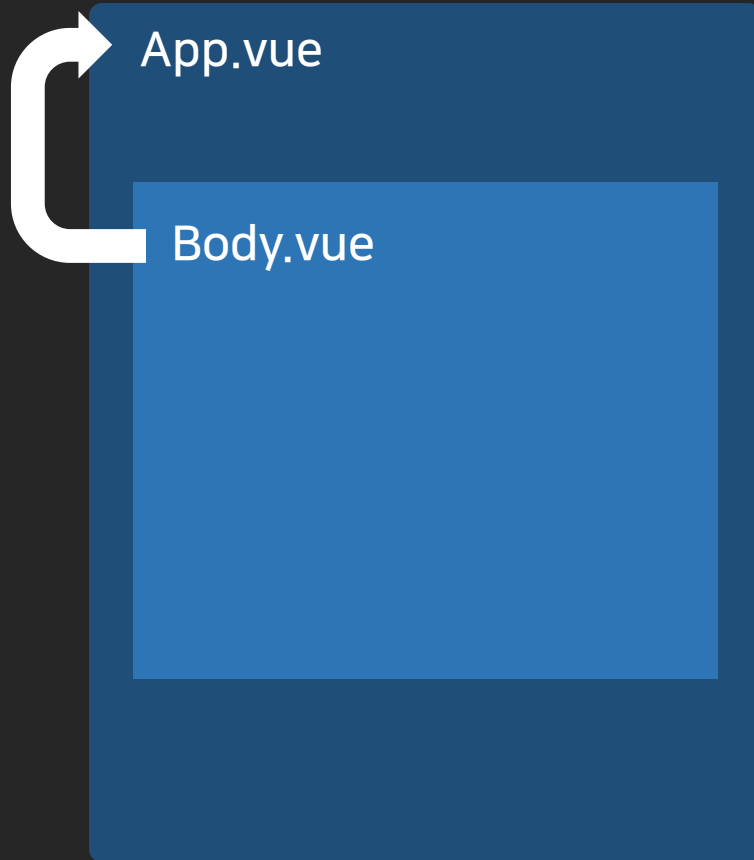
EventBus.js를 import한 뒤에
select-filter 이벤트가 일어나면 자료받아서 어찌저찌함
(버스수신은 mounted()안에 보통 많이 작성)

1. FilterBox안의 div박스를 누르면,
현재 누른 필터 이름이 **App.vue**까지 전해져야함.

2. 필터이름 데이터를 물려받아서
업로드한 이미지를 꾸밈 (class를 추가함)



v-model 을 컴포넌트에다가 쓰면?



부모 컴포넌트에
자식에서 입력한 값을
데이터바인딩하는 방법 3개 :

1. 커스텀이벤트
2. 이벤트버스 (조부모)
3. 커스텀이벤트와 v-model

v-model 의 용도 1.

input 역방향 데이터바인딩

사용자가 입력한 값을
데이터로 저장해줌

```
<child v-on:hyodo="지갑 = $event"/>
```

커스텀 이벤트와 함께 v-model 써도 나름 가능

```
<child v-model="지갑"/>
```

v-model 의 용도 2.

컴포넌트 데이터바인딩

자식 컴포넌트에서 입력한 값을
부모 컴포넌트의 데이터로 저장해줌

<https://vuejs.org/v2/guide/components.html#Using-v-model-on-Components> 안중요하니 자세한내용은 여기 참고

[Slots를 이용해 컴포넌트 개발하기]

Props : 하위컴포넌트에게 전해주는 data()

Slots : 하위컴포넌트에게 전해주는 <HTML> 또는 <Component>

Slots 쓰는 이유 :

1. Flexible한 하위 컴포넌트 개발 가능
2. 컴포넌트.vue가 깔끔해짐 (Props 정의 안해도 되니까)

[Slots 를 이용한 컴포넌트 개발]

Child.vue

```
<div>  
  <slot></slot>  
</div>
```



부모가 집어넣을 <HTML> 공간을 마련해줌

App.vue

```
<Child>  
  <p>부모가 하위 컴포넌트에 집어넣을 HTML은 여기에!</p>  
</Child>
```


[Slots vs Props : 뭐가 더 좋아보임?]

