

Vue 01

목차

- Intro to Vue.js
- Quick Start
- Basic syntax

Gitlab

- 오늘 진행하는 수업 자료는 아래 gitlab repository에 업로드 됩니다.

<https://lab.ssafy.com/05/a/vuejs>

■ 사전 준비사항

- 아래 공용 노션 문서에서 “Vue.js 사전 준비사항” 설정

<http://bit.ly/05-document>

이어서..

삼성 청년 SW 아카데미

지난 시간

JavaScript 기초

- **DOM(Document Object Model)**

- DOM Tree
- DOM Manipulation

- **EventListener**

- Event
- `addEventListener(type, listener)`

JavaScript 심화

- **ECMAScript**
 - 프로그래밍 언어
- **AJAX(XHR)**
 - 비동기(요청)
- **Callback Function**
- **Promise**
 - `.then & .catch`
 - Axios

이어서..

삼성 청년 SW 아카데미

오늘의 주제

Vue.js Intro

Front-End Development

- 프론트엔드 개발은 HTML, CSS, 그리고 JavaScript를 활용해서 데이터를 볼 수 있게 만들어 줌
 - 이 작업을 통해 사용자는 데이터를 눈으로 볼 수 있고 데이터와 상호작용 할 수 있음
- 대표적인 프론트엔드 프레임워크
 - Vue.js, React, Angular

What is Vue.js

- 사용자 인터페이스를 만들기 위한 프로그레시브 프레임워크
- 현대적인 tool과 다양한 라이브러리를 통해 SPA(Single Page Application)를 완벽하게 지원
- Evan You에 의해 발표 (2014)
 - 구글의 Angular 개발자 출신
 - 학사 미술, 미술사 전공/ 석사 디자인 & 테크놀로지 전공
 - 구글 Angular 보다 더 가볍고, 간편하게 사용할 수 있는 프레임워크를 만들기 위해 개발

SPA

- 단일 페이지 애플리케이션(Single Page Application)
- 현재 페이지를 동적으로 작성함으로써 사용자와 소통하는 웹 애플리케이션
- 단일 페이지로 구성되며 서버로부터 처음에만 페이지를 받아오고 이후에는 **동적으로 DOM을 구성**
 - 즉, 처음 페이지를 받은 이후에는 서버로부터 완전한 새로운 페이지를 불러오지 않고 현재 페이지를 동적으로 다시 작성함
 - 연속되는 페이지 간의 **사용자 경험(UX)을 향상**
 - 모바일 사용이 증가하고 있는 현재 트래픽의 감소와 속도, 사용성, 반응성의 향상은 매우 중요한 이슘이기 때문
- 동작 원리의 일부가 **CSR**의 구조를 따름

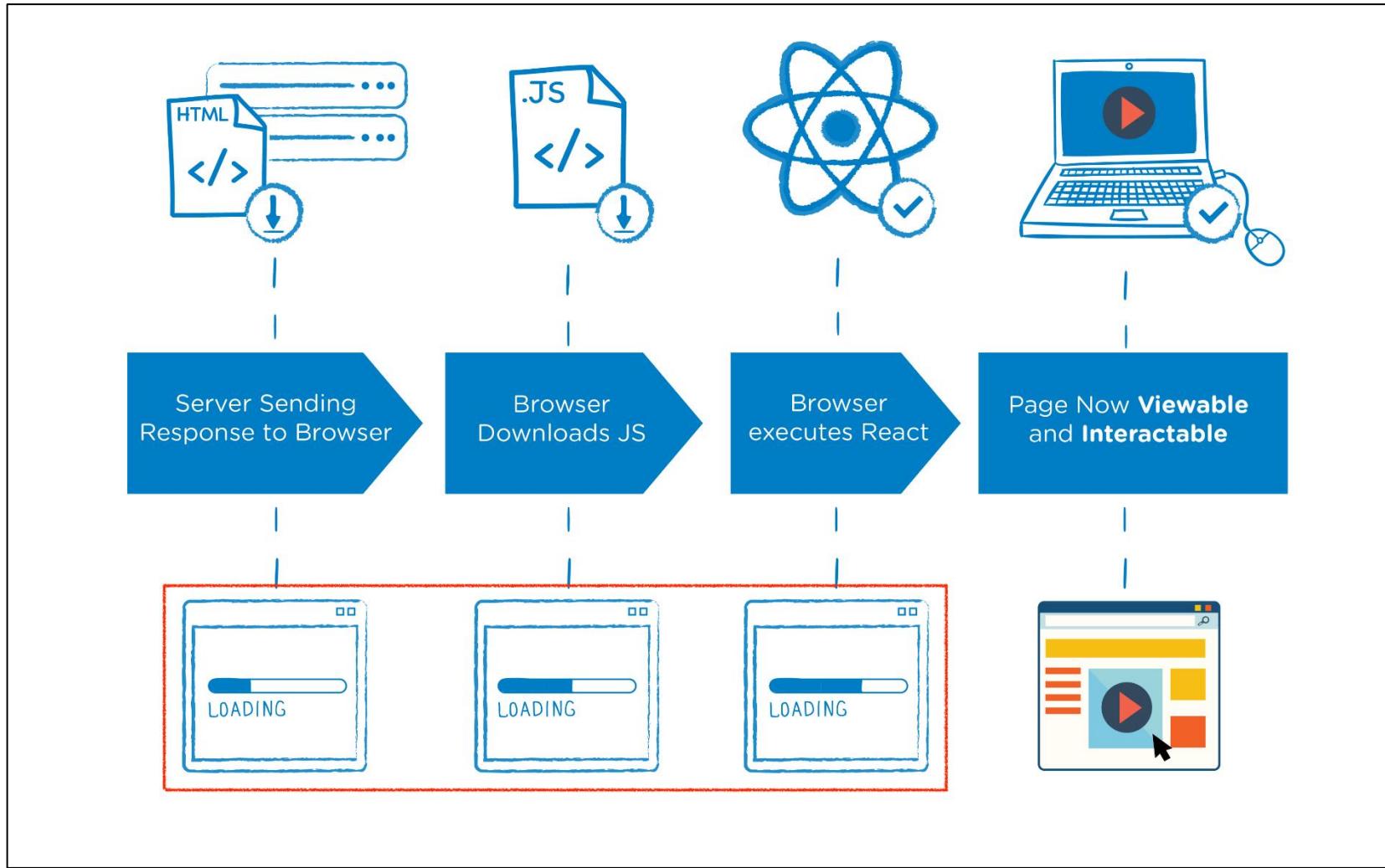
SPA 등장 배경

- 과거 웹 사이트들은 요청에 따라 매번 새로운 페이지를 응답하는 방식이었음
 - Multi Page Form (MPA)
- 스마트폰이 등장하면서 모바일 최적화에 대한 필요성이 생김
 - 모바일 네이티브 앱과 같은 형태의 웹 페이지가 필요해짐
- 이러한 문제를 해결하기 위해 Vue와 같은 프론트엔드 프레임워크가 등장
 - CSR, SPA의 등장
- 1개의 웹페이지에서 여러 동작이 이루어지며 모바일 앱과 비슷한 형태의 사용자 경험을 제공

CSR

- Client Side Rendering
- 최초 요청 시 서버에서 빈 문서를 응답하고 이후 클라이언트에서 데이터를 요청해 데이터를 받아 DOM을 렌더링하는 방식
- SSR보다 초기 전송되는 페이지의 속도는 빠르지만, 서비스에서 필요한 데이터를 클라이언트(브라우저)에서 추가로 요청하여 재구성해야 하기 때문에 전체적인 페이지 완료 시점은 SSR보다 느림
- SPA가 사용하는 렌더링 방식

