

# Vue

# INDEX

---

- Vue intro
- Why Vue
- Vue instance
- Basic of syntax
- Vue advanced

# Vue Intro

## | 사전 준비

- 공용 노션 “Vue.js 사전 준비 사항” 문서 참고
- VSCode Vetur extension 설치
  - 문법 하이라이팅, 자동완성, 디버깅 기능 제공
- Chrome Vue devtools extension 설치 및 설정
  - 크롬 브라우저 개발자 도구에서 vue 디버깅 기능 제공


## | 개요

- Front-end 개발이란 무엇인가
- Front-end framework란 무엇인가
- Vue를 배우는 이유
- Vue 기초 문법

# Front-end Development

# What is Front-end Development?

## | 개요

- 우리가 앞으로 할 일은 JavaScript를 활용한 Front-end 개발
- Back-end 개발은 Django로 하면 되는데,
- Front-end 개발은?
  - Vue.js 
  - Vue.js === JavaScript Front-end Framework

# What is Front-end Development?

## | Front-end Framework

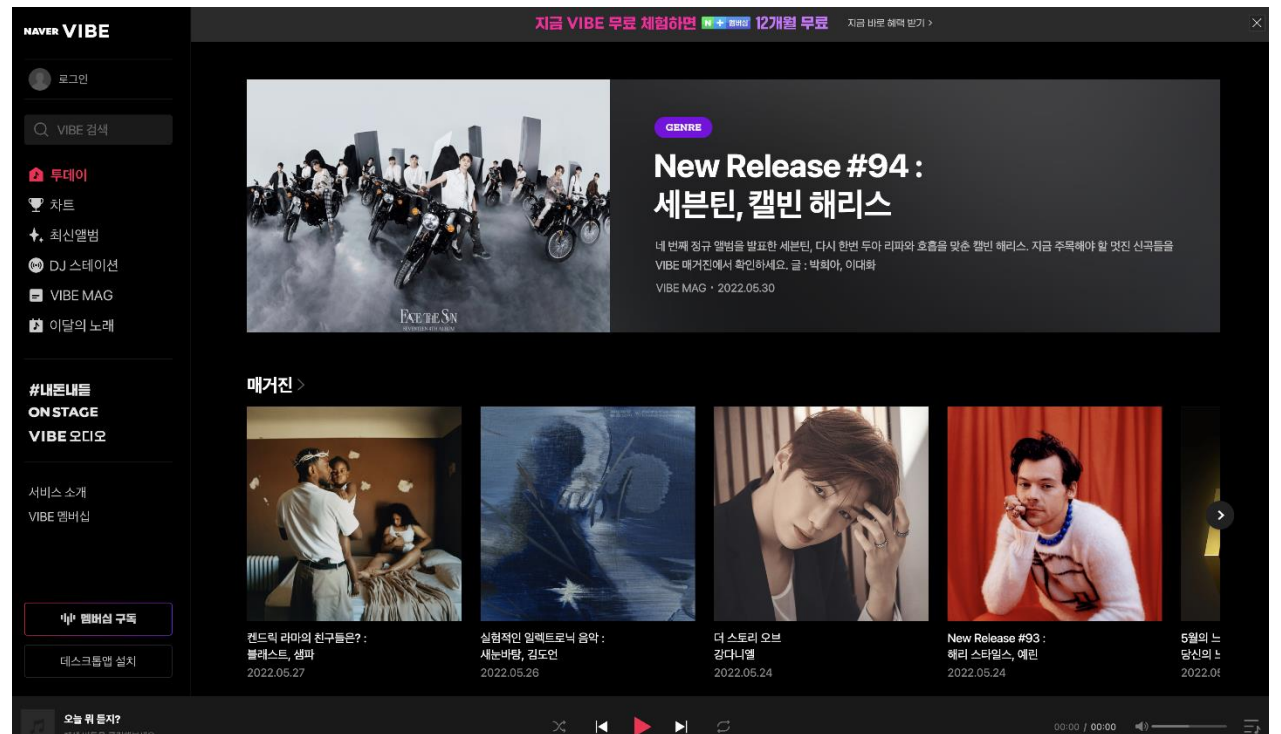
- Front-end(FE) 개발이란?
  - 사용자에게 보여주는 화면 만들기
- **Web App** (SPA)을 만들 때 사용하는 도구
  - SPA - Single Page Application



# What is Front-end Development?

## Web App이란? (1/2)

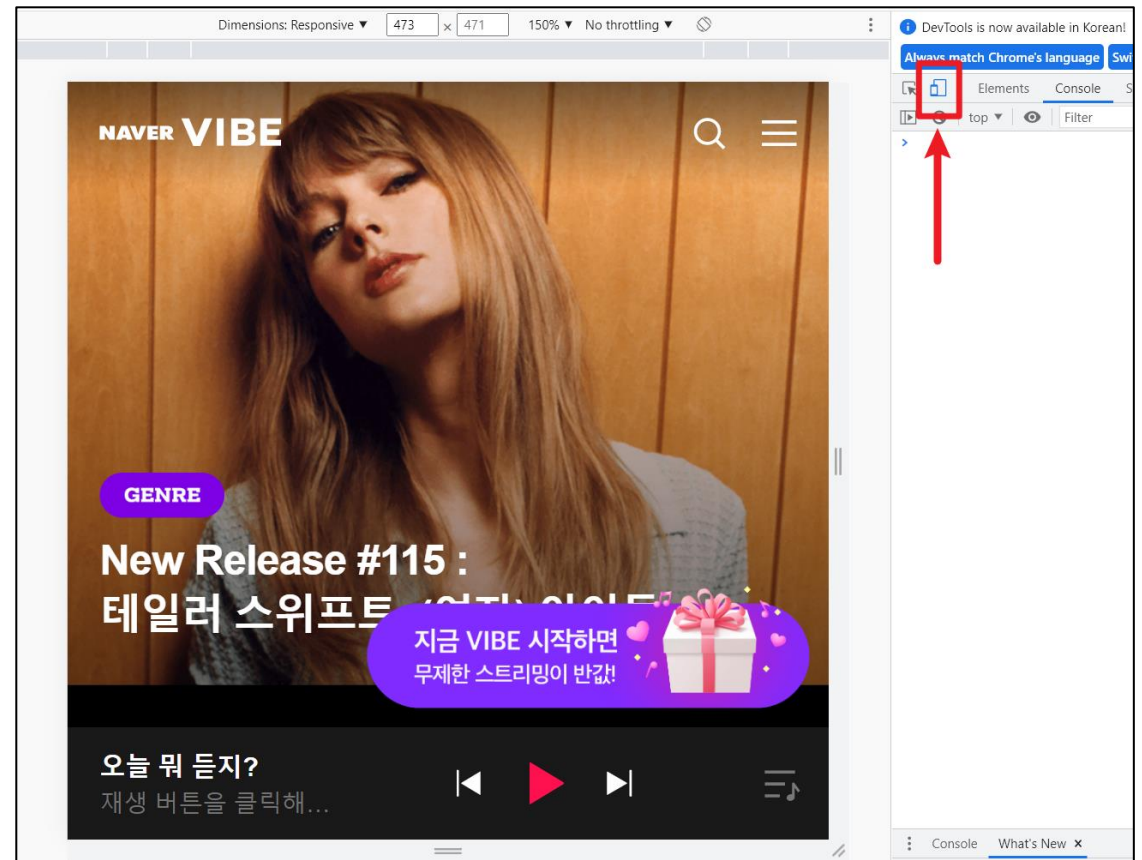
- 웹 브라우저에서 실행되는 어플리케이션 소프트웨어
- VIBE 웹 사이트로 이동
- <https://vibe.naver.com/today>



# What is Front-end Development?

## Web App이란? (2/2)

- 개발자 도구 > 디바이스 모드
- 웹 페이지가 그대로 보이는 것이 아닌  
디바이스에 설치된 App처럼 보이는 것
- 웹 페이지가 디바이스에 맞는  
적절한 UX/UI로 표현되는 형태



# What is Front-end Development?

## SPA (Single Page Application)

- Web App과 함께 자주 등장할 용어 SPA
- 이전까지는 사용자의 요청에 대해 적절한 페이지 별 template을 반환
- SPA는 서버에서 최초 1장의 HTML만 전달받아 모든 요청에 대응하는 방식
  - 어떻게 한 페이지로 모든 요청에 대응 할 수 있을까?
  - **CSR** (Client Side Rendering) 방식으로 요청을 처리하기 때문