Child.vue

```
<div>  
  <slot></slot>  
</div>
```

App.vue

```
<Child>  
  <p>부모가 집어넣을 HTML!</p>  
</Child>
```

Child.vue

```
<div>  
  <p>{{ text }}</p>  
</div>
```

```
props : { text : String }
```

App.vue

```
<Child v-bind:text="데이터"></Child>
```

```
데이터 : '안녕하세요'
```

[더보기 버튼 (간단한 Ajax 요청)]

Ajax : 새로그침없이 서버와 GET/POST 통신을 할 수 있게 도와줌

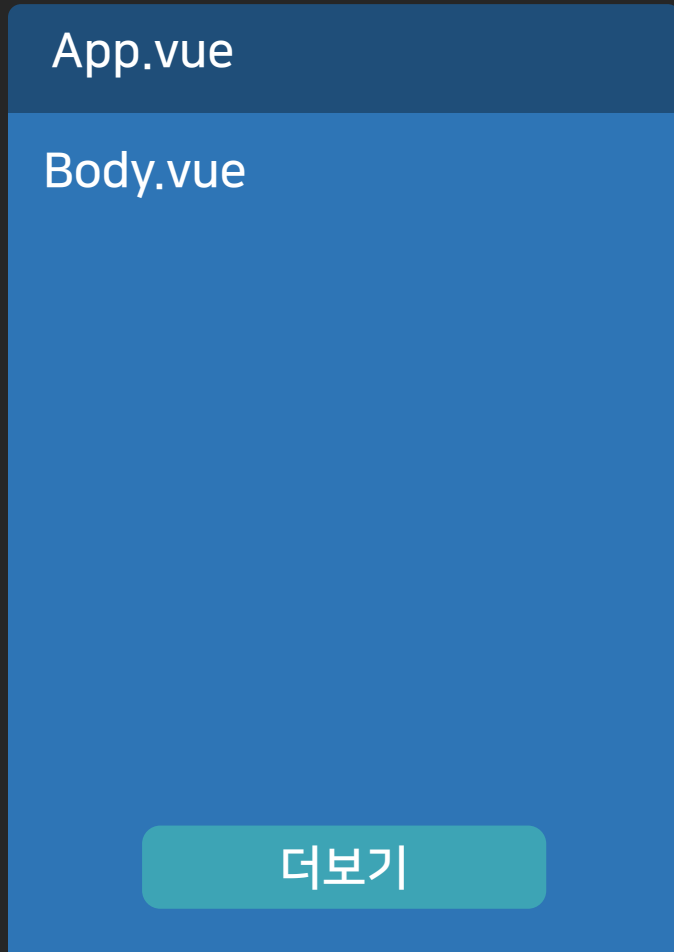
Axios : Promise 신문법을 이용해 Ajax요청 문법을 쉽게 도와주는 라이브러리

```
yarn add axios
```

Ajax 데이터 준비 : public 폴더에 postdata2.json 준비

(여기다 넣으면 사이트 발행할 때 항상 같이 딸려옴)

[더보기 버튼 (간단한 Ajax 요청)]



1. 더보기 버튼을 누르면
2. 어딘가의 URL에 Ajax 요청을해서
3. 받아온 JSON 데이터를
4. Body.vue의 data에 추가하면 되지 않을까요?

[더보기 버튼 (간단한 Ajax 요청)]

요청시 필요한 사항 :

1. 어디로 요청할 것인지 (URL)
2. GET인지 POST인지
3. 데이터 받아오면 뭐할 건지
4. 못받고 에러나면 뭐할 건지

[GET요청 하는 법 (데이터 가져오기)]

```
import axios from 'axios';
```

```
axios.get('postData2.json')  
  .then(결과 => {  
    console.log(결과.data)  
  })
```

1. 요청방식과 URL

2. then(성공시 실행할 코드)

[GET요청 하는 법 (데이터 가져오기)]

```
import axios from 'axios';
```

```
axios.get('postData2.json')
```

```
.then( 결과 => {
```

```
    console.log(결과.data) 
```

```
})
```

결과.data

결과.status

결과.headers

수신한 결과엔 여러가지 정보가 담김

[GET요청 하는 법 (데이터 가져오기)]

```
import axios from 'axios';
```

```
axios.get('postData2.json')  
  .then( 결과 => {  
    console.log(결과.data)  
  }).catch( 에러 => {  
    console.log(에러)  
  })
```

3. catch(실패시 실행할 코드)