CSR



CSR

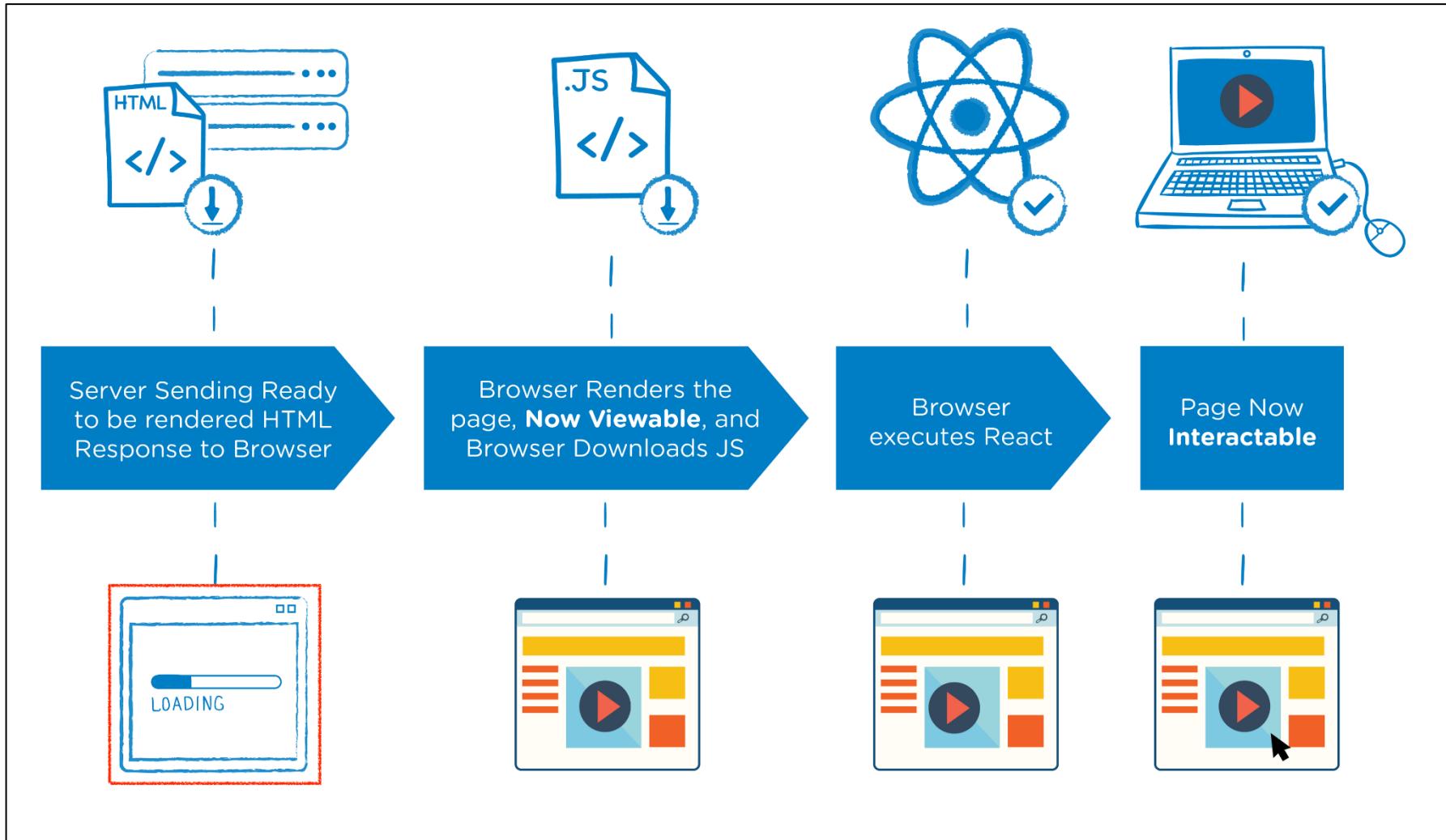
- 장점
 - 서버와 클라이언트 간의 트래픽 감소
 - 웹 애플리케이션에 필요한 모든 정적 리소스를 최초에 한번 다운로드
 - 사용자 경험 향상
 - 전체 페이지를 다시 렌더링하지 않고 변경되는 부분만을 갱신
- 단점
 - SEO(검색엔진 최적화) 문제가 발생할 수 있음

SSR

- Server Side Rendering
- 서버에서 사용자에게 보여줄 페이지를 모두 구성하여 사용자에게 페이지를 보여주는 방식
- 서버를 이용해서 페이지를 구성하기 때문에
클라이언트에서 구성하는 CSR보다 페이지를 구성하는 속도는 늦지만
사용자에게 보여주는 콘텐츠 구성이 완료되는 시점은 빨라짐

Vue.js Intro

SSR



SSR

- 장점
 - 초기 로딩 속도가 빠르기 때문에 사용자가 컨텐츠를 빨리 볼 수 있음
 - SEO(검색엔진 최적화)가 가능
- 단점
 - 모든 요청에 새로고침이 되기 때문에 사용자 경험이 떨어짐
 - 상대적으로 요청 횟수가 많아져 서버 부담이 커짐

SEO

- Search Engine Optimization (검색 엔진 최적화)
- 웹 페이지 검색엔진이 자료를 수집하고 순위를 매기는 방식에 맞게 웹페이지를 구성해서 검색 결과의 상위에 노출될 수 있도록 하는 작업
- 인터넷 마케팅 방법 중 하나
- 구글 등장 이후 검색엔진들이 컨텐츠의 신뢰도를 파악하는 기초 지표로 사용됨
 - 다른 웹 사이트에서 얼마나 인용되었나를 반영
 - 결국 타 사이트에 인용되는 횟수를 늘리는 방향으로 최적화

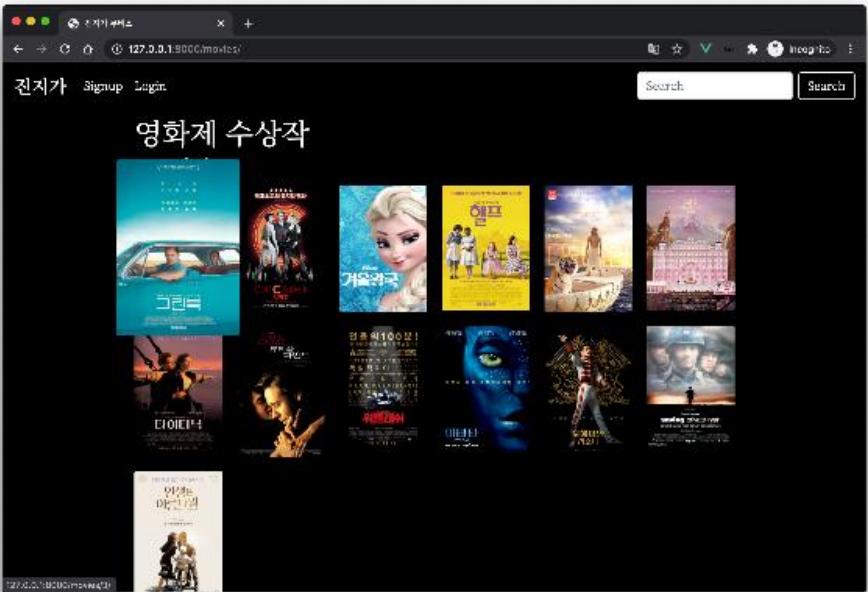
SEO 문제 대응

- Vue.js 또는 React 등의 SPA 프레임워크는 SSR을 지원하는 SEO 대응 기술이 이미 존재
 - SEO 대응이 필요한 페이지에 대해서는 선별적 SEO 대응 가능
- 혹은 추가적으로 프레임워크를 사용하기도 함
 - Nuxt.js
 - Vue.js 응용 프로그램을 만들기 위한 Framework
 - SSR을 지원
 - Next.js
 - React 응용 프로그램을 만들기 위한 Framework
 - SSR을 지원

SPA with SSR

- CSR과 SSR을 적절히 사용
 - 예를 들어, Django에서 Axios를 활용한 좋아요/팔로우 로직의 경우 대부분은 Server에서 완성된 HTML을 제공하는 구조 (SSR)
 - 다만, 특정 요소(좋아요/팔로우)만 AJAX를 활용한 비동기요청으로 필요한 데이터를 Client에서 서버로 직접 요청을 보내 받아오고 JS를 활용해 DOM을 조작 (CSR)

Vue.js

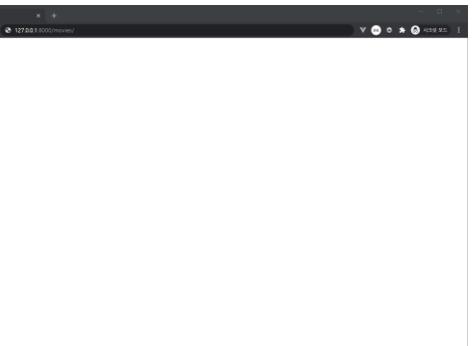
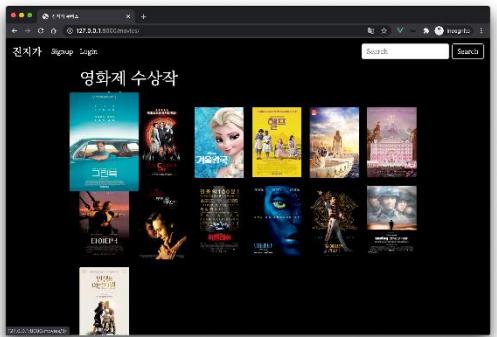


HTML



Server

Vue.js



Server

이어서..

삼성 청년 SW 아카데미

Why Vue.js ?