# What is Front-end Development?

## [참고] SSR (Server Side Rendering) 이란? (1/2)

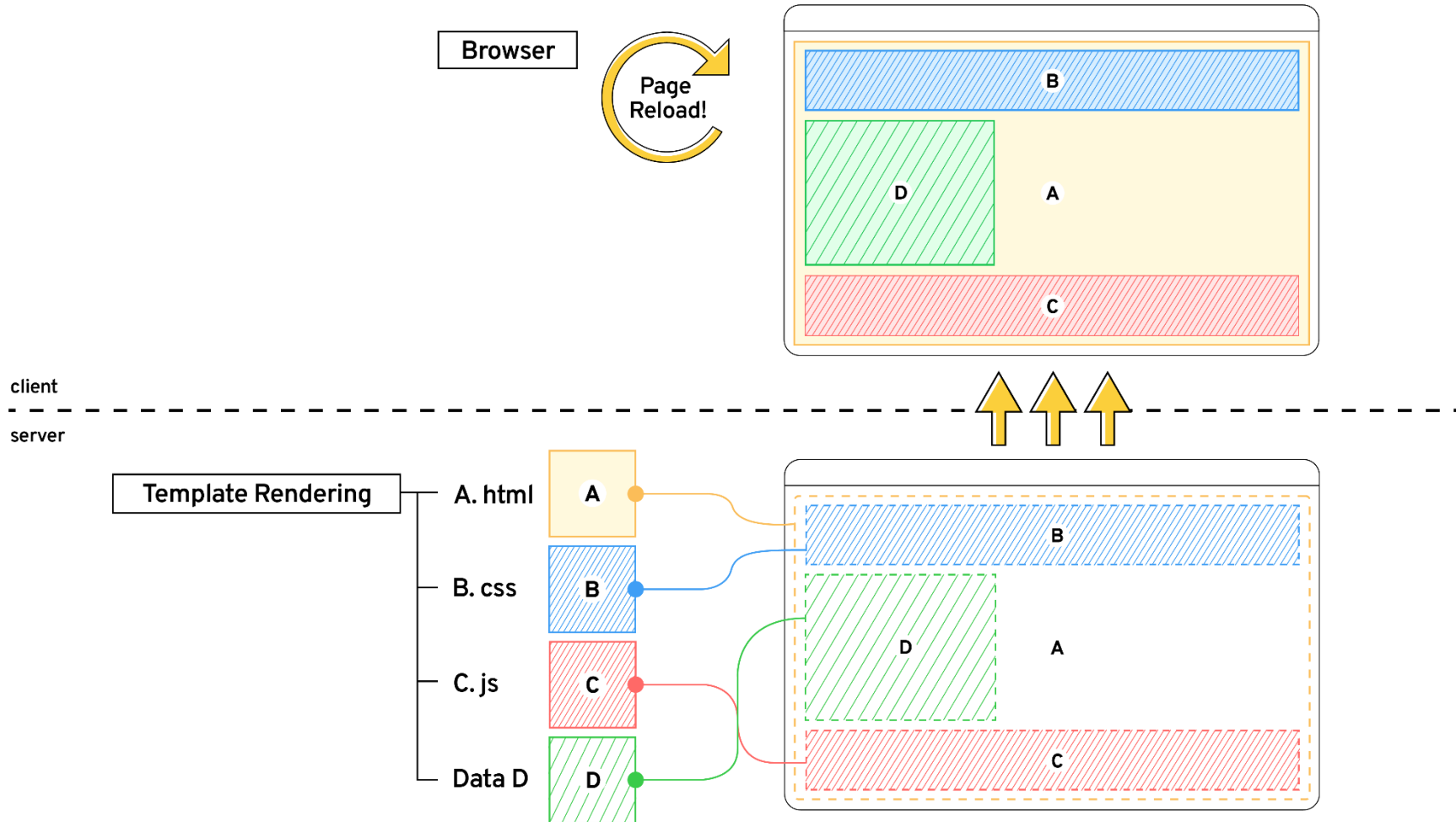
- 기존의 요청 처리 방식은 SSR
- Server가 사용자의 요청에 적합한 HTML을 렌더링하여 제공하는 방식
- 전달 받은 새 문서를 보여주기 위해 브라우저는 새로고침을 진행

```
{% extends 'base.html' %}

{% block content %}
    <h1>User List Page</h1>
    {% for user in users %}
        <p>{{ user.pk }}</p>
        <p>{{ user.username }}</p>
        <p>{{ user.password }}</p>
        <hr>
    {% empty %}
        <p>아직 가입한 유저가 없습니다.</p>
    {% endfor %}
{% endblock content %}
```

# What is Front-end Development?

## [참고] SSR (Server Side Rendering) 이란? (2/2)



# What is Front-end Development?

## CSR (Client Side Rendering) 이란? (1/3)

- 최초 한 장의 HTML을 받아오는 것은 동일
  - 단, server로부터 최초로 받아오는 문서는 빈 html 문서

```
<body>
  <noscript>
    <strong>We're sorry but ...</strong>
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
</body>
```

# What is Front-end Development?

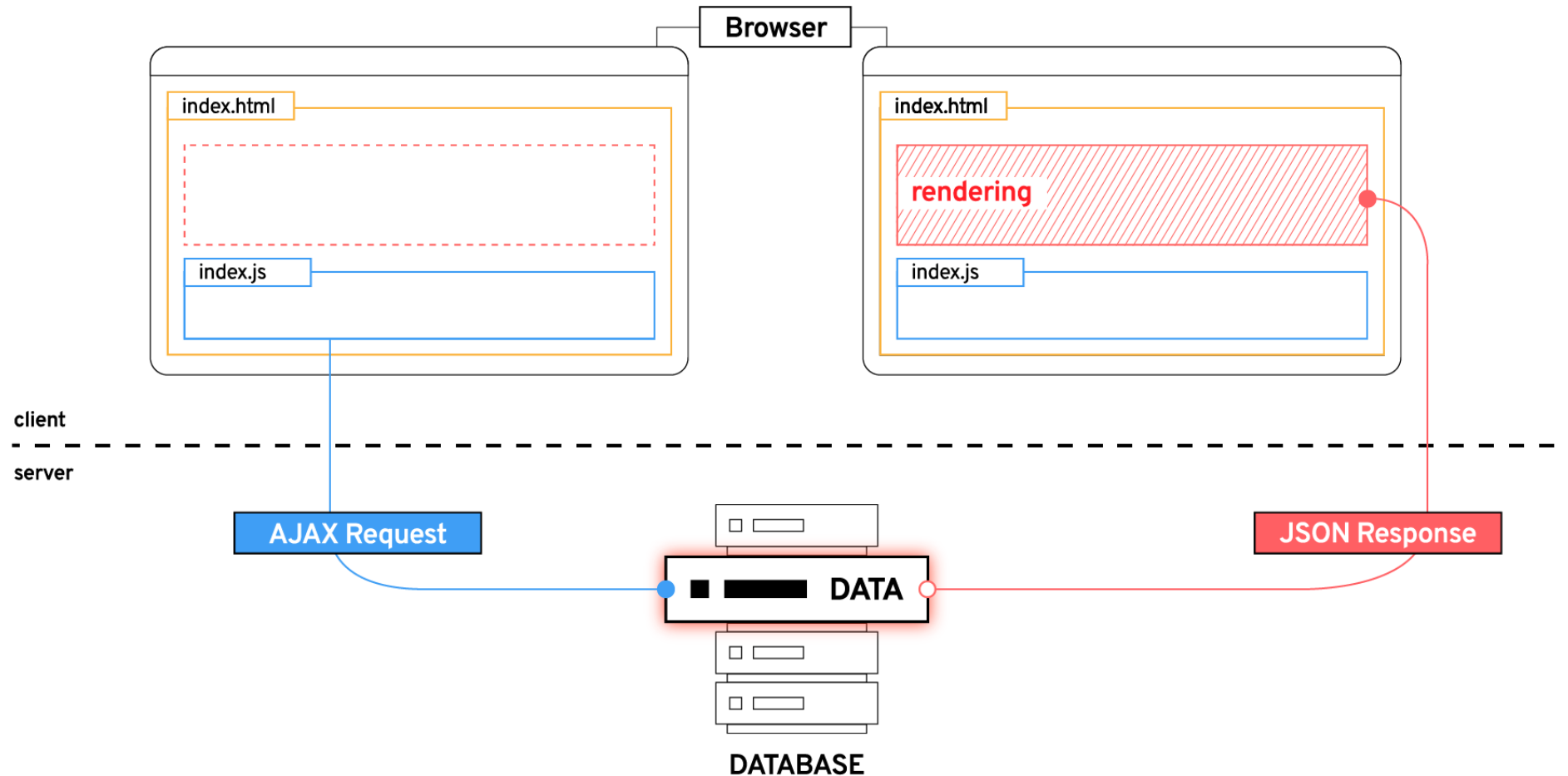
## CSR (Client Side Rendering) 이란? (2/3)

- 각 요청에 대한 대응을 JavaScript를 사용하여 필요한 부분만 다시 렌더링
1. 필요한 페이지를 서버에 **AJAX**로 요청
  2. 서버는 화면을 그리기 위해 필요한 데이터를 JSON 방식으로 전달
  3. **JSON** 데이터를 JavaScript로 처리, DOM 트리에 반영(렌더링)

```
axios.get(  
  HOST_URL,  
  {  
    headers:{  
      Authorization: `Token ${key}`  
    }  
  }  
)  
  .then(res => {  
    this.todos = res.data  
  })  
  .catch(err => console.log(err))
```

# What is Front-end Development?

## CSR (Client Side Rendering) 이란? (3/3)





# What is Front-end Development?

## 왜 CSR 방식을 사용하는 걸까?

1. 모든 HTML 페이지를 서버로부터 받아서 표시하지 않아도 됨  
== 클라이언트 - 서버간 통신 즉, 트래픽 감소  
== 트래픽이 감소한다 = 응답 속도가 빨라진다
2. 매번 새 문서를 받아 새로고침하는 것이 아니라 필요한 부분만 고쳐 나가므로 각 요청이 끊임없이 진행
  - SNS에서 추천을 누를 때 마다 첫 페이지로 돌아간다 = 끔직한 App!
  - 요청이 자연스럽게 진행이 된다 = UX 향상
3. BE와 FE의 작업 영역을 명확히 분리 할 수 있음
  - 각자 맡은 역할을 명확히 분리한다 = 협업이 용이해짐

# What is Front-end Development?

## CSR은 만능일까? (1/2)

- 첫 구동 시 필요한 데이터가 많으면 많을수록  
최초 작동 시작까지 오랜 시간이 소요
- Naver, Netflix, Disney+ 등 모바일에 설치된  
Web-App을 실행 하게 되면 잠깐의 로딩 시간이 필요



# What is Front-end Development?

## | CSR은 만능일까? (2/2)

- **검색 엔진 최적화**(SEO, Search Engine Optimization)가 어려움
  - 서버가 제공하는 것은 텅 빈 HTML
  - 내용을 채우는 것은 AJAX 요청으로 얻은 JSON 데이터로 클라이언트(브라우저)가 진행
- 대체적으로 HTML에 작성된 내용을 기반으로 하는 검색 엔진에  
빈 HTML을 공유하는 SPA 서비스가 노출되기는 어려움