[POST 요청 하는 법 (서버로 데이터 전송)]

```
axios.post(`http://example.com/posts`, {  
  data: this.post  
})  
  .then( 결과 => {  
    //  
  })  
  .catch( 에러 => {  
    //  
  })
```


[IE9+ 에러 : Promise가 없는데요? 해결법]

1. 스크립트태그로 index.html에 첨부하거나

```
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.min.js"></script>  
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.min.js"></script>
```

1. 플러그인으로 설치해서 main.js에 import

```
yarn add es6-promise
```

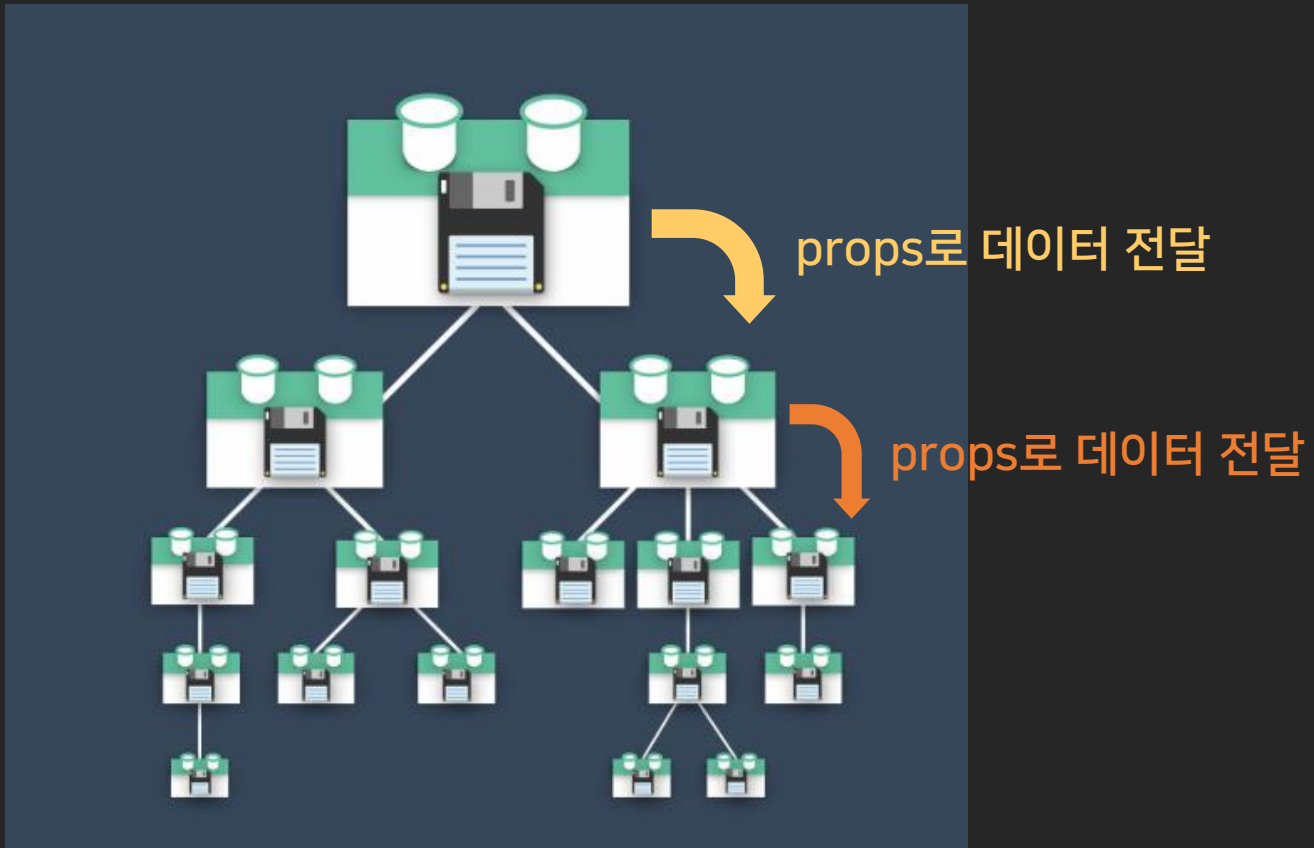
```
import 'es6-promise/auto'
```

Vuex

“props랑 event로 데이터 주고받기 개힘드네” 라는 생각이 들 때 쓰자

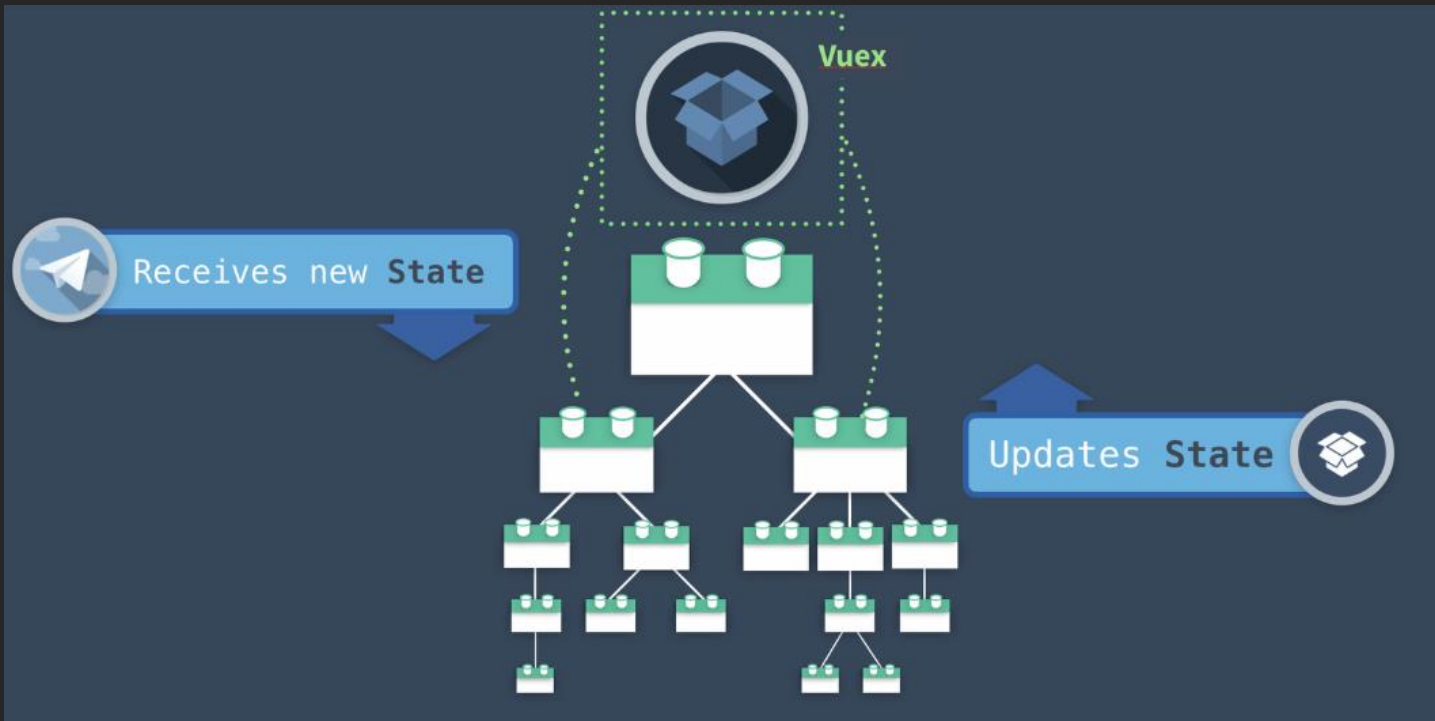
[Vuex 개념]

도대체 props를 몇번을 써야되는것이어.. 더 쉬운 방법 없을까여



[Vuex 개념]