Github star ranking

- 가장 인기있는 프론트엔드 프레임워크
- 하지만 더 근본적인 이유는 따로 있음

Organizations Ranking

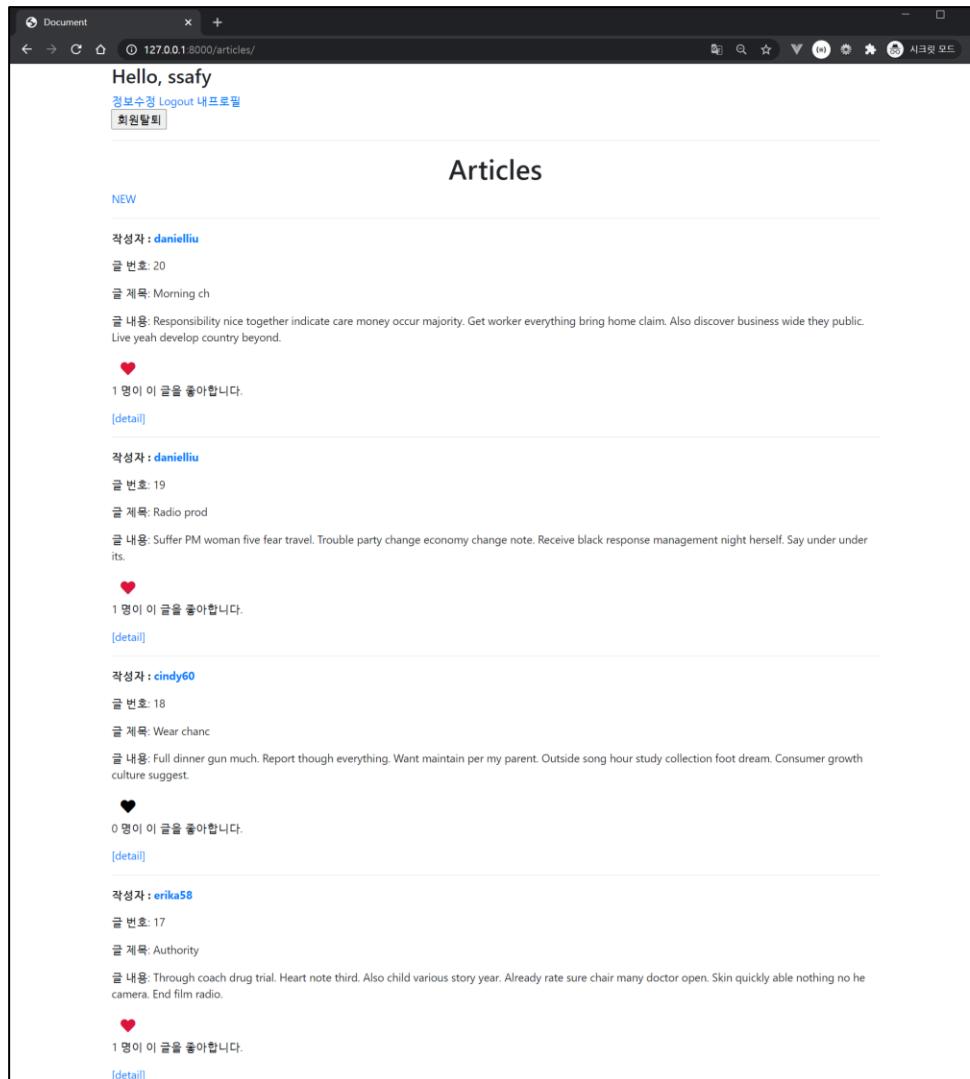
	1. microsoft	★ 1388439
	2. google	★ 1286998
	3. facebook	★ 824800
	4. apache	★ 679618
	5. alibaba	★ 545391
	6. vuejs	★ 464642
	7. tensorflow	★ 387632
	8. freeCodeCamp	★ 365424
	9. Tencent	★ 355753

Why Vue.js ?

Vue.js는 왜 사용해야 할까?

- 현대 웹 페이지는 페이지 규모가 계속해서 커지고 있으며
그만큼 사용하는 데이터도 늘어나고
사용자와의 상호작용도 많이 이루어짐
- 결국 Vanilla JS 만으로는 관리하기가 어려움
 - 예시) “페이스북의 친구가 이름을 변경했을 경우 변경되어야 하는 것들”
 - 타임라인의 이름, 페이스북 메시지 상의 이름, 내 주소록에서의 친구 이름 등
 - → 페이스북이 React를 개발한 이유

LIKE with Vanilla JavaScript



LIKE with Vanilla JavaScript

```
<form class="d-inline like-form" data-id="{{ article.pk }}>
  {% csrf_token %}
  {% if user in article.like_users.all %}
    <button class="btn btn-link">
      <i id="like-{{ article.pk }}" class="fas fa-heart fa-lg" style="color:crimson;">
        </i>
    </button>
  {% else %}
    <button class="btn btn-link">
      <i id="like-{{ article.pk }}]" class="fas fa-heart fa-lg" style="color:black;"></i>
    </button>
  {% endif %}
</form>
<p>
  <span id="like-count-{{ article.pk }}>
    {{ article.like_users.all|length }}
  </span> 명이 이 글을 좋아합니다.
</p>
```

1. 선택 - 무엇을

```
const forms = document.querySelectorAll('.like-form')

forms.forEach(function (form) {
  form.addEventListener('submit', function (event) {
    event.preventDefault()

    const articleId = event.target.dataset.articleId
    const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value

    axios.post(`http://127.0.0.1:8000/articles/${articleId}/like/`, {}, {
      headers: {
        'X-CSRFToken': csrftoken
      }
    })
    .then(function (res) {
      const count = res.data.count
      const liked = res.data.liked

      const likeIconColor = document.querySelector(`#like-${articleId}`)
      const likeCount = document.querySelector(`#like-count-${articleId}`)

      likeCount.innerText = `${count} 명이 이 글을 좋아합니다.`

      if (liked) {
        likeIconColor.style.color = 'crimson'
      } else {
        likeIconColor.style.color = 'black'
      }
    })
  })
})
```

3. 변경

LIKE with Vanilla JavaScript

하지만 선택해야하는 데이터와
동시에 변경되어야 하는 요소가 많아진다면..?