# What is Front-end Development?

## [참고] SEO(Search Engine Optimization) (1/2)

- google, bing과 같은 검색 엔진 등에 내 서비스나 제품 등이 효율적으로 검색 엔진에 노출되도록 개선하는 과정을 일컫는 작업
- **검색** = 각 사이트가 운용하는 검색 엔진에 의해 이루어지는 작업
- **검색 엔진** = 웹 상에 존재하는 가능한 모든 정보들을 긁어 모으는 방식으로 동작
  - 정보의 대상은 주로 HTML에 작성된 내용
  - JavaScript가 실행된 이후의 결과를 확인하는 과정이 없음

# What is Front-end Development?

## [참고] SEO(Search Engine Optimization) (2/2)

- 최근에는 SPA, 즉 CSR로 구성된 서비스의 비중이 증가
  - SPA 서비스도 검색 대상으로 넓히기 위해 JS를 지원하는 방식으로 발전
- 단, 단순 HTML만을 분석하는 것보다 몇 배의 리소스가 필요한 작업이기에 여전히 CSR의 검색 엔진 최적화 문제가 모두 해결된 것은 아님

# What is Front-end Development?

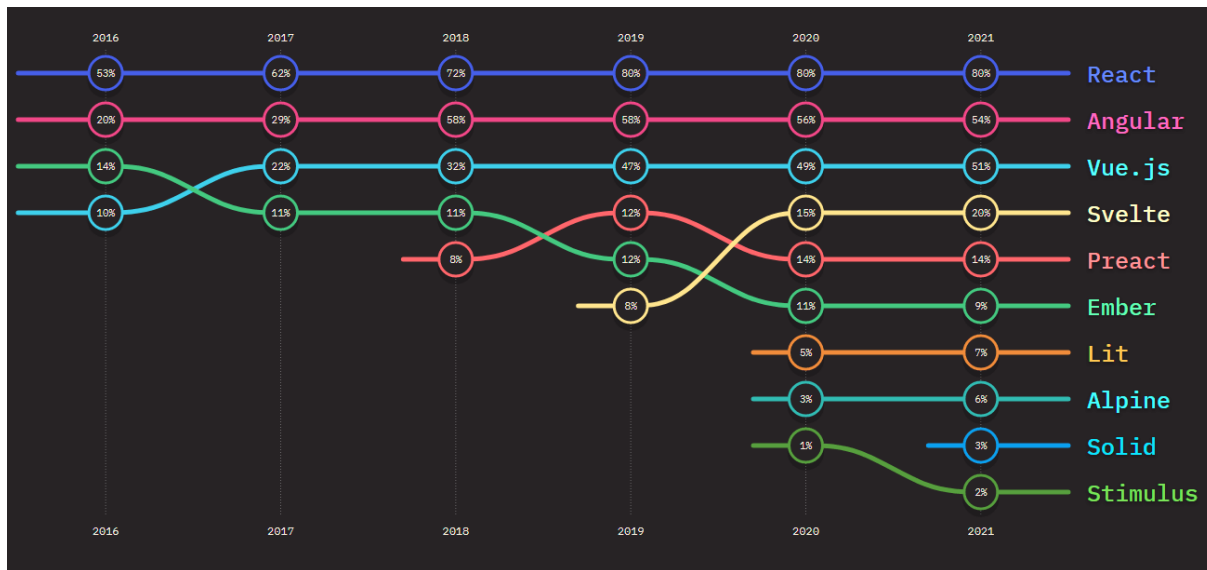
## | CSR vs SSR

- CSR과 SSR은 흑과 백이 아님
  - 내 서비스에 적합한 렌더링 방식을 적절하게 활용할 수 있어야 함
- SPA 서비스에서도 SSR을 지원하는 Framework이 발전하고 있음
  - Vue의 Nuxt.js
  - React의 Next.js
  - Angular Universal 등

# What is Front-end Development?

## 여러가지 Front-end Framework

- Front-end Framework == HTML + CSS + JS 를 더 편하게 작업하기 위한 툴
  - React, Angular, Svelte, **Vue** 등
- 2022년 Front-end Framework 인기도



# What is Front-end Development?

## | 그럼 이런 프레임워크를 꼭 써야 할까?

- No, 더 쉽게 개발하기 위해서 사용하는 것
- 실제로 Github은 이러한 Front-end Framework를 사용하지 않음
- 하지만 대부분의 기업에서는 생산성과 협업을 위해 Framework를 사용해서 개발



# Why Vue

## | 왜 우리는 vue를 배울까? (1/2)

- 쉽다.
- Vue는 타 Framework에 비해 입문자가 시작하기에 좋은 Framework
- 왜 Vue는 상대적으로 낮은 진입 장벽을 가질 수 있었을까?

## | 왜 우리는 vue를 배울까? (2/2)

- Vue를 발표한 개발자 Evan You
- 학사 - 미술, 미술사 / 석사 - 디자인 & 테크놀로지 전공
- 구글의 Angular 개발자 출신
  - Vue는 타 Framework에 비해 입문자가 시작하기에 좋은 Framework
  - Angular보다 **가볍고, 간편하게 사용할 수 있는** Framework를 만들기 위해 퇴사
  - 2014년 **Vue** 발표

# Why Vue

## | Vue 국내/외 실용 사례



# Why Vue

## Vue는 정말 쉬울까?

- Vue 구조는 매우 직관적임
- FE Framework를 빠르고 쉽게 학습하고 활용 가능
- 추후 필요하다면,  
다른 FE Framework 학습 시  
빠르게 적응 가능

```
// 01_vue_intro.vue

<template>
  <!-- HTML -->
  <div>
    <p>Hello :)</p>
  </div>
</template>

<script>
  // JavaScript
</script>

<style>
  /* CSS */
  p {
    color: black;
  }
</style>
```

## | Vue 없이 코드 작성하기 (1/3)

- 입력 받은 값을 name 뒤에 출력하기
- 02\_html\_only.html 에서 진행

```
// 02_html_only.html

<!DOCTYPE html>
<html lang="en">
  ...
  <body>
    <div id="app">
      <p id="name">name : </p>
      <input id="inputName" type="text">
    </div>

    <script>
      // CODE HERE
    </script>
  </body>
</html>
```

## | Vue 없이 코드 작성하기 (2/3)

1. input tag 선택
2. P tag 선택
3. addEventListener 추가

```
// 02_html_only.html

<body>
  <div id="app">
    <p id="name">name : </p>
    <input id="inputName" type="text">
  </div>

  <script>
    // CODE HERE
    const name = document.querySelector('#name ')
    const input = document.querySelector('#inputName ')
    input.addEventListener('input', function (e) {
      ...
    })
  </script>
</body>
```

## | Vue 없이 코드 작성하기 (3/3)

- 입력 받은 데이터를 p tag에 추가하려고 한다면?
- 기존에 가지고 있었던 text도 신경 써야함
  - data를 관리하기 위한 추가 작업이 필요함

```
// 02_html_only.html

input.addEventListener('input', function (e) {
  name.innerText = name.innerText + e.target.value
})
```



## | Vue CDN (1/2)

- Vue로 작업을 시작하기 위하여 CDN을 가져와야 함
- Django == Python Web Framework
  - pip install
- Vue === JS Front-end Framework
  - Bootstrap에서 사용하였던 CDN 방식 제공
  - npm 활용은 추후에 진행 예정

## Vue CDN (2/2)

- Vue CDN을 위하여 **Vue2 공식 문서** 접속

- <https://v2.vuejs.org/>

1. Getting Started

2. Installation

3. Development version CDN 복사

### Installation



The official guide assumes intermediate level knowledge of HTML, CSS, and JavaScript. If you are totally new to frontend development, it might not be the best idea to jump right into a framework as your first step - grasp the basics then come back! Prior experience with other frameworks helps, but is not required.

The easiest way to try out Vue.js is using the **Hello World example**. Feel free to open it in another tab and follow along as we go through some basic examples. Or, you can **create an index.html file** and include Vue with:

```
<!-- development version, includes helpful console warnings -->  
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
```

HTML

## Vue로 코드 작성하기 (1/3)

- 입력 받은 값을 name 뒤에 출력하기
- 03\_html\_vue.html 에서 진행

```
// 03_html_vue.html

<!DOCTYPE html>
<html lang="en">
  ...
  <body>
    <div id="app">
      <p id="name">name : </p>
      <input id="inputName" type="text">
    </div>

    <script>
      // CODE HERE
    </script>
  </body>
</html>
```

## Vue로 코드 작성하기 (2/3)

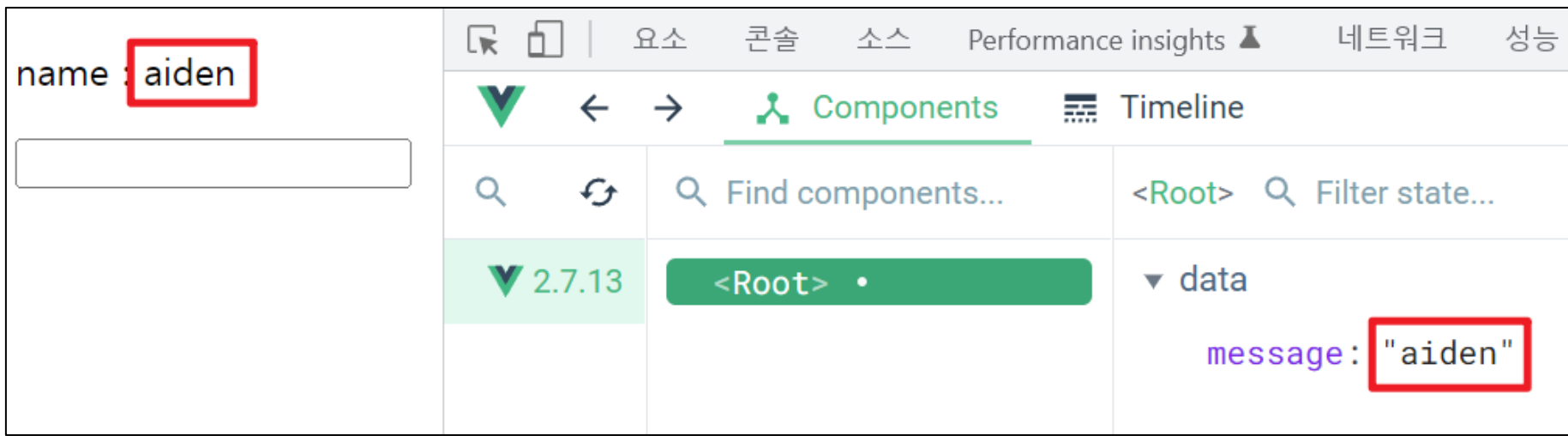
1. Vue CDN 가져오기
2. Vue instance 생성
  - Vue instance - 1개의 Object
  - 미리 정해진 속성명을 가진 Object
3. **el, data** 설정
  - data에 관리할 속성 정의
4. 선언적 렌더링 **{{ }}**
  - Vue data를 화면에 렌더링

```
// 03_html_vue.html

<body>
  <div id="app">
    <p id="name">name : {{ message }} </p>
    <input type="text">
  </div>
  <!-- Vue CDN 생략 -->
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        message: '',
      },
    })
  </script>
</body>
```