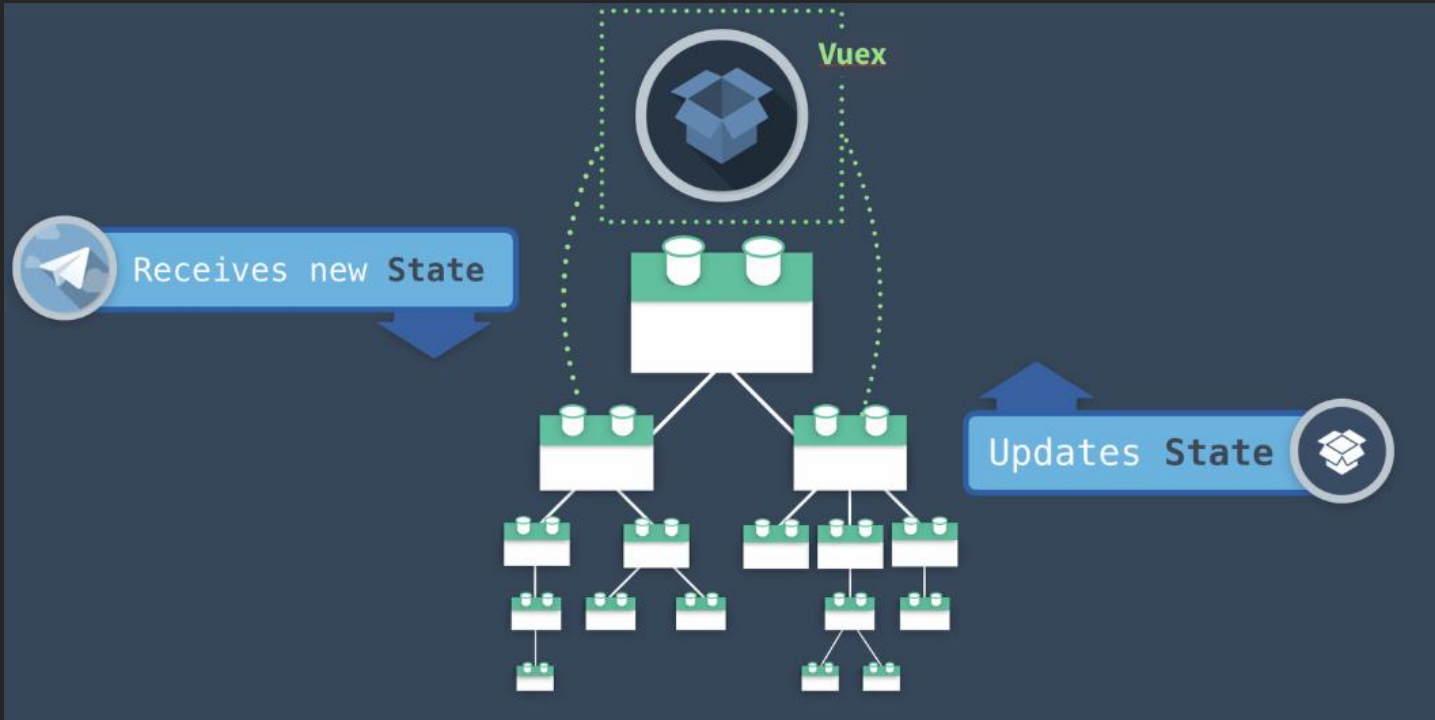
모든 Component가
데이터를 쉽게 꺼내고 쉽게 업데이트할 수 있게 도와주는 저장공간



Vuex는
데이터를 **State**라고 부릅니다.

[Vuex 장점]

1. 데이터를 App.vue에 저장하지 않고 하나의 js 파일에서 관리함
2. 데이터 넣고 뺄 땐 한줄의 코드만 작성하면 됩니다. (props/event 안해도 됨)



[Vuex 4개의 기능]

state

데이터 저장공간

getters

저장한 데이터 가져오기 기능

mutations

데이터 업데이트 기능

actions

mutations 실행하는 기능

[Vuex 설치]

```
yarn add vuex
```

[Vuex 셋팅 방법]

1. store 폴더 만들기
2. store 폴더 내에 store.js 같은 파일 하나 생성해서 작성

```
import Vue from 'vue'  
import Vuex from 'vuex'
```

```
Vue.use(Vuex);
```

```
export let store = new Vuex.Store({  
  state: {},  
  getters : {},  
  mutations: {},  
  actions : {}  
})
```


[Vuex 셋팅 방법]

3. main.js 상단에 아까 만든 파일 첨부

```
import { store } from './store/store.js'
```

4. 하단에는 store : 'store' 추가

```
new Vue({  
  el: '#app',  
  store,  
  router,  
  render: h => h(App)  
})
```

Vuex - State에 데이터 저장하는 법

익숙한 object 형식으로 저장하면 끝

```
state : {  
  name : "John Doe",  
  age : "28",  
}
```

store/index.js

1. 그냥 state 라는 곳에 저장하시면 됩니다. 끝!

(당연히 숫자, 문자 하드코딩으로 집어넣는게 아니라
서버 API 주소에서 가져오는 형태가 많겠죠?)