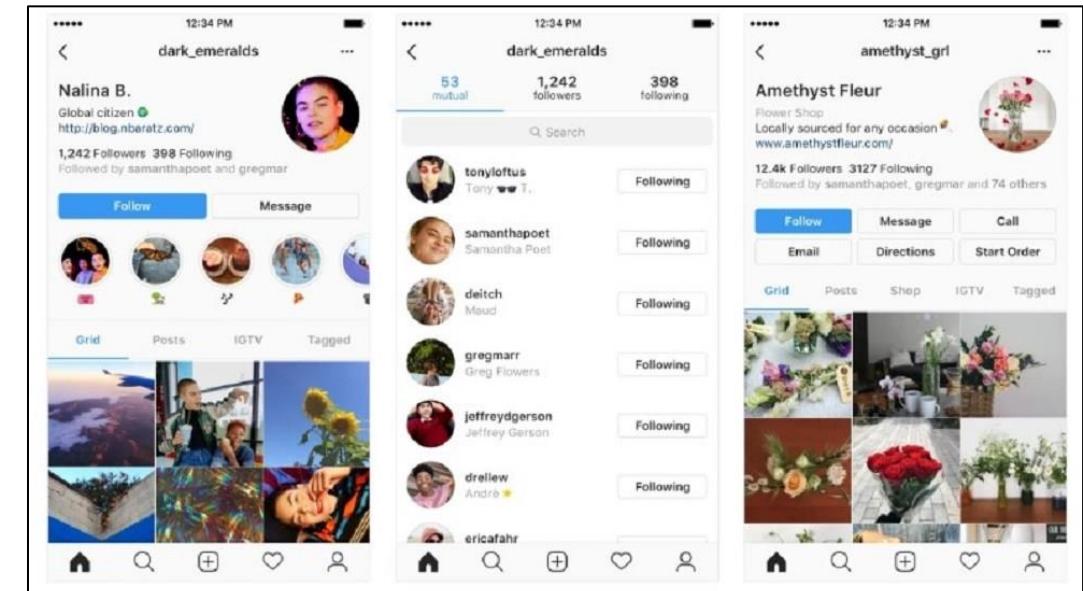
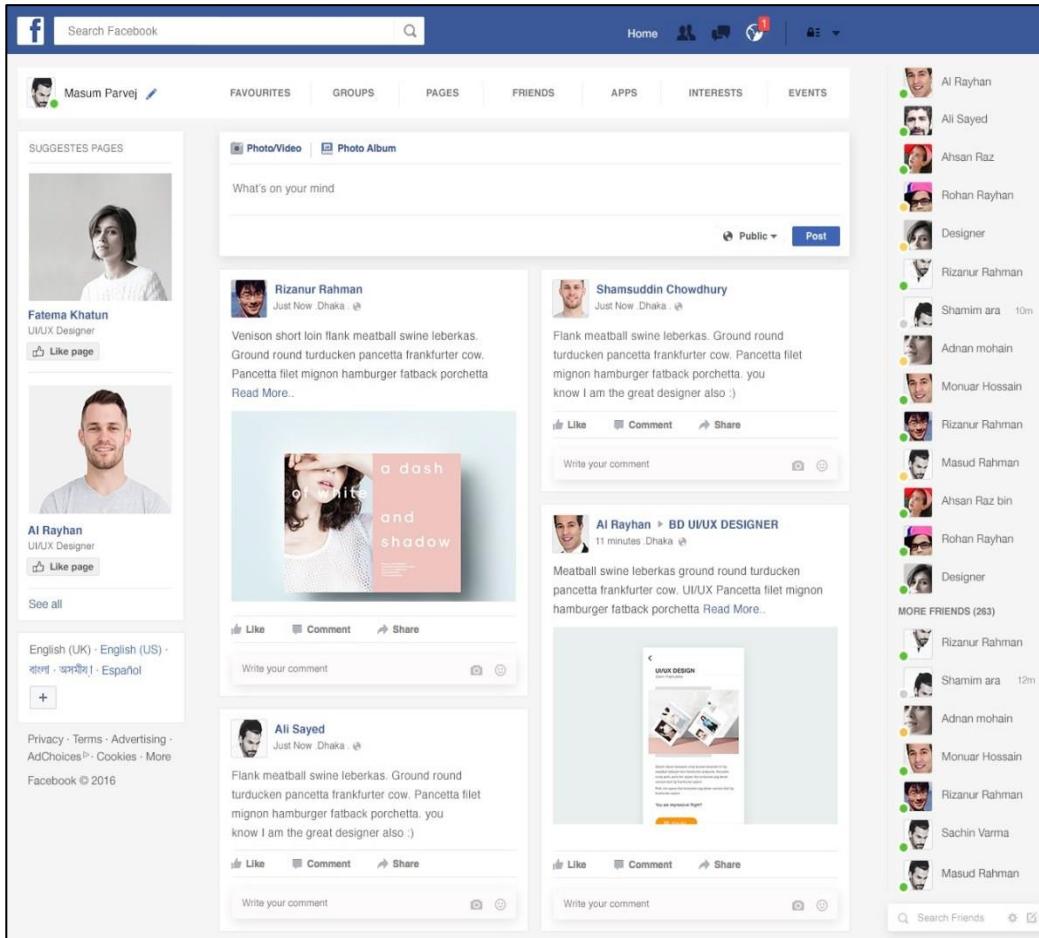
```
<form class="d-inline like-form" data-id="{{ article.pk }}>  
  {% csrf_token %}  
  {% if user in article.like_users.all %}  
    <button class="btn btn-link">  
      <i id="like-{{ article.pk }}" class="fas fa-heart fa-lg" style="color:crimson;">  
    </i>  
    </button>  
  {% else %}  
    <button class="btn btn-link">  
      <i id="like-{{ article.pk }}]" class="fas fa-heart fa-lg" style="color:black;"></i>  
    </button>  
  {% endif %}  
</form>  
<p>  
  <span id="like-count-{{ article.pk }}>  
    {{ article.like_users.all|length }}  
  </span> 명이 이 글을 좋아합니다.  
</p>
```

1. 선택 - 무엇을

```
const forms = document.querySelectorAll('.like-form')  
  
forms.forEach(function (form) {  
  form.addEventListener('submit', function (event) {  
    event.preventDefault()  
  
    const articleId = event.target.dataset.articleId  
    const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value  
  
    axios.post(`http://127.0.0.1:8000/articles/${articleId}/like/`, {}, {  
      headers: {  
        'X-CSRFToken': csrftoken  
      }  
    })  
    .then(function (res) {  
      const count = res.data.count  
      const liked = res.data.liked  
  
      const likeIconColor = document.querySelector(`#like-${articleId}`)  
      const likeCount = document.querySelector(`#like-count-${articleId}`)  
  
      likeCount.innerText = `${count} 명이 이 글을 좋아합니다.`  
  
      if (liked) {  
        likeIconColor.style.color = 'crimson'  
      } else {  
        likeIconColor.style.color = 'black'  
      }  
    })  
  })  
})
```

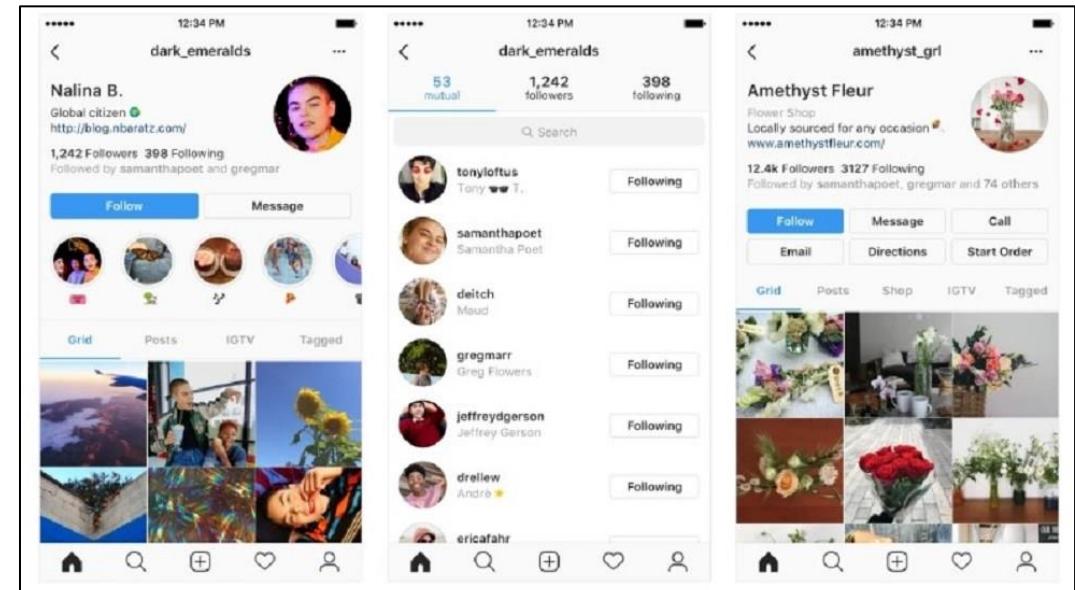
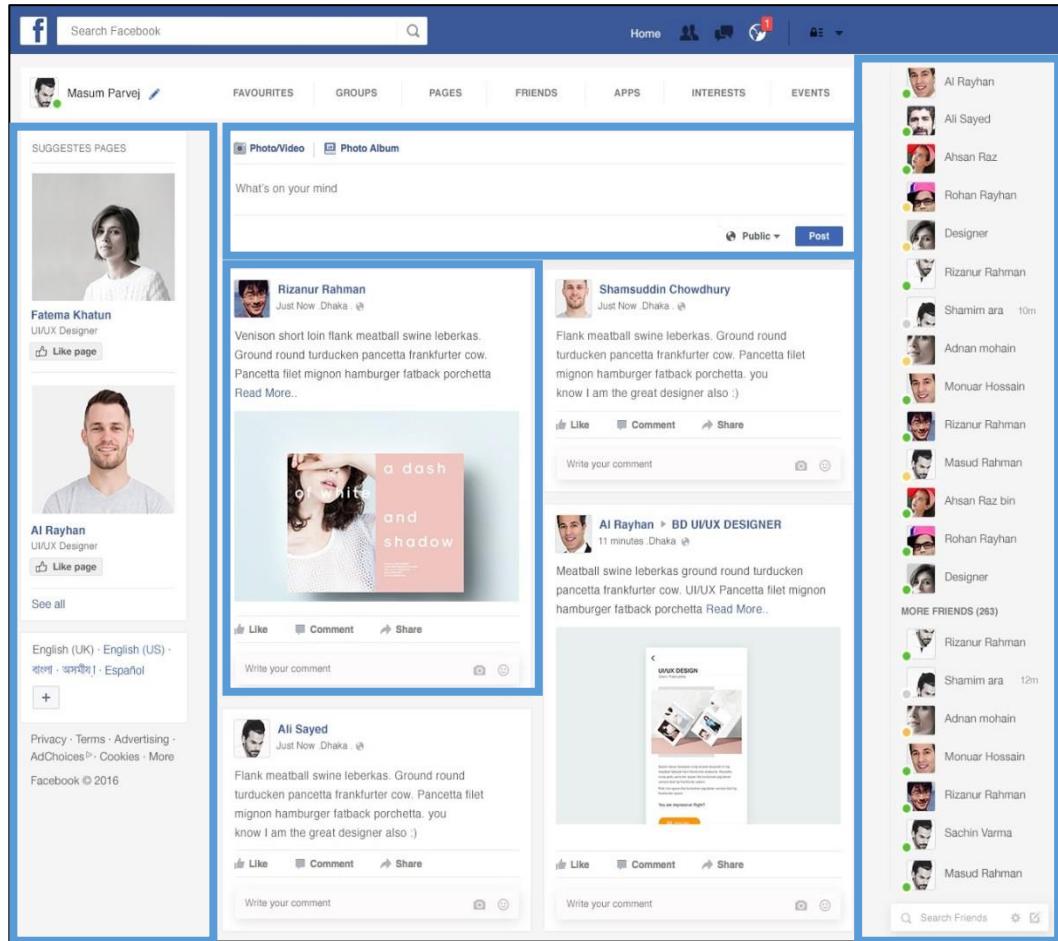
3. 변경