## [참고] Dev Tools 확인

- Vue devtools에서 data 변경 -> DOM 반영
- 눈에 보이는 화면을 조작하는 것이 아닌 Vue가 가진 data를 조작



## | Vue로 코드 작성하기 (3/3)

### 4. input tag에 **v-model** 작성

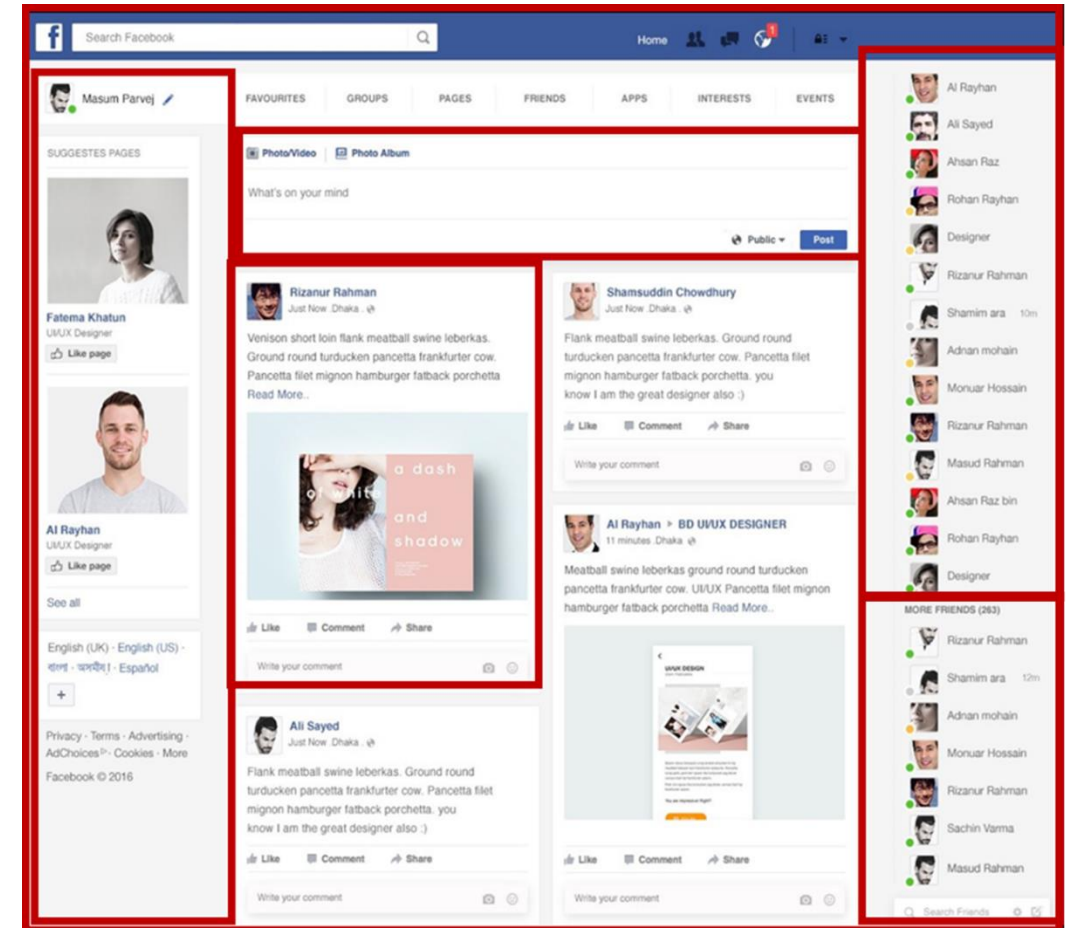
- input에 값 입력 -> Vue data 반영
- Vue data -> DOM 반영

```
// 03_html_vue.html

<body>
  <div id="app">
    <p id="name">name : {{ message }} </p>
    <input type="text" v-model="message">
  </div>
  <!-- Vue CDN -->
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        message: '',
      },
    })
  </script>
</body>
```

## Facebook 예시 (1/3)

- 한 명의 유저가 이름을 변경한다면  
화면에서 조작해야 할 영역이 매우 많음



## Facebook 예시 (2/3)

- Vanilla JS만으로 모든 데이터를 조작한다면?

불필요한 코드의 반복

```
<label for="inputArea">Username: </label>
<input type="text" id="inputArea">
<hr>

<h1>안녕하세요 <span id="username1"></span></h1>

<div>
  <span id="username2"></span>님의 친구목록
</div>

<div>
  <span id="username3"></span>님의 알림목록
</div>

<div>
  <span id="username4"></span>님의 친구 요청 목록
</div>
```

```
const inputArea = document.querySelector('#inputArea')
const initialText = 'Unknown'

const username1 = document.querySelector('#username1')
const username2 = document.querySelector('#username2')
const username3 = document.querySelector('#username3')
const username4 = document.querySelector('#username4')

username1.innerText = initialText
username2.innerText = initialText
username3.innerText = initialText
username4.innerText = initialText

inputArea.addEventListener('change', function (event) {
  const newUsername = event.target.value

  username1.innerText = newUsername
  username2.innerText = newUsername
  username3.innerText = newUsername
  username4.innerText = newUsername
})
```



## Facebook 예시 (3/3)

- Vue를 통해 데이터를 관리한다면? = 변경 사항도 한 번에 반영

하나의 Data로 관리

```
<div id="app">
  <label for="inputArea">Username: </label>
  <input v-on:keyup.enter="onInputChange" type="text" id="inputArea">
  </div>
  <h1>안녕하세요 {{ username }}</h1>

  <div>
    {{ username }} 님의 친구 목록
  </div>

  <div>
    {{ username }} 님의 알림 목록
  </div>

  <div>
    {{ username }} 님의 친구 요청 목록
  </div>
</div>
```

2. DOM re-render

```
const app = new Vue({
  el: '#app',
  data: {
    username: 'Unknown'
  },
  methods: {
    onInputChange: function (event) {
      this.username = event.target.value
    }
  }
})
```

1. Data changes

# Vue 2 vs Vue 3

## | Vue3

- 2022년 02월 부터 Vue 프레임워크의 기본 버전이 3버전으로 전환
- 대체적인 설정들이 Vue3을 기본으로 적용되어 있음
  - ex) 공식문서, CDN, npm 등

## | Vue2

- 여전히 Vue2가 많이 사용됨 (legacy code)
- 사용된 기간이 긴 만큼 상대적으로 많은 문서의 양, 참고자료, 질문/답변
- 안정적인 측면에서는 아직 Vue2가 우세한 편

# 이어서 ..

삼성 청년 SW 아카데미

# Vue instance

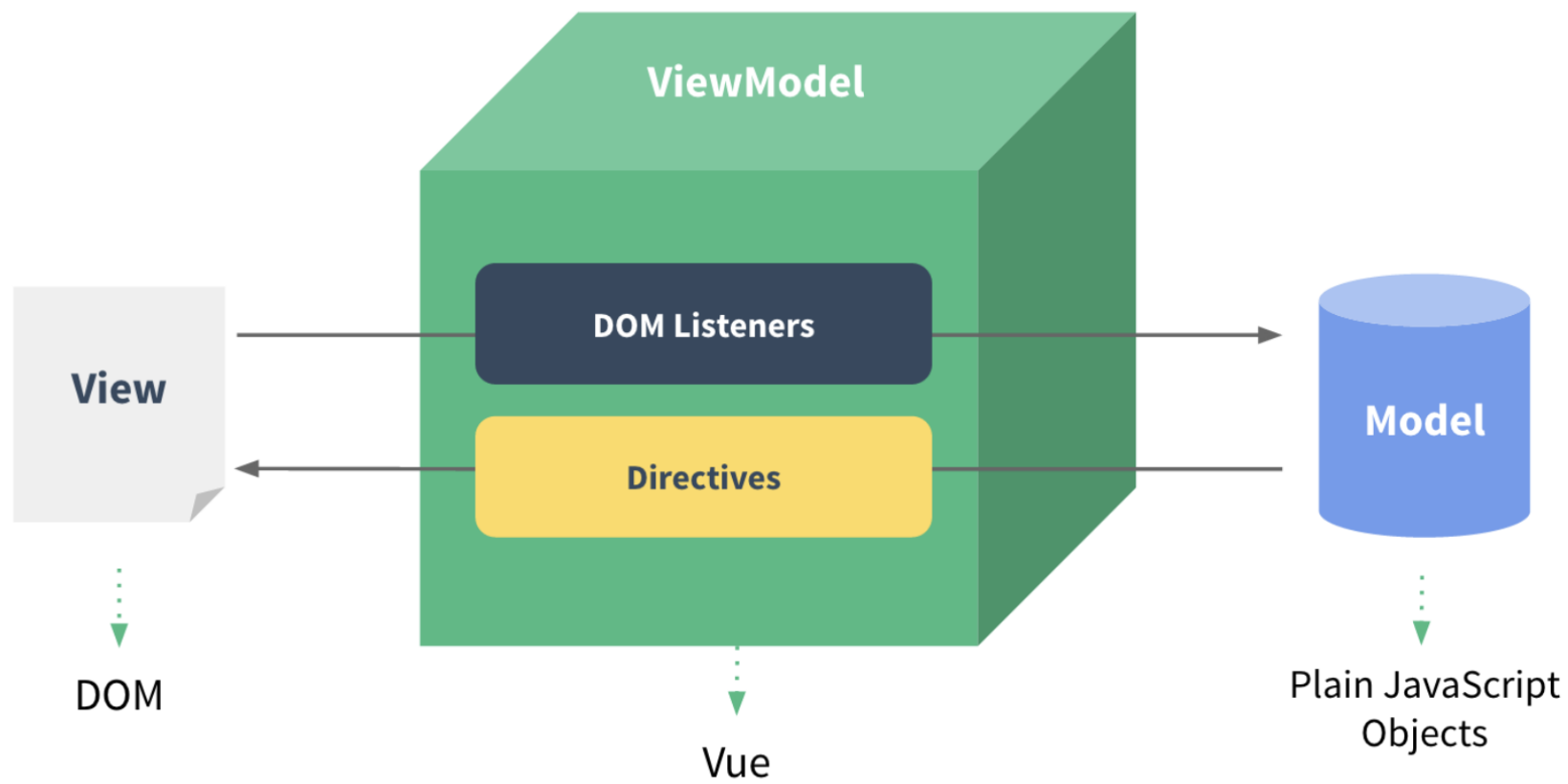
## MVVM Pattern (1/4)