Getters로 데이터 꺼내는 법

getters에서 먼저 데이터 **꺼내는 방법**을 지정해줘야합니다.

```
getters : {  
  NAME (state) {  
    return state.name  
  }  
}
```

1. NAME이라는 함수 만들어놓기. (역할은 name 값 꺼내기)

store/index.js

```
let name = this.$store.getters.NAME
```

2. 원하는 vue파일에서 \$store.getter.NAME 쓰기

vuex 함수들은 보통 대문자로

Mutations로 데이터 수정하는 법 1

mutations로 데이터 수정방법을 지정해줘야합니다.

```
mutations : {  
  SET_NAME ( state, 이름 ) {  
    state.name = 이름  
  }  
}
```

1. SET_NAME 함수만들기 (역할은 name 데이터 수정)

store/index.js

그럼 이제 `.commit()` 을 이용해서 데이터를 업데이트할 수 있습니다

```
this.$store.commit( "SET_NAME" , 'John' )
```

2. vue파일에서 쓸때 `.commit(함수이름, 데이터)`

Mutations로 데이터 수정하는 법 2 (Ajax가 필요한 경우)

SET_NAME이라는 Mutations 함수 만든후 commit 함수 쓰시면 된다고 했는데

```
this.$store.commit( "SET_NAME" , 'John' )
```

Ajax요청이 필요하다면.. 그냥 쓰면 안되고
Actions 함수를 만든 후 그 안에서 .commit() 하셔야합니다.

이유1 : 원래 웹앱이 서버와 통신해서 데이터 업데이트할 땐 ajax 많이 씀

이유2 : Actions 함수는 비동기통신이 가능 (서버랑 데이터 주고받는 대기시간 동안 다른거 실행가능)

그래서 : Actions 안에 담아서 ajax 완료 후에 업데이트해라~ 이런 식으로 작성합니다.

이유3 : Mutations 함수는 비동기통신 못함

Actions vs Mutations

mutations

그냥 state(데이터) 수정하는 데만 관심있음

actions

Ajax요청과 mutations 동작시키는 데만 관심있음

```
this.$store.commit( "SET_NAME" , 'John' )
```

```
this.$store.dispatch( "SET_NAME" , 'John' )
```

Actions로 Mutations (데이터수정) 작동시키기

```
actions : {  
  SET_NAME (context, payload) {  
    context.commit( "SET_NAME" , payload);  
  }  
}
```

1. SET_NAME 함수 또만들기 (역할은 mutation 함수 동작)

2. ★ 함수의 기능 : mutation 함수 동작시키기 (commit)

store/index.js

dispatch를 이용해서 동작할 action을 고르고, 데이터를 넣으면 됩니다.

```
this.$store.dispatch( "SET_NAME" , 'John' )
```

3. vue에서 실행시킬 땐 dispatch() 문법 쓰기

[Vuex – Mutations 하면 되는건데 왜 Actions함?]

actions를 이용하면 '비동기' 형식으로 업데이트가 가능합니다.

```
actions : {  
  GET_NAME (context, payload) {  
    axios.get('/more.json')  
      .then( 가져온데이터 => {  
        context.commit( "SET_NAME" , 가져온데이터);  
      });  
  }  
}
```

1. SET_NAME 함수 만들기 (역할은 mutation 함수 동작)

2. 근데 Ajax요청 성공시에만 mutation하게 함

store/index.js

```
this.$store.dispatch( "GET_NAME" , 'John' )
```

3. vue에서 실행시킬 땐 dispatch() 문법 쓰기

Case1. 서버에서 데이터 가져와서 업데이트하기 (GET)

```
actions : {  
  GET_DATA (context, payload) {  
    axios.get('/more.json')  
      .then( 가져온데이터 => {  
        context.commit( "SET_NAME" , 가져온데이터);  
      });  
  }  
}
```

1. more.json경로에 있는 데이터 가져옴

2. 성공하면 우리 vuex데이터도 업데이트

store/index.js

```
this.$store.dispatch( "GET_DATA" , 'John' )
```

0. vue컴포넌트에서 GET_DATA함수를 쓰면

Case2. 서버로 데이터 보낼 때(POST)

```
actions : {  
  ADD_DATA (context, 보낼데이터) {  
    axios.post('/more.json', 보낼데이터)  
      .then( 응답 => {  
        context.commit( "SET_NAME" , 보낼데이터 );  
      });  
  }  
}
```