페이스북 예시



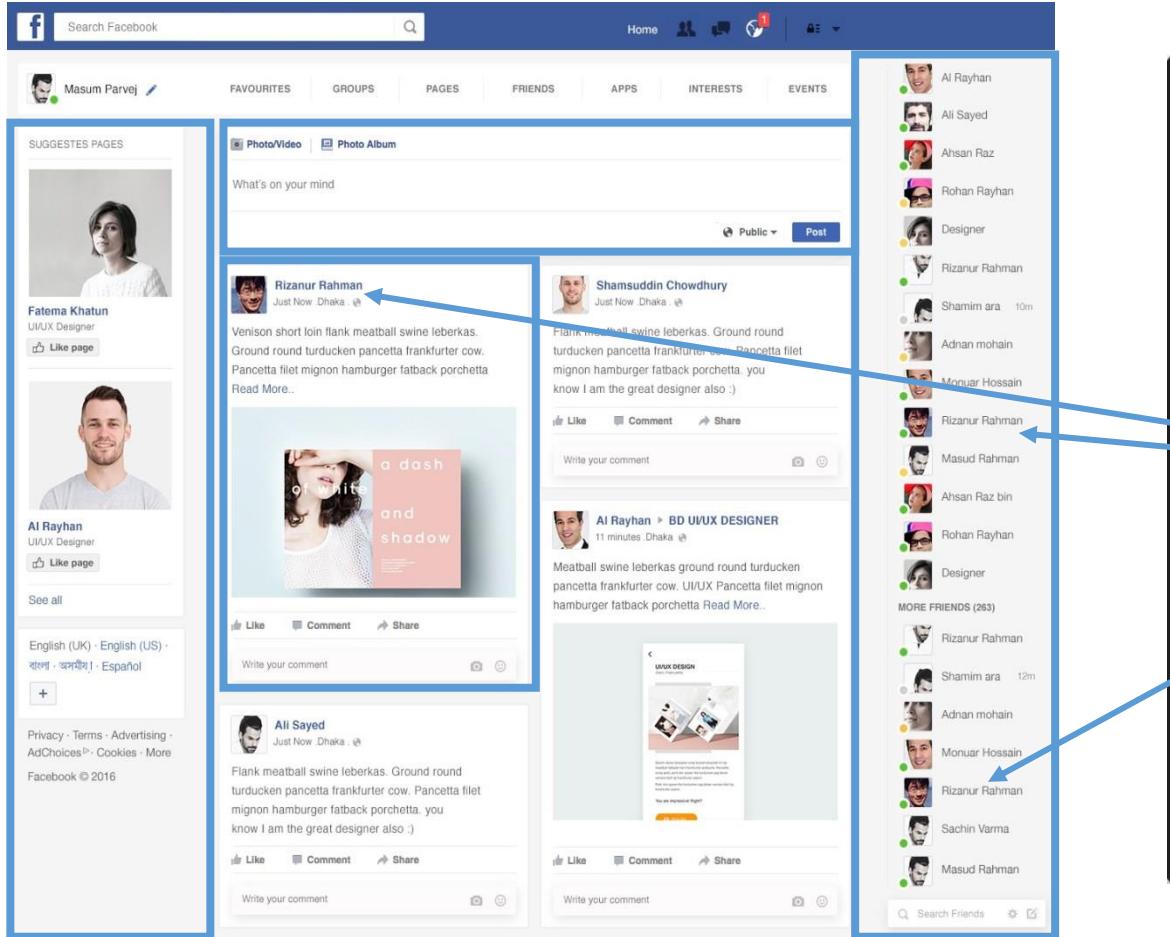
출처: <https://www.designideas.pics/facebook-home-page-redesign/>

페이스북 예시



출처: <https://www.designideas.pics/facebook-home-page-redesign/>

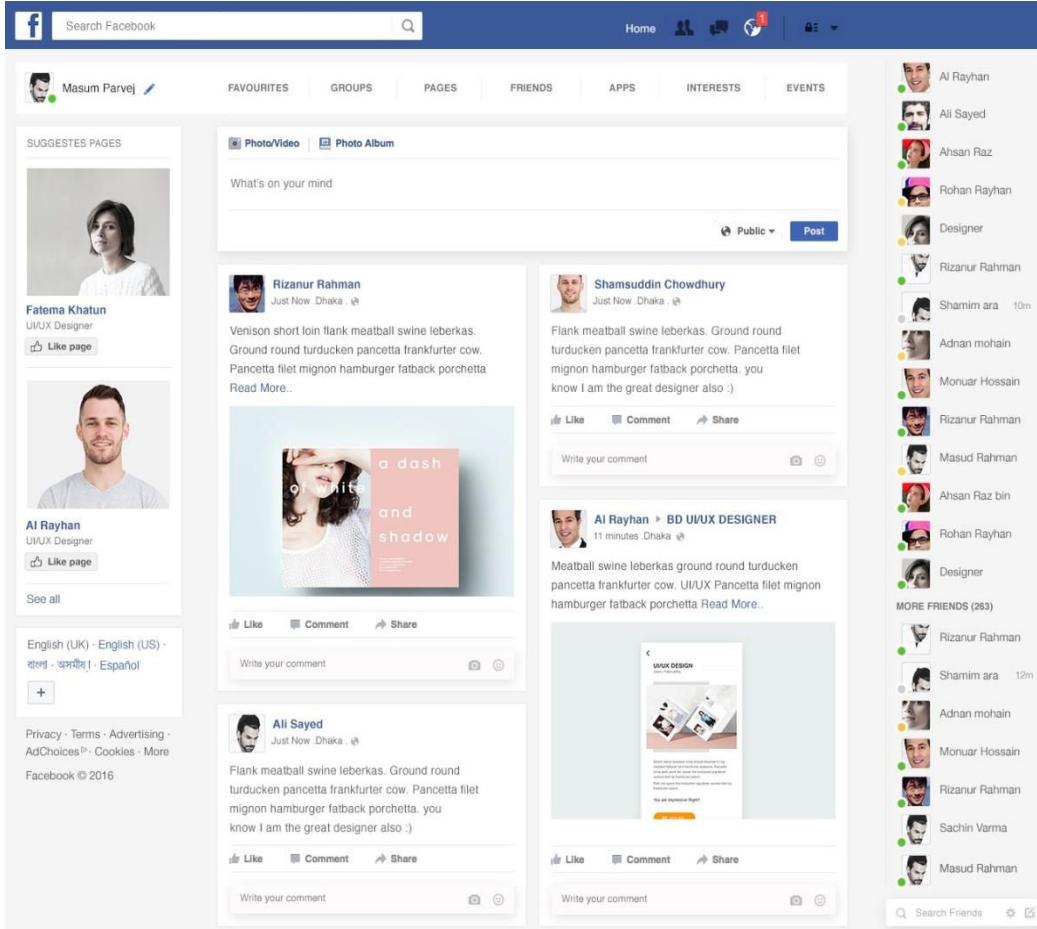
페이스북 예시



```
[  
  {  
    "username": "Fatema Khatun",  
    "loggedIn": false,  
  },  
  {  
    "username": "Rizanur Rahman",  
    "loggedIn": true,  
  },  
  {  
    "username": "Masum Pavej",  
    "loggedIn": false,  
  }]
```