- 소프트웨어 아키텍처 패턴의 일종
- 마크업 언어로 구현하는 그래픽 사용자 인터페이스( view )의 개발을

Back-end( model )로부터 분리시켜

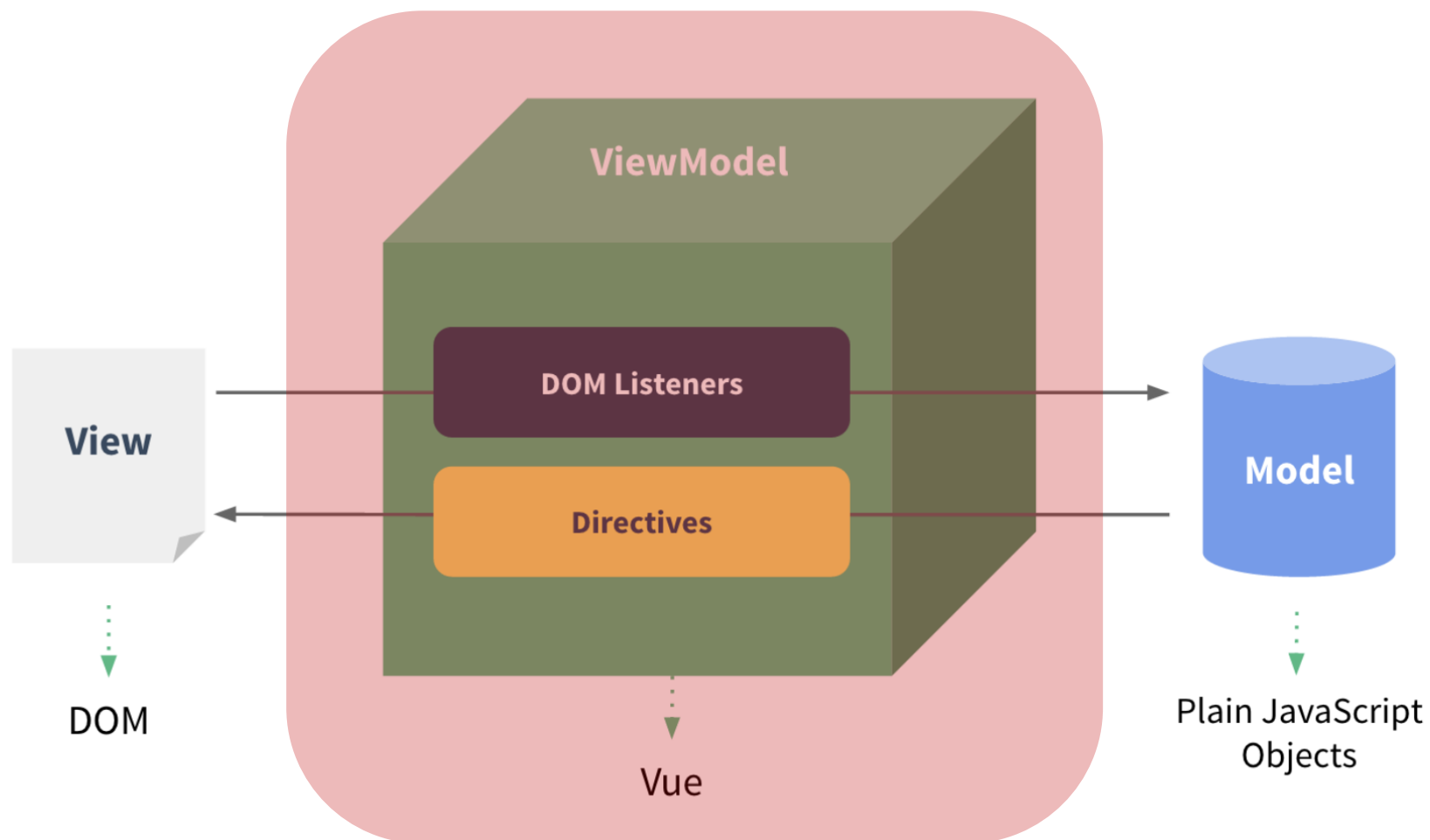
view가 어느 특정한 모델 플랫폼에 종속되지 않도록 함

## MVVM Pattern (2/4)





## MVVM Pattern (3/4)



## MVVM Pattern (4/4)

- **View** - 우리 눈에 보이는 부분 = DOM !
- **Model** - 실제 데이터 = JSON !
- **View Model** (Vue)
  - View를 위한 Model
  - View와 연결(binding)되어 Action을 주고 받음
  - Model이 변경되면 View Model도 변경되고 바인딩된 View도 변경됨
  - View에서 사용자가 데이터를 변경하면  
View Model의 데이터가 변경되고 바인딩된 다른 View도 변경됨

## MVVM Pattern 정리

- MVC 패턴에서 Controller를 제외하고 View Model을 넣은 패턴
- View는 Model을 모르고, Model도 View를 모른다  
== DOM은 Data를 모른다, Data도 DOM을 모른다 (독립성 증가, 적은 의존성)
- View에서 데이터를 변경하면 View Model의 데이터가 변경되고,  
연관된 다른 View도 함께 변경된다

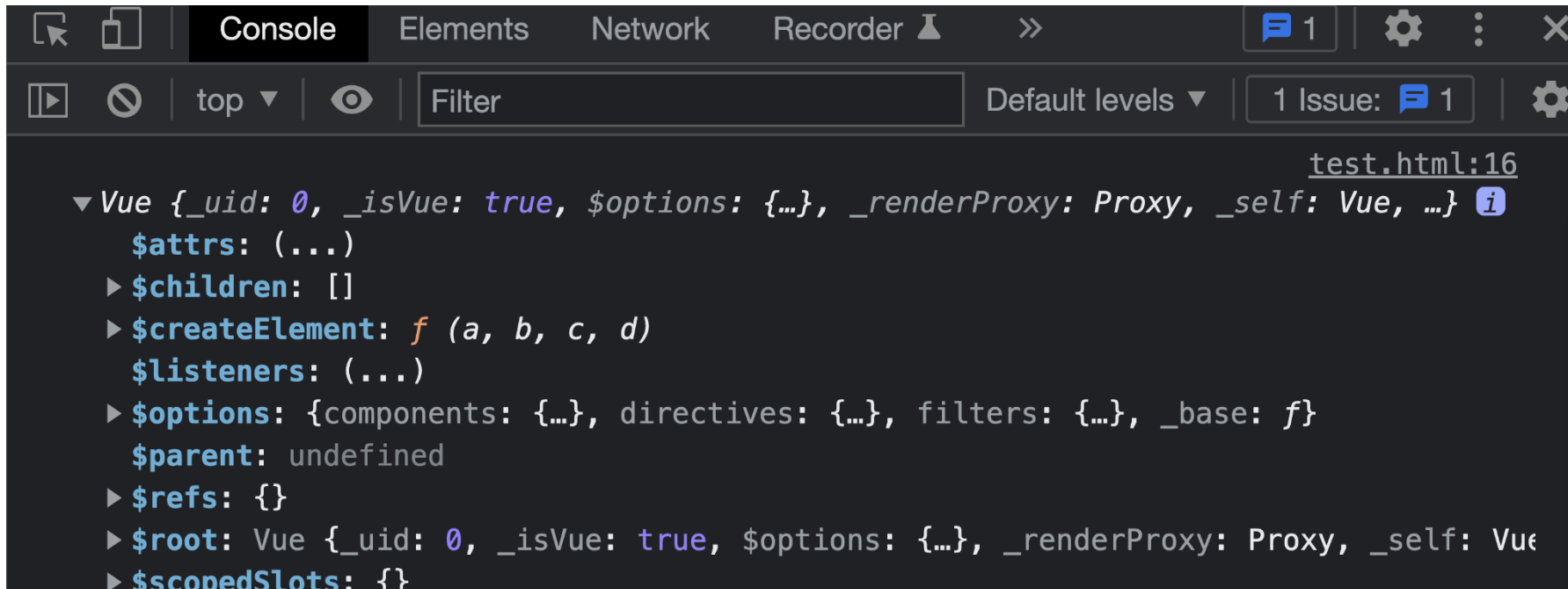
## Vue instance (1/2)

- 04\_vue\_start.html에서 작업 진행
  1. Vue CDN 가져오기
  2. new 연산자를 사용한 생성자 함수 호출
    - vue instance 생성
  3. 인스턴스 출력 및 확인

```
// 04_vue_start.html

<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>
<body>
  <!-- vue CDN 작성 -->
  <script>
    // CODE HERE
    const vm = new Vue()
    console.log(vm)
  </script>
</body>
</html>
```