1. POST요청으로 'John' 이라는 데이터 보냄

2. 성공했으면.. vuex 데이터도 업데이트하자

store/index.js

```
this.$store.dispatch( "ADD_DATA" , 'John' )
```

0. ADD_DATA함수를 쓰면

Computed / mapstate 를 이용해 조금 쉽게 데이터를 꺼내자

```
computed: {  
  name() {  
    return this.$store.state.name  
  }  
},
```

App.vue

1. Store에 있는 데이터를 쉽게 꺼내고 싶다면
2. Computed안에 함수를 하나 만들어주세요

```
</p>{{name}}</p>
```

Computed / mapstate 를 이용해 조금 쉽게 데이터를 꺼내자

```
import { mapState } from 'vuex'
```

```
computed: {  
  ...mapState(['name', 'name2']),  
},
```

App.vue

1. mapState 라는 함수를 이용하면
2. 내가 원하는 데이터를 자동으로 computed 함수/데이터로 만들어줍니다.
3. 자매품 : mapGetters 등

PWA 만들기

장점

1. OS에 영향받지 않는 앱
2. 앱스토어 안거침 (심사기간, 수수료 등)
3. 저렴한 마케팅 비용

특징

1. URL입력창 없는 웹브라우저와 동일
2. Service worker로 빠른 브라우징 환경 제공

단점

1. 앱이 아니면 안되는 기능들 (앱아이콘 배지, 위젯 등)
2. 푸시알림, GPS 위치정보는 이용가능

PWA 만들기 :

(터미널에서 설치) `vue add @vue/pwa`



vue CLI의 도움으로 vue PWA 셋팅을 한번에 끝낼 수 있습니다.
(이렇게 설치하면 앱에 크롬 추노마크 안생김)

PWA 만들기 : manifest.json

public/manifest.json

우리 사이트의 정보를 알려줍니다.

```
{ "name": "설치유도용 앱이름",  
  "short_name": "폰에서 표시할 앱이름",  
  "start_url": "index.html",  
  "display": "standalone",  
  "theme_color": "#0476F2",  
  "background_color": "#fff",  
  "icons": [  
    {  
      "src": "앱아이콘이미지경로",  
      "sizes": "128x128",  
      "type": "image/png"  
    }  
  ]  
}
```

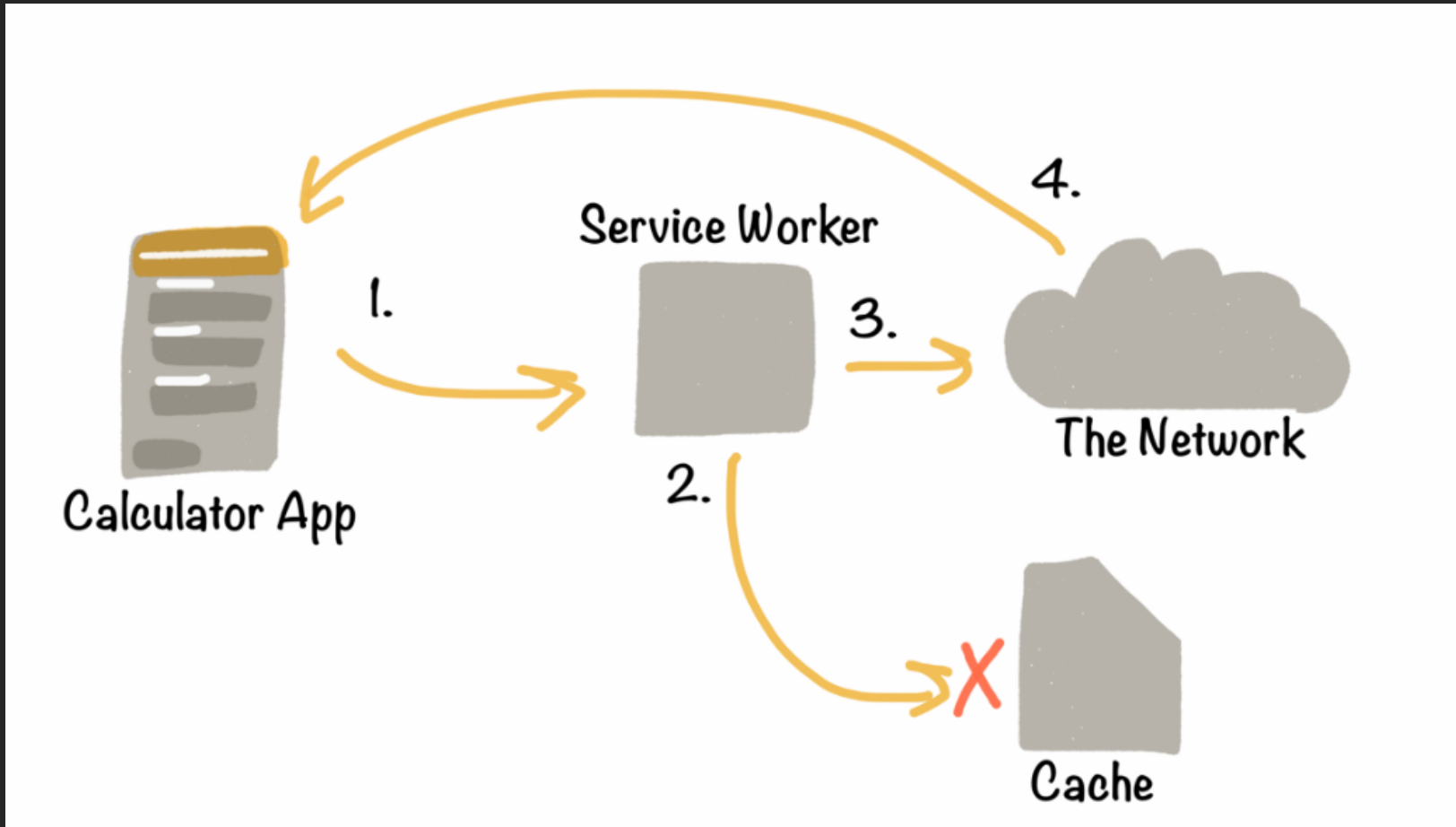
index.html 안의 <head>태그 안에 넣을 정보들

```
<link rel="manifest" href="/manifest.json"/>  
<meta name="apple-mobile-web-app-capable" content="yes">  
<meta name="apple-mobile-web-app-status-bar-style" content="default">  
<meta name="apple-mobile-web-app-title" content="DC-Covers">  
<meta name="msapplication-TileImage" content="이미지경로">  
<meta name="msapplication-TileColor" content="#000">
```