출처: <https://www.designideas.pics/facebook-home-page-redesign/>

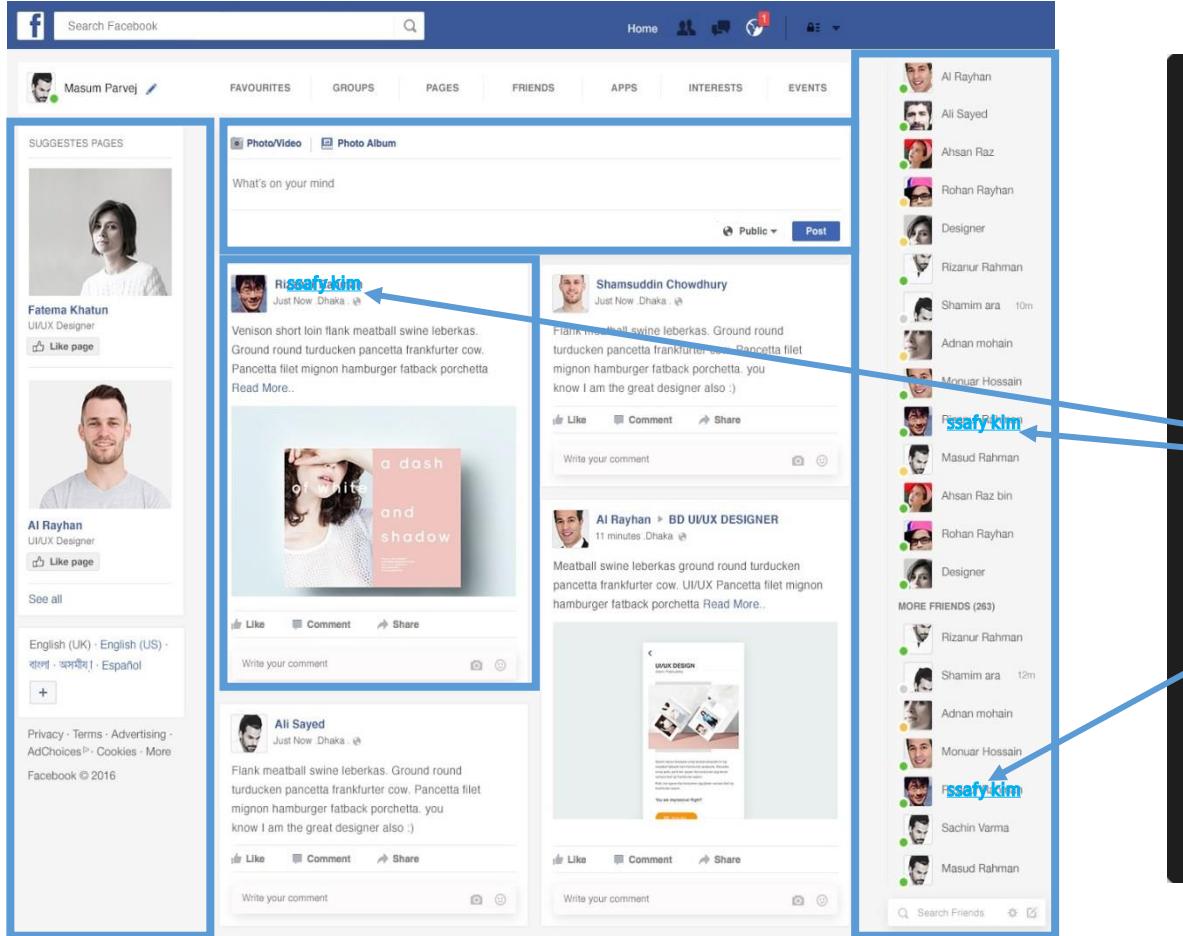
페이스북 예시 - 데이터 통한 DOM 조작



```
[  
  {  
    "username": "Fatema Khatun",  
    "loggedIn": false,  
  },  
  {  
    "username": "ssafy kim",  
    "loggedIn": true,  
  },  
  {  
    "username": "Masum Pavej",  
    "loggedIn": false,  
  }]  
[  
  {  
    "id": 1,  
    "name": "Al Rayhan",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic1.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#4CAF50",  
    "statusDot": true  
  },  
  {  
    "id": 2,  
    "name": "Ali Sayed",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic2.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 3,  
    "name": "Ahsan Raz",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic3.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 4,  
    "name": "Rohan Rayhan",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic4.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FFC107",  
    "statusDot": true  
  },  
  {  
    "id": 5,  
    "name": "Designer",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic5.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 6,  
    "name": "Rizanur Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic6.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 7,  
    "name": "Shamim ara",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic7.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 8,  
    "name": "Adnan mohain",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic8.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 9,  
    "name": "Monuar Hossain",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic9.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 10,  
    "name": "Rizanur Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic10.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 11,  
    "name": "Masud Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic11.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 12,  
    "name": "Ahsan Raz bin",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic12.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 13,  
    "name": "Rohan Rayhan",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic13.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 14,  
    "name": "Designer",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic14.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 15,  
    "name": "MORE FRIENDS (263)",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic15.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 16,  
    "name": "Rizanur Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic16.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 17,  
    "name": "Shamim ara",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic17.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 18,  
    "name": "Adnan mohain",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic18.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 19,  
    "name": "Monuar Hossain",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic19.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 20,  
    "name": "Rizanur Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic20.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  },  
  {  
    "id": 21,  
    "name": "Sachin Varma",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic21.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF9800",  
    "statusDot": true  
  },  
  {  
    "id": 22,  
    "name": "Masud Rahman",  
    "profilePic": "https://www.designideas.pics/facebook-home-page-redesign/_/profilepic22.jpg",  
    "status": "UI/UX Designer",  
    "statusColor": "#FF5722",  
    "statusDot": true  
  }]
```

출처: <https://www.designideas.pics/facebook-home-page-redesign/>

페이스북 예시 - 데이터 통한 DOM 조작



```
[  
  {  
    "username": "Fatema Khatun",  
    "loggedIn": false,  
  },  
  {  
    "username": "ssafy kim",  
    "loggedIn": true,  
  },  
  {  
    "username": "Masum Pavej",  
    "loggedIn": false,  
  }]  
]
```

출처: <https://www.designideas.pics/facebook-home-page-redesign/>

Reactive - Vanilla JavaScript 예시



```
<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
})
```

Reactive - Vue.js 예시

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

결론

- Vanilla JS
 - 한 유저가 100만개의 게시글을 작성했다고 가정
 - 이 유저가 닉네임을 변경하면, 100만개의 게시글의 작성자 이름이 모두 수정되어야 함
 - ‘모든 요소’를 선택해서 ‘이벤트’를 등록하고 값을 변경해야 함
- Vue.js
 - DOM과 Data가 연결되어 있으면
 - Data를 변경하면 이에 연결된 DOM은 알아서 변경
 - 즉, 우리가 신경 써야 할 것은 오로지 **Data**에 대한 관리 뿐

이어서..

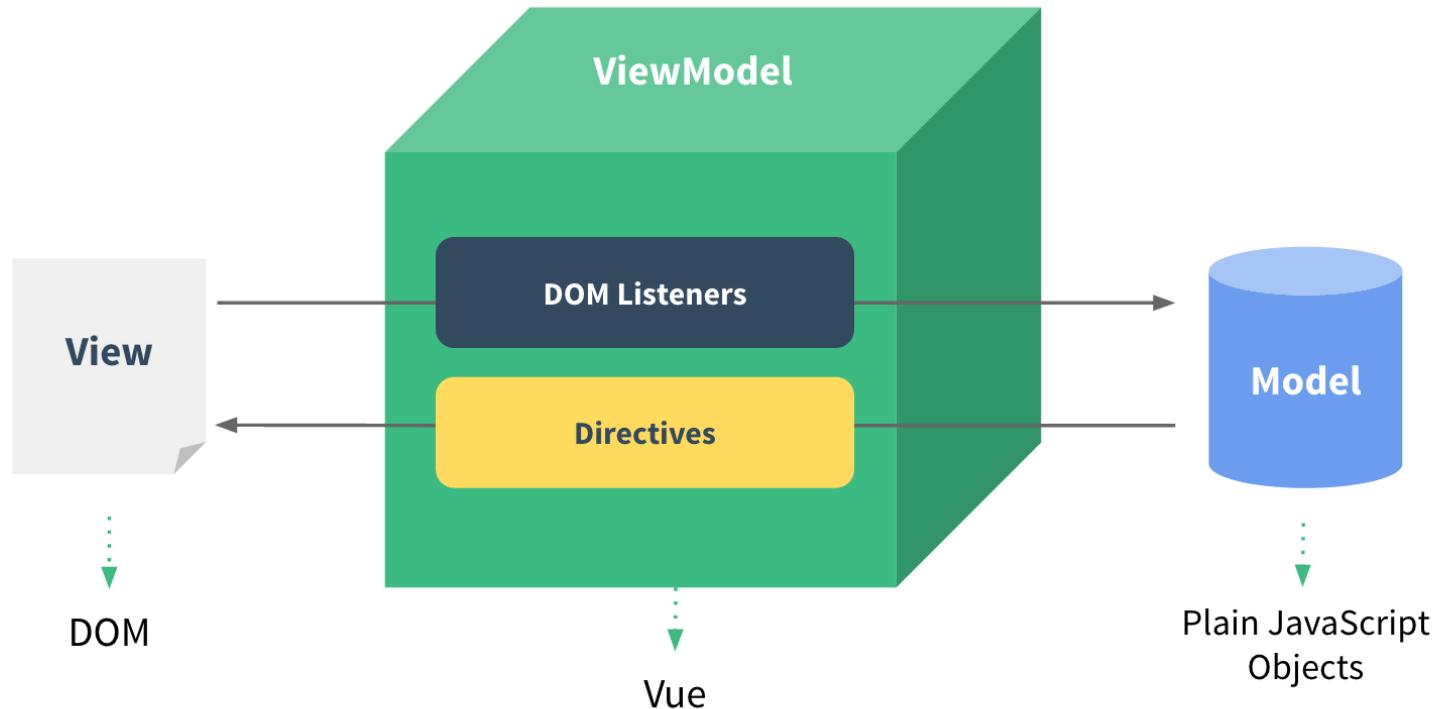
삼성 청년 SW 아카데미

Concepts of Vue.js

MVVM Pattern

- 애플리케이션 로직을 UI로부터 분리하기 위해 설계된 디자인 패턴
- 구성 요소
 1. Model
 2. View
 3. View Model

MVVM in Vue.js



HTML

DOM과 Data의 중개자

{ key: value }

Model

- “Vue에서 Model은 JavaScript Object다.”
- JavaScript의 Object 자료 구조
- 이 Object는 Vue Instance 내부에서 data로 사용되는데 이 값이 바뀌면 View(DOM)가 반응