## Vue instance (2/2)



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays a log message from 'test.html:16' showing a Vue instance object. The object is a Vue instance with various properties and methods. The console interface includes a toolbar with icons for back, forward, and search, as well as a filter input field and a 'Default levels' dropdown. The log message is expanded, showing the following properties and methods:

```
Vue {_uid: 0, _isVue: true, $options: {...}, _renderProxy: Proxy, _self: Vue, ...}
  $attrs: (...)
  $children: []
  $createElement: f (a, b, c, d)
  $listeners: (...)
  $options: {components: {...}, directives: {...}, filters: {...}, _base: f}
  $parent: undefined
  $refs: {}
  $root: Vue {_uid: 0, _isVue: true, $options: {...}, _renderProxy: Proxy, _self: Vue}
  $scopedSlots: {}
```

- Vue instance === 1개의 객체
- 아주 많은 속성과 메서드를 이미 가지고 있고, 이러한 기능들을 사용하는 것

## [참고] 생성자 함수 (1/3)

- 05\_constructor\_func.js 에서 진행
- JS에서 객체를 하나 생성한다고 한다면?
  - 하나의 객체를 선언하여 생성
- 동일한 형태의 객체를 또 만든다면?
  - 또 다른 객체를 선언하여 생성

```
const member = {  
  name: 'aiden',  
  age: 22,  
  sId: 2022311491,  
}
```

```
const member2 = {  
  name: 'haley',  
  age: 20,  
  sId: 2022311492,  
}
```

## [참고] 생성자 함수 (2/3)

- 동일한 구조의 객체를 여러 개 만들고 싶다면?
- 생성자 함수는 특별한 함수를 의미하는 것이 아님
- **new** 연산자로 사용하는 함수

```
function Member(name, age, sId) {  
  this.name = name  
  this.age = age  
  this.sId = sId  
}  
  
const member3 = new Member('isaac', 21, 2022654321)
```

## [참고] 생성자 함수 (3/3)

- 함수 이름은 반드시 대문자로 시작
- 생성자 함수를 사용할 때는 반드시 **new** 연산자를 사용



## el (element) (1/3)

- Vue instance와 DOM을 mount(연결)하는 옵션
  - View와 Model을 연결하는 역할
  - HTML id 혹은 class와 마운트 가능
- Vue instance와 연결되지 않은 DOM 외부는 Vue의 영향을 받지 않음
  - Vue 속성 및 메서드 사용 불가

## el (element) (2/3)

- 04\_vue\_start.html 에서 작업 진행
- 새로운 Vue instance 생성
- 생성자 함수 첫번째 인자로 Object 작성
- el 옵션에 #app 작성 = DOM 연결
- 인스턴스 출력

```
<div id="app">
</div>
...
<script>
  // const vm = new Vue()
  // console.log(vm)

  const app = new Vue({
    el: '#app'
  })
  console.log(app)
</script>
```

## el (element) (2/3)

- 04\_vue\_start.html 에서 작업 진행
- 새로운 Vue instance 생성
- 생성자 함수 첫번째 인자로 **Object** 작성
- el 옵션에 **#app** 작성 = DOM 연결
- 인스턴스 출력

```
<div id="app">
</div>

...
<script>
  // const vm = new Vue()
  // console.log(vm)

  const app = new Vue({
    el: '#app'
```

```
▼ Vue {_uid: 0, _isVue: true, __v_skip: true, _scope: EffectScope, $options: {...}, ...} ⓘ
  $attrs: (...)
  ▶ $children: []
  ▶ $createElement: f (a, b, c, d)
  ▶ $el: div#app
  $listeners: (...)
  ▶ $options: {components: {...}, directives: {...}, filters: {...}, el: '#app', _base: f, ...}
  $parent: undefined
  ▶ $refs: {}
  ▶ $root: Vue {_uid: 0, _isVue: true, __v_skip: true, _scope: EffectScope, $options: {...}, ...}
  ▶ $scopedSlots: {}
  ▶ $slots: {}
```

## el (element) (3/3)

- Vue와 연결되지 않은 div 생성
  - 두 div 모두에 `{{ message }}` 작성
  - 결과 확인
- `message` 속성이 정의 되지 않았다는 경고와
- `{{ message }}` 가 그대로 출력되는 차이

```
<div id="app">
  {{ message }}
</div>
<div>
  {{ message }}
</div>
<script>
  const app = new Vue({
    el: '#app'
  })
  console.log(app)
</script>
```

## el (element) (3/3)

- Vue와 연결되지 않은 div 생성
  - 두 div 모두에 `{{ message }}` 작성
  - 결과 확인
- `message` 속성이 정의 되지 않았다는 경고와
- `{{ message }}` 가 그대로 출력되는 차이

```
<div id="app">
  {{ message }}
</div>
<div>
  {{ message }}
</div>
<script>
  const app = new Vue({
    el: '#app'
  })
  console.log(app)
```

✖ ▶ [Vue warn]: Property or method "message" is not defined on the [vue.js:5106](#) instance but referenced during render. Make sure that this property is reactive, either in the data option, or for class-based components, by initializing the property. See: <https://v2.vuejs.org/v2/guide/reactivity.html#Declaring-Reactive-Properties>.

(found in <Root>)

## | data (1/2)

- Vue instance의 데이터 객체 혹은 인스턴스 속성
- 데이터 객체는 반드시 기본 객체 `{ }` (Object) 여야 함
- 객체 내부의 아이템들은 value로 모든 타입의 객체를 가질 수 있음
- 정의된 속성은 interpolation `{{ }}` 을 통해 view에 렌더링 가능함

## data (2/2)

- Vue instance에 **data** 객체 추가
- data 객체에 **message** 값 추가
- 결과 확인
- 추가된 객체의 각 값들은 **this.message** 형태로 접근 가능

```
<div id="app">
  {{ message }}
</div>

<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello, Vue!'
    },
  })
</script>
```

## methods (1/2)

- Vue instance의 **method**들을 정의하는 곳
- **methods** 객체 정의
  - 객체 내 print method 정의
  - print method 실행 시  
Vue instance의 data내 message 출력
- 콘솔창에서 app.print() 실행

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello, Vue!'
    },
    methods: {
      print: function () {
        console.log(this.message)
      },
    }
  })
</script>
```



## methods (1/2)

- Vue instance의 **method**들을 정의하는 곳
- **methods** 객체 정의
  - 객체 내 print method 정의
  - print method 실행 시  
Vue instance의 data내 message 출력
- 콘솔창에서 app.print() 실행

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello, Vue!'
    },
    methods: {
      print: function () {
        console.log(this.message)
      },
    }
  })
```

```
> app.print()
Hello, Vue!                                04_vue_start.html:36
< undefined
>
```

## methods (2/2)

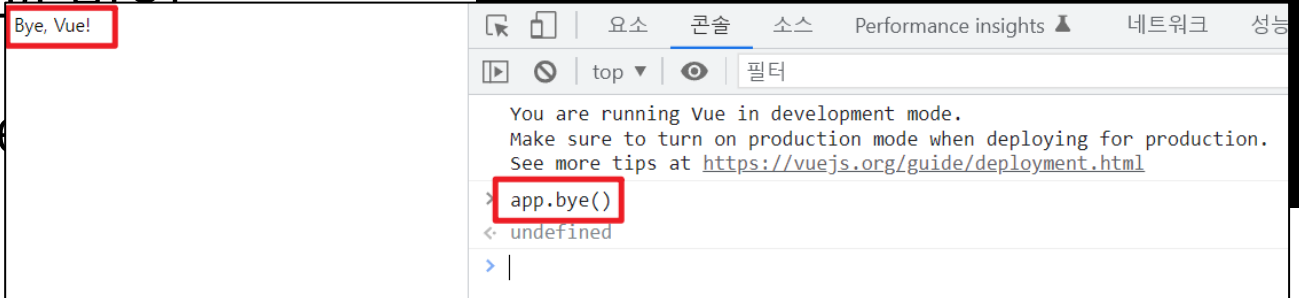
- method를 호출하여 data 변경 가능
  - 객체 내 bye method 정의
  - print method 실행 시  
Vue instance의 data내 message 변경
- 콘솔창에서 app.bye() 실행
  - DOM에 바로 변경된 결과 반영
  - Vue의 강력한 반응성(reactivity)

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello, Vue!'
    },
    methods: {
      ...
      bye: function () {
        this.message = 'Bye, Vue!'
      },
    }
  })
</script>
```

## methods (2/2)

- method를 호출하여 data 변경 가능
  - 객체 내 bye method 정의
  - print method 실행 시  
Vue instance의 data내 message 변경
- 콘솔창에서 app.bye() 실행
  - DOM에 바로 변경된 결과 보여
- Vue의 강력한 반응성(reactivity)

```
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello, Vue!'
    },
    methods: {
      ...
      bye: function () {
        this.message = 'Bye, Vue!'
      },
    }
  })
```



## [주의] methods with Arrow Function

- 메서드를 정의 할 때,  
Arrow Function을 사용하면 안됨
- Arrow Function의 this는 함수가 선언될 때  
상위 스코프를 가리킴
- 즉 this가 상위 객체 window를 가리킴
- 호출은 문제 없이 가능하나  
this로 Vue의 data를 변경하지 못함

```
<script>
  const app = new Vue({
    el: '#app',
    ...
    methods: {
      ...
      arrowBye: () => {
        this.message = 'Arrow?'
        console.log(this)
      }
    }
  })
</script>
```

## [주의] methods with Arrow Function

- 메서드를 정의 할 때,  
Arrow Function을 사용하면 안됨
- Arrow Function의 this는 함수가 선언될 때  
상위 스코프를 가리킴
- 즉 this가 상위 객체 window를 가리킴
- 호출은 문제 없이 가능하나  
this로 Vue의 data를 변경하지 못함