PWA 만들기 : ServiceWorker.js

service worker의 동작방식



PWA 만들기 : ServiceWorker.js

구글이 만든 Workbox라는 플러그인을 이용하기 때문에
실은 알아서 Cache 목록을 만들어 주는데,
더 커스터마이징 하고 싶으면
Vue 앱 기본 경로에
vue.config.js 파일 생성후 다음 코드 복사

```
module.exports = {  
  // ...other vue-cli plugin options...  
  pwa: {  
    name: 'My App',  
    themeColor: '#4DBA87',  
    msTileColor: '#000000',  
    appleMobileWebAppCapable: 'yes',  
    appleMobileWebAppStatusBarStyle: 'black',  
  
    // configure the workbox plugin  
    //workboxPluginMode: 'InjectManifest',  
    workboxOptions: {  
      // swSrc is required in InjectManifest mode.  
      swSrc: 'dev/sw.js',  
      // ...other Workbox options...  
    }  
  }  
}
```

Add to Home Screen 팝업 Prompt 띄우기

```
mounted(){  
  let deferredPrompt;  
  
  window.addEventListener('beforeinstallprompt', (e) => {  
    deferredPrompt = e;  
  
    //이제 원하는 순간에 밑의 코드로 팝업 띄우기  
  
    deferredPrompt.prompt();  
  });  
}
```

참고용 Vuestagram 개발단계

1. App.vue 만들기 (상단, 하단메뉴)
2. Body.vue 만들기 (내용 넣을 부분)
3. Post.vue 만들기 (게시물 보여줄 컴포넌트)
4. 포스팅 데이터를 Post.vue까지 데이터바인딩
5. Body.vue 내에 업로드 화면만들기 (if 를 이용한 페이지 구분)
6. 업로드1) 필터선택 화면 만들기
7. 업로드2) 글입력 화면 만들기
8. 업로드3) 발행누르면 App.vue까지 데이터가 전송되고 (이벤트) 메인페이지 이동
9. 필터 선택옵션 추가
 - 인스타필터 CSS 라이브러리 첨부하기
 - FilterBox.vue만들어서 for문으로 돌리기, 필터선택시 필터이름으로된 class명추가, 필터 선택시 class명이 App.vue까지 전송 (이벤트버스)