View

- “Vue에서 View는 DOM(HTML)이다.”
- Data의 변화에 따라서 바뀌는 대상

ViewModel

- “Vue에서 ViewModel은 모든 Vue Instance이다.”
- View와 Model 사이에서 Data와 DOM에 관련된 모든 일을 처리
- ViewModel을 활용해 Data를 얼마만큼 잘 처리해서 보여줄 것인지(DOM)를 고민하는 것

이어서..

삼성 청년 SW 아카데미

Quick Start of Vue.js

Django & Vue.js 코드 작성 순서

- Django
 - url → views → template
 - 데이터의 흐름
- Vue.js
 - “Data가 변화하면 DOM이 변경된다.”
 1. Data 로직 작성
 2. DOM 작성

■ 공식문서 “시작하기” 따라하기

<https://kr.vuejs.org/v2/guide/index.html>

이어서..

삼성 청년 SW 아카데미

Basic syntax of Vue.js

Vue Instance

Vue instance

- 모든 Vue 앱은 Vue 함수로 새 인스턴스를 만드는 것부터 시작
- Vue 인스턴스를 생성할 때는 Options 객체를 전달해야 함
 - 여러 Options들을 사용하여 원하는 동작을 구현
- Vue Instance === Vue Component

```
const app = new Vue({  
})
```

Options/DOM - 'el'

- Vue 인스턴스에 연결(마운트) 할 기존 DOM 엘리먼트가 필요
- CSS 선택자 문자열 혹은 HTMLElement로 작성
- new 를 이용한 인스턴스 생성때만 사용

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

Options/Data - 'data'

- Vue 인스턴스의 데이터 객체
- Vue 앱의 상태 데이터를 정의하는 곳
- Vue template에서 interpolation을 통해 접근 가능
- v-bind, v-on과 같은 �렉티브에서도 사용 가능
- Vue 객체 내 다른 함수에서 this 키워드를 통해 접근 가능
- 주의
 - 화살표 함수를 `data`에서 사용하면 안됨
 - 화살표 함수가 부모 컨텍스트를 바인딩하기 때문에, `this`는 예상과 달리 Vue 인스턴스를 가리키지 않음

```
const app = new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello',  
  }  
})
```

Options/Data - 'methods'

- Vue 인스턴스에 추가할 메서드
- Vue template에서 interpolation을 통해 접근 가능
- v-on과 같은 �렉티브에서도 사용 가능
- Vue 객체 내 다른 함수에서 this 키워드를 통해 접근 가능
- 주의
 - 화살표 함수를 메서드를 정의하는데 사용하면 안됨
 - 화살표 함수가 부모 컨텍스트를 바인딩하기 때문에, `this`는 Vue 인스턴스가 아니며 `this.a`는 정의되지 않음

```
const app = new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello',  
  },  
  methods: {  
    greeting: function () {  
      console.log('hello')  
    }  
  }  
)
```

this keyword in vue.js

- Vue 함수 객체 내에서 vue 인스턴스를 가리킴
- 단, JavaScript 함수에서의 this 키워드는 다른 언어와 조금 다르게 동작하는 경우가 있으니 제공되는 별도 handout 참고
- 화살표 함수를 사용하면 안되는 경우

```
<div id="app">
  <button @click="myFunc">a</button>
  <button @click="yourFunc">b</button>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      a: 1,
    },
    methods: {
      myFunc: function () {
        console.log(this) // Vue instance
      },
      yourFunc: () => {
        console.log(this) // window
      }
    }
  })
</script>
```

1. data

2. method 정의

Template Syntax

Template Syntax

- 렌더링 된 DOM을 기본 Vue 인스턴스의 데이터에 선언적으로 바인딩 할 수 있는
HTML 기반 템플릿 구문을 사용

1. Interpolation
2. Directive

Interpolation (보간법)

1. Text

- 메시지: {{ msg }}

2. Raw HTML

-

3. Attributes

- <div v-bind:id="dynamicId"></div>

4. JS 표현식

- {{ number + 1 }}
- {{ message.split('').reverse().join('') }}

Directive (디렉티브)

- v- 접두사가 있는 특수 속성
- 속성 값은 단일 JS 표현식이 됨 (v-for는 예외)
- 표현식의 값이 변경될 때 반응적으로 DOM에 적용하는 역할을 함

- 전달인자 (Arguments)

- `:` (콜론)을 통해 전달인자를 받을 수도 있음

```
<a v-bind:href="url"> ... </a>
<a v-on:click="doSomething"> ... </a>
```

- 수식어 (Modifiers)

- `.` (점)으로 표시되는 특수 접미사

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

- 디렉티브를 특별한 방법으로 바인딩 해야 함을 나타냄

v-text

- 엘리먼트의 `textContent`를 업데이트
- 내부적으로 interpolation 문법이
`v-text`로 컴파일 됨

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

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello',
    }
  })
</script>
```

v-html

- 엘리먼트의 innerHTML을 업데이트
 - XSS 공격에 취약할 수 있음
- 임의로 사용자로부터 입력 받은 내용은

v-html에 ‘절대’ 사용 금지

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>Hello</div>
    <div v-html="myHtml"></div>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      myHtml: '<b>Hello</b>',
    }
  })
</script>
</body>
</html>
```

v-show

- 조건부 렌더링 중 하나
- 엘리먼트는 항상 렌더링 되고 DOM에 남아있음
- 단순히 엘리먼트에 display CSS 속성을 토글

```
<body>
  <div id="app">
    <p v-show="isTrue">
      true
    </p>
    <p v-show="isFalse">
      false
    </p>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      isTrue: true,
      isFalse: false,
    }
  })
</script>
</body>
```

v-if, v-else-if, v-else

- 조건부 렌더링 중 하나
- 조건에 따라 블록을 렌더링
- 디렉티브의 표현식이 true 일때만 렌더링
- 엘리먼트 및 포함된 디렉티브는 토글하는 동안 삭제되고 다시 작성됨

```
<body>
  <div id="app">
    <!-- 1 -->
    <div v-if="seen">
      seen이 true일때만 렌더링.
    </div>

    <!-- 2 -->
    <div v-if="myType === 'A'">
      A
    </div>
    <div v-else-if="myType === 'B'">
      B
    </div>
    <div v-else-if="myType === 'C'">
      C
    </div>
    <div v-else>
      Not A/B/C
    </div>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      seen: false,
      myType: 'A',
    }
  })
</script>
</body>
```

v-show 와 v-if

- **v-show** (Expensive initial load, cheap toggle)
 - CSS display 속성을 hidden으로 만들어 toggle
 - 실제로 렌더링은 되지만 눈에서 보이지 않는 것이기 때문에 딱 1번만 렌더링이 되는 경우라면 v-if에 비해 상대적으로 렌더링 비용이 높음
 - 하지만 자주 변경되는 요소라면 한번 렌더링 된 이후부터는 보여주는 지에 대한 여부만 판단하면 되기 때문에 토큰 비용이 적음
- **v-if** (Cheap initial load, expensive toggle)
 - 전달인자가 false인 경우 렌더링 되지 않음
 - 화면에서 보이지 않을 뿐만 아니라 렌더링 자체가 되지 않기 때문에 렌더링 비용이 낮음
 - 하지만 자주 변경되는 요소에 경우 다시 렌더링 해야 하기 때문에 비용이 증가할 수 있음