```
<script>  
  const app = new Vue({  
    el: '#app',  
    ...  
    methods: {  
      ...  
    }  
  })
```

```
> app.arrowBye()  
< undefined  
> app.print()  
Hello, Vue! 04_vue_start.html:36  
< undefined  
> |
```

```
> app.arrowBye()  
Window {window: Window, self: Window, document: document, name: '', location:  
  Location, ...} 04_vue_start.html:46  
< undefined  
> |
```

# 이어서 ..

삼성 청년 SW 아카데미

# Basic of Syntax

## Template Syntax

- Vue2 guide > Template Syntax 참고
- 렌더링 된 DOM을 기본 Vue instance의 data에 선언적으로 바인딩  
할 수 있는 HTML 기반 template syntax를 사용
  - 렌더링 된 DOM : 브라우저에 의해 보기 좋게 그려질 HTML 코드
  - HTML 기반 template syntax : HTML 코드에 직접 작성할 수 있는 문법 제공
  - 선언적으로 바인딩 : Vue instance와 DOM을 연결



## Text Interpolation

- 06\_basic\_of\_syntax.html에서 진행
- 가장 기본적인 바인딩(연결) 방법
- 중괄호 2개로 표기
- DTL과 동일한 형태로 작성
- Text interpolation 방법은 모두 **일반 텍스트**로 표현

```
<div id="app">
  <p>메시지: {{ msg }}</p>
  <p>HTML 메시지 : {{ rawHTML }}</p>
</div>

<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      msg: 'Text interpolation',
      rawHTML: '<span
style="color:red"> 빨간 글씨</span>'
    }
  })
</script>
```

## RAW HTML

- **v-html** directive을 사용하여 data와 바인딩
- directive - HTML 기반 template syntax
- HTML의 기본 속성이 아닌 Vue가 제공하는 특수 속성의 값으로 data를 작성

```
<div id="app">
  <p>HTML 메시지 :
    <span v-html="rawHTML"></span>
  </p>
</div>

<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      rawHTML: '<span
style="color:red"> 빨간 글씨</span>'
    }
  })
</script>
```

## [참고] JS 표현식

- 표현식 형태로 작성 가능

```
<div id="app">
  <p>{{ msg.split('').reverse().join('') }}</p>
</div>

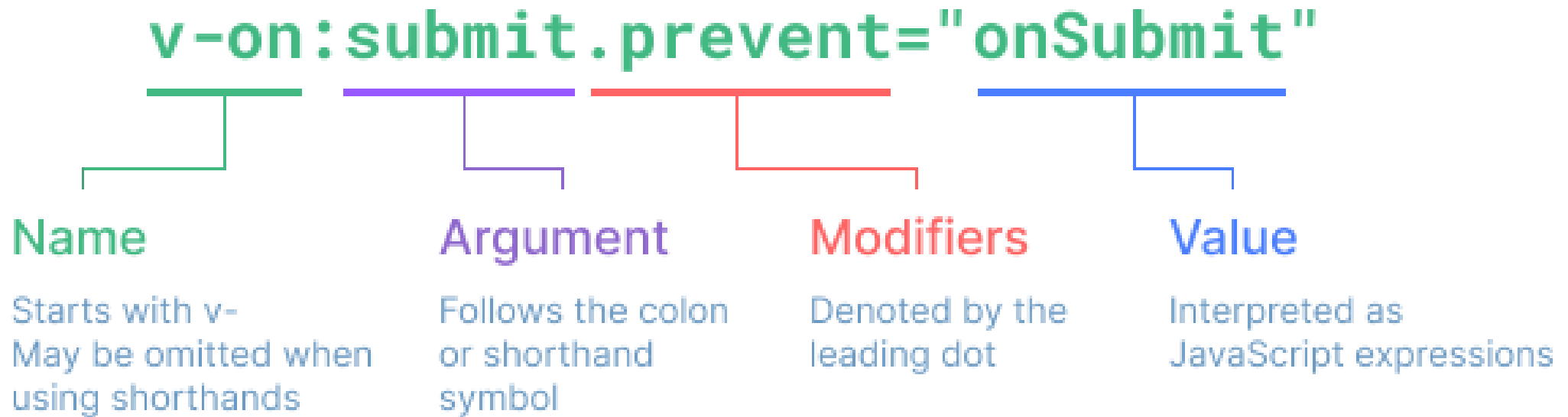
<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      msg: 'Text interpolation',
    }
  })
</script>
```

# Directives

## | Directives 기본 구성 (1/2)

- v-접두사가 있는 특수 속성에는 값을 할당 할 수 있음
  - 값에는 JS 표현식을 작성 할 수 있음
- directive의 역할은 **표현식의 값이 변경될 때 반응적으로 DOM에 적용하는 것**

## Directives 기본 구성 (2/2)



- ``:`` 을 통해 전달인자를 받을 수 있음
- ``.` 으로 표시되는 특수 접미사 - directive를 특별한 방법으로 바인딩 해야 함

## | 새 Vue instance 생성

- 06\_basic\_of\_syntax.html에서 진행
- 각각의 instance들은 연결된 DOM element에만 영향을 미침
- 연결되지 않은 DOM이 Vue의 영향을 받지 않았던 것과 동일한 상황

```
<div id="app">
  ...
</div>

<div id="app2">
</div>

<!-- Vue CDN -->
<script>
  ...
  const app2 = new Vue({
    el: '#app2',
  })
</script>
```

# Directives

## v-text

- Template Interpolation과 함께 가장 기본적인 바인딩 방법
- {{ }} 와 동일한 역할
  - 정확히 동일한 역할인 것은 아님

```
<div id="app2">
  <p v-text="message"></p>
  <!-- 같음 -->
  <p>{{ message }}</p>
</div>

<!-- Vue CDN -->
<script>
  const app2 = new Vue({
    el: '#app2',
    data: {
      message: 'Hello!',
    }
  })
</script>
```



## v-html

- RAW HTML을 표현할 수 있는 방법
- 단, 사용자가 입력하거나 제공하는 콘텐츠에는 **절대 사용 금지**
  - XSS 공격 참고

```
<div id="app2">
  ...
  <p v-html="html"></p>
</div>

<!-- Vue CDN -->
<script>
  const app2 = new Vue({
    el: '#app2',
    data: {
      ...
      html: '<a
href="https://www.google.com">GOOGLE</a>'
    }
  })
</script>
```

## v-show (1/2)

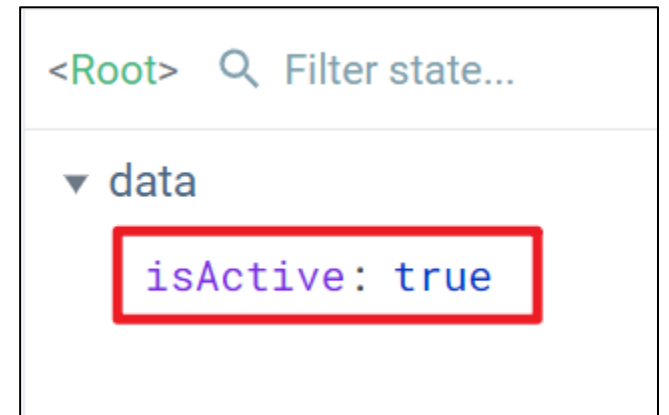
- 표현식에 작성된 값에 따라 element를 보여 줄 것인지 결정
  - boolean 값이 변경 될 때 마다 반응
- 대상 element의 display 속성을 기본 속성과 none으로 toggle
- 요소 자체는 항상 DOM에 렌더링 됨

```
<div id="app3">
  <p v-show="isActive">보이니? 안보이니?</p>
</div>

<!-- Vue CDN -->
<script>
  const app3 = new Vue({
    el: '#app3',
    data: {
      isActive: false
    }
  })
</script>
```

## v-show (2/2)

- 바인딩 된 isActive의 값이 false이므로 첫 방문 시 p tag는 보이지 않음
  - vue dev tools에서 isActive 변경 시 화면에 출력
  - 값을 false로 변경 시 다시 사라짐
- 화면에서만 사라졌을 뿐, DOM에는 존재한다.
  - display 속성이 변경되었을 뿐



# Directives

## v-if

- v-show와 사용 방법은 동일
- isActive의 값이 변경 될 때 반응
- 단, 값이 false인 경우 **DOM에서 사라짐**
- v-if v-else-if v-else 형태로 사용

```
<div id="app3">
  ...
  <p v-if="isActive">안보이니? 보이니?</p>
</div>

<!-- Vue CDN -->
<script>
  const app3 = new Vue({
    el: '#app3',
    data: {
      isActive: false
    }
  })
</script>
```

## | v-show VS v-if

- **v-show** (Expensive initial load, cheap toggle)
  - 표현식 결과와 관계 없이 렌더링 되므로 초기 렌더링에 필요한 비용은 v-if 보다 높을 수 있음
  - display 속성 변경으로 표현 여부를 판단하므로 렌더링 후 toggle 비용은 적음
- **v-if** (Cheap initial load, expensive toggle)
  - 표현식 결과가 false인 경우 렌더링조차 되지 않으므로 초기 렌더링 비용은 v-show 보다 낮을 수 있음
  - 단, 표현식 값이 자주 변경되는 경우 잦은 재 렌더링으로 비용이 증가할 수 있음

## | v-for (1/3)

- 07\_basic\_of\_syntax\_2.html에서 진행
- for .. in .. 형식으로 작성
- 반복한 데이터 타입에 모두 사용 가능
- index를 함께 출력하고자 한다면 (char, index) 형태로 사용 가능

```
<div id="app">
  <h2>String</h2>
  <!-- <div v-for="char in myStr"></div> -->
  <div v-for="(char, index) in myStr">
    <p>{{ index }}번째 문자열 {{ char }}</p>
  </div>
</div>
<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      myStr: 'Hello, World!'
    }
  })
</script>
```

## v-for (2/3)

- 배열 역시 문자열과 동일하게 사용 가능
- 각 요소가 객체라면 **dot notation**으로 접근 할 수 있음

```
<h2>Array</h2>
<div v-for="(item, index) in myArr2">
  <p>{{ index }}번째 아이템</p>
  <p>{{ item.name }}</p>
</div>

<script>
  const app = new Vue({
    data: {
      myArr2: [
        { id: 1, name: 'python'},
        ...
      ],
    }
  })
</script>
```

## [참고] 특수 속성 key

- “v-for 사용 시 반드시 key 속성을 각 요소에 작성”
- 주로 v-for directive 작성 시 사용
- vue 화면 구성 시 이전과 달라진 점을 확인 하는 용도로 활용
  - 따라서 key가 중복되어서는 안됨
- 각 요소가 고유한 값을 가지고 있다면 생략할 수 있음

```
<div
  v-for="(item, index) in myArr2"
  :key="`arry-${index}`"
>
<p>{{ index.id }}번째 아이템</p>
<p>{{ item.name }}</p>
</div>
```



## | v-for (3/3)

- 객체 순회 시 value가 할당되어 출력
- 2번째 변수 할당 시 key 출력 가능

```
<h2>Object</h2>
<div v-for="(value, key) in myObj" :key="key">
  <p>{{ key }} : {{ value }}</p>
</div>

<script>
  const app = new Vue({
    data: {
      ...,
      myObj: {
        name: 'harry',
        age: 27
      },
    }
  })
</script>
```

## v-on (1/2)

- 08\_basic\_of\_syntax\_3.html에서 진행
- `:` 을 통해 전달받은 인자를 확인
- 값으로 JS 표현식 작성
- addEventListener의 첫 번째 인자와 동일한 값들로 구성
- 대기하고 있던 이벤트가 발생하면 할당된 표현식 실행

```
<div id="app">
  <button v-on:click="number++">
    increase Number
  </button>
  <p>{{ number }}</p>
</div>

<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      number: 0,
    },
  })
</script>
```

## v-on (2/2)

- method를 통한 data 조작도 가능
- method에 인자를 넘기는 방법은 일반 함수를 호출할 때와 동일한 방식
- `:` 을 통해 전달된 인자에 따라 특별한 modifiers (수식어)가 있을 수 있음
  - ex) v-on:keyup.enter 등
  - vue2 가이드 > api > v-on 파트 참고
- `@` shortcut 제공
  - ex) @keyup.click



## v-bind (1/2)

- HTML 기본 속성에 Vue data를 연결
- class의 경우 다양한 형태로 연결 가능
  - 조건부 바인딩
    - { 'class Name': '조건 표현식' }
    - 삼항 연산자도 가능
  - 다중 바인딩
    - ['JS 표현식', 'JS 표현식', ...]

```
<div id="app2">
  <a v-bind:href="url">Go To GOOGLE</a>
</div>

<!-- Vue CDN -->
<script>
  const app2 = new Vue({
    el: '#app2',
    data: {
      url: 'https://www.google.com/',
    },
  })
</script>
```

## | v-bind (2/2)

- Vue data의 변화에 반응하여 DOM에 반영하므로 상황에 따라 유동적 할당 가능
- ``:`` shortcut 제공
  - ex) `:class` 등
  - v-for 에서 사용하였던 `:key`는 v-bind의 shortcut을 활용한 것

# Directives

## v-model

- 09\_basic\_of\_syntax\_4.html에서 진행
- Vue instance와 DOM의 양방향 바인딩
- Vue data 변경 시 v-model로 연결된 사용자 입력 element에도 적용

```
<div id="app">
  <h3>{{ myMessage }}</h3>
  <input v-model="myMessage" type="text">
  <hr>
</div>

<!-- Vue CDN -->
<script>
  const app = new Vue({
    el: '#app',
    data: {
      myMessage: '',
    },
  })
</script>
```

# 이어서 ..

삼성 청년 SW 아카데미

# Vue advanced



## computed

- Vue instance가 가진 options 중 하나
- computed 객체에 정의한 함수를 페이지가 최초로 렌더링 될 때 호출하여 계산
  - 계산 결과가 변하기 전까지 함수를 재호출하는 것이 아닌 계산된 값을 반환
- **10\_computed.html**에서 methods와의 차이 확인

## methods VS computed

- methods

- 호출 될 때마다 함수를 실행
- 같은 결과여도 매번 새롭게 계산

- computed

- 함수의 종속 대상의 변화에 따라 계산 여부가 결정됨
- 종속 대상이 변하지 않으면 항상 저장(캐싱)된 값을 반환

## watch (1/2)

- 11\_watch.html에서 결과 확인
- 특정 데이터의 변화를 감지하는 기능
  1. watch 객체를 정의
  2. 감시할 대상 data를 지정
  3. data가 변할 시 실행 할 함수를 정의
- 첫 번째 인자는 변동 후 data
- 두 번째 인자는 변동 전 data

```
<button @click="number++"></button>

<script>
  const app = new Vue({
    data: {
      number: 0,
    },
    watch: {
      number: function(val, oldVal) {
        console.log(val, oldVal)
      },
    }
  })
</script>
```

## watch (2/2)

- 실행 함수를 Vue method로 대체 가능
  1. 감시 대상 data의 이름으로 객체 생성
  2. 실행하고자 하는 method를 handler에 문자열 형태로 할당
- Array, Object의 내부 요소 변경을 감지를 위해서는 **deep** 속성 추가 필요

## filters

- 텍스트 형식화를 적용할 수 있는 필터
- interpolation 혹은 v-bind를 이용할 때 사용 가능
- 필터는 자바스크립트 표현식 마지막에 ``|`` (파이프)와 함께 추가되어야 함
- 이어서 사용(chaining) 가능
- `12_filters.html`에서 결과 확인

# 이어서 ..

삼성 청년 SW 아카데미

# Vue Style Guide

## Vue Style Guide

- Vue의 스타일 가이드 규칙은 우선순위를 기준으로 4가지 범주를 설정
- 우선순위
  - A: 필수 (Essential)
  - B: 적극 권장 (Strongly Recommended)
  - C: 권장 (Recommended)
  - D: 주의 필요 (Use with Caution)
- <https://v2.vuejs.org/v2/style-guide/>



## 우선순위 특징

- A: 필수 (Essential)
  - 오류를 방지하는 데 도움이 되므로 어떤 경우에도 규칙을 학습하고 준수
- B: 적극 권장 (Strongly Recommended)
  - 규칙을 어겨도 코드는 여전히 실행되겠지만, 규칙 위반은 드물어야 함
- C: 권장 (Recommended)
  - 일관성을 보장하도록 임의의 선택을 할 수 있음
- D: 주의 필요 (Use with Caution)
  - 잠재적 위험 특성을 고려함

## | 오늘 학습하고 지켜야 할 스타일 가이드 2가지

- 우선순위 A
  1. v-for는 항상 key와 함께 사용하기
  2. v-for를 쓴 엘리먼트에 절대 v-if를 사용하지 말기

## 1. v-for는 항상 key와 함께 사용하기

- 내부 컴포넌트의 상태를 일관되게 유지하기 위해 v-for에 항상 key를 사용하기
- 데이터의 예측 가능한 행동을 유지 시키기 (객체 불변성)

```
todos: [  
  { id: 1, text: 'Learn to use v-for' },  
  { id: 2, text: 'Learn to use key' }  
]
```

```
<!-- bad -->  
  
<ul>  
  <li v-for="todo in todos">  
    {{ todo.text }}  
  </li>  
</ul>
```



```
<!-- good -->  
  
<ul>  
  <li  
    v-for="todo in todos"  
    :key="todo.id"  
  >  
    {{ todo.text }}  
  </li>  
</ul>
```

## | 2. v-for를 쓴 엘리먼트에 절대 v-if를 사용하지 말기

- 함께 쓸 수 있다고 생각되는 2가지 경우
  1. 목록의 항목을 필터링할 때  
(`v-for="user in users" v-if="user.isActive"`)
  2. 숨김 목록의 렌더링을 피할 때  
(`v-for="user in users" v-if="shouldShowUsers"`)

## 1. 목록의 항목을 필터링할 때

- Vue가 디렉티브를 처리할 때 v-for는 v-if보다 우선순위가 높음
- 즉 user의 일부분만 렌더링하고 싶은데도 불구하고, v-for가 우선순위를 가지기 때문에 모든 user에 대해 반복해야 함

```
<!-- bad -->

<ul>
  <li
    v-for="user in users"
    v-if="user.isActive"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

computed를 사용해서 isActive가 true인 user만  
반복하게 해서 효율적으로 렌더링할 수 있도록 함

```
computed: {
  activeUsers() {
    return this.users.filter((user) => user.isActive)
  }
}
```

```
<!-- good -->

<ul>
  <li
    v-for="user in activeUsers"
    :key="user.id"
  >
    {{ user.name }}
  </li>
</ul>
```

## 2. 숨김 목록의 렌더링을 피할 때

- v-if를 컨테이너 엘리먼트로 옮기기
  - 더 이상 목록의 모든 사용자에게 대해 shouldShowUsers를 확인하지 않도록 함
- 한 번 확인하고 shouldShowUsers가 false인 경우 v-for를 평가하지도 않음

```
<!-- bad -->  
  
<ul>  
  <li  
    v-for="user in users"  
    v-if="shouldShowUsers"  
    :key="user.id"  
  >  
    {{ user.name }}  
  </li>  
</ul>
```



```
<!-- good -->  
  
<ul v-if="shouldShowUsers">  
  <li  
    v-for="user in users"  
    :key="user.id"  
  >  
    {{ user.name }}  
  </li>  
</ul>
```

## | Vue Style Guide 정리

- 우선순위에 상관없이 최대한 스타일 가이드를 지키며 작성하기
- <https://v2.vuejs.org/v2/style-guide/>

# 이어서 ..

삼성 청년 SW 아카데미



# 마무리

# 마무리

- Vue intro
- Why Vue
- Vue instance
- Basic of syntax
- Vue advanced

# 다음 방송에서 만나요!

삼성 청년 SW 아카데미