Basic syntax of Vue.js

v-for

- 원본 데이터를 기반으로 엘리먼트 또는 템플릿 블록을 여러 번 렌더링
- item in items 구문 사용
- item 위치의 변수를 각 요소에서 사용할 수 있음
 - 객체의 경우는 key
- v-for 사용 시 반드시 key 속성을 각 요소에 작성
- v-if와 함께 사용하는 경우 v-for는 v-if보다 우선순위가 높음
 - 단, 가능하면 v-if와 v-for를 동시에 사용하지 말 것

```
<body>
  <div id="app">
    <div v-for="fruit in fruits">
      {{ fruit }}
    </div>

    <div v-for="(fruit, index) in fruits" :key="`fruit-${index}`">
      {{ fruit }}
    </div>

    <div v-for="todo in todos" :key="todo.id">
      {{ todo.title }} : {{ todo.completed }}
    </div>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      fruits: ['apple', 'banana', 'coconut'],
      todos: [
        { id: 1, title: 'todo1', completed: true },
        { id: 2, title: 'todo2', completed: false },
        { id: 3, title: 'todo3', completed: true },
      ],
    }
  })
</script>
</body>
```

Basic syntax of Vue.js

v-on

- 엘리먼트에 이벤트 리스너를 연결
- 이벤트 유형은 전달인자로 표시
- 특정 이벤트가 발생했을 때, 주어진 코드가 실행 됨
- 약어 (Shorthand)
 - @
 - v-on:click → @click

```
<div id="app">
  <!-- 메서드 핸들러 -->
  <button v-on:click="doThis">Button</button>
  <button @click="doThis">Button</button>

  <hr>

  <!-- 기본 동작 방지 -->
  <form action="/articles/" @submit.prevent>
    <button>Submit!</button>
  </form>

  <hr>

  <!-- 키 별칭을 이용한 키 입력 수식어 -->
  <input type="text" @keyup.enter="onInputEnter">
  <input type="text" @keypress.enter="onInputEnter('ssafy')">

  <hr>

  {{ message }}
  <button @click="changeMessage">My Change Button</button>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: 'Hello Vue.js',
    },
    methods: {
      doThis: function () {
        alert('hello')
      },
      changeMessage: function () {
        this.message = 'New Hello Vue.js'
      },
      onInputEnter: function (inputValue) {
        console.log('Enter Enter!')
        console.log(inputValue)
      }
    }
  })
</script>
```

Basic syntax of Vue.js

v-bind

- HTML 요소의 속성에

Vue의 상태 데이터를 값으로 할당

- Object 형태로 사용하면 value가 true인 key가 class 바인딩 값으로 할당

- 약어 (Shorthand)

- :(콜론)

- v-bind:href → :href

```
<div id="app">
  <!-- 속성 바인딩 -->
  
  

  <hr>

  <!-- 클래스 바인딩 -->
  <div :class="{ active: isRed }">클래스 바인딩</div>

  <h2 :class="[activeRed, myBackground]">
    Hello Vue.js
  </h2>

  <hr>

  <!-- 스타일 바인딩 -->
  <ul>
    <li v-for="todo in todos" :class="{ active: todo.isActive }" :style="{ fontSize: fontSize + 'px' }">
      {{ todo }}
    </li>
  </ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      imageSrc: 'https://picsum.photos/200/300/',
      isRed: true,
      activeRed: 'active',
      myBackground: 'my-background-color',
      todos: [
        { id: 1, title: 'todo 1', isActive: true },
        { id: 2, title: 'todo 2', isActive: false },
      ],
      fontSize: 30,
    }
  })
</script>
```

Basic syntax of Vue.js

v-model

- HTML form 요소의 값과 data를 양방향 바인딩
- 수식어
 - .lazy
 - input 대신 change 이벤트 이후에 동기화
 - .number
 - 문자열을 숫자로 변경
 - .trim
 - 입력에 대한 trim을 진행

```
<body>
  <div id="app">
    <h2>1. Input -> Data</h2>
    <h3>{{ myMessage }}</h3>
    <input @input="onInputChange" type="text">
    <hr>

    <h2>2. Input <-> Data</h2>
    <h3>{{ myMessage2 }}</h3>
    <input v-model="myMessage2" type="text">
    <hr>

    <h2>3. Checkbox</h2>
    <input type="checkbox" id="checkbox" v-model="checked">
    <label for="checkbox">{{ checked }}</label>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        myMessage: '',
        myMessage2: '',
        checked: true,
      },
      methods: {
        onInputChange: function (event) {
          this.myMessage = event.target.value
        },
      }
    })
  </script>
</body>
```

Options/Data - 'computed'

- 데이터를 기반으로 하는 계산된 속성
- 함수의 형태로 정의하지만 함수가 아닌
함수의 반환 값이 바인딩 됨
- 종속된 대상을 따라 저장(캐싱) 됨
- 종속된 대상이 변경될 때만 함수를 실행
- 즉, Date.now() 처럼 아무 곳에도 의존하지 않는
computed 속성의 경우 절대로 업데이트되지 않음
- 반드시 반환 값이 있어야 함

```
<body>
  <div id="app">
    <p>{{ num }}</p>
    <p>{{ doubleNum }}</p>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        num: 2,
      },
      computed: {
        doubleNum: function () {
          return this.num * 2
        }
      },
    })
  </script>
</body>
```

computed & method

- computed 속성 대신 methods에 함수를 정의할 수도 있음
 - 최종 결과에 대해 두 가지 접근 방식은 서로 동일
- 차이점은 computed 속성은 종속 대상을 따라 저장(캐싱) 됨
- 즉, computed는 종속된 대상이 변경되지 않는 한 computed에 작성된 함수를 여러 번 호출해도 계산을 다시 하지 않고 계산되어 있던 결과를 반환
- 이에 비해 methods를 호출하면 렌더링을 다시 할 때마다 항상 함수를 실행

Options/Data - 'watch'

- 데이터를 감시
- 데이터에 변화가 일어났을 때 실행되는 함수

```
<body>
  <div id="app">
    <p>{{ num }}</p>
    <button @click="num += 1">add 1</button>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      num: 2,
    },
    watch: {
      num: function () {
        console.log(` ${this.num} 이 변경되었습니다.`)
      }
    }
  })
</script>
</body>
```

computed & watch

- computed
 - 특정 데이터를 직접적으로 사용/가공하여 다른 값으로 만들 때 사용
 - 속성은 계산해야 하는 목표 데이터를 정의하는 방식으로
소프트웨어 공학에서 이야기하는 ‘선언형 프로그래밍’ 방식
- watch
 - 특정 데이터의 변화 상황에 맞춰 다른 data 등이 바뀌어야 할 때 주로 사용
 - 감시할 데이터를 지정하고 그 데이터가 바뀌면 이런 함수를 실행하라는 방식으로
소프트웨어 공학에서 이야기하는 ‘명령형 프로그래밍’ 방식

computed & watch

- computed
 - “특정 값이 변동하면 특정 값을 새로 계산해서 보여준다.”
- watch는
 - “특정 값이 변동하면 다른 작업을 한다.”
 - 특정 대상이 변경되었을 때 콜백 함수를 실행 시키기 위한 트리거
- computed가 코드 반복이 적은 등 우수하다고 평가하는 경향이 있음

선언형 & 명령형

- 선언형 프로그래밍
 - 계산해야 하는 목표 데이터를 정의 (computed)
- 명령형 프로그래밍
 - 데이터가 바뀌면 특정 함수를 실행해! (watch)

Options/Assets - 'filters'

- 텍스트 형식화를 적용할 수 있는 필터
- interpolation 혹은 v-bind를 이용할 때 사용 가능
- 필터는 자바스크립트 표현식 마지막에 “|” (파이프)와 함께 추가되어야 함
- 체이닝 가능

```
<body>
  <div id="app">
    <p>{{ numbers | getOddNums | getUnderTenNums }}</p>
  </div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
    },
    filters: {
      getOddNums: function (nums) {
        const oddNums = nums.filter(function (num) {
          return num % 2
        })
        return oddNums
      },
      getUnderTenNums: function (nums) {
        const underTen = nums.filter(function (num) {
          return num < 10
        })
        return underTen
      }
    }
  })
</script>
</body>
```

Lifecycle Hooks

- 각 Vue 인스턴스는 생성될 때 일련의 초기화 단계를 거침
 - 예를 들어 데이터 관찰 설정이 필요한 경우, 인스턴스를 DOM에 마운트하는 경우, 그리고 데이터가 변경되어 DOM를 업데이트하는 경우 등
- 그 과정에서 사용자 정의 로직을 실행할 수 있는 **라이프사이클 퓨** 도 호출됨
- [공식문서](#)를 통해 각 라이프사이클 퓨의 상세 동작을 참고

Lifecycle Hooks 예시

- created 혹은 인스턴스가 생성된 후에 호출됨

```
<script>
  new Vue({
    data: {
      a: 1
    },
    created: function () {
      console.log('a is: ' + this.a) // => 'a is: 1'
    }
  })
</script>
```

Lifecycle Hooks 예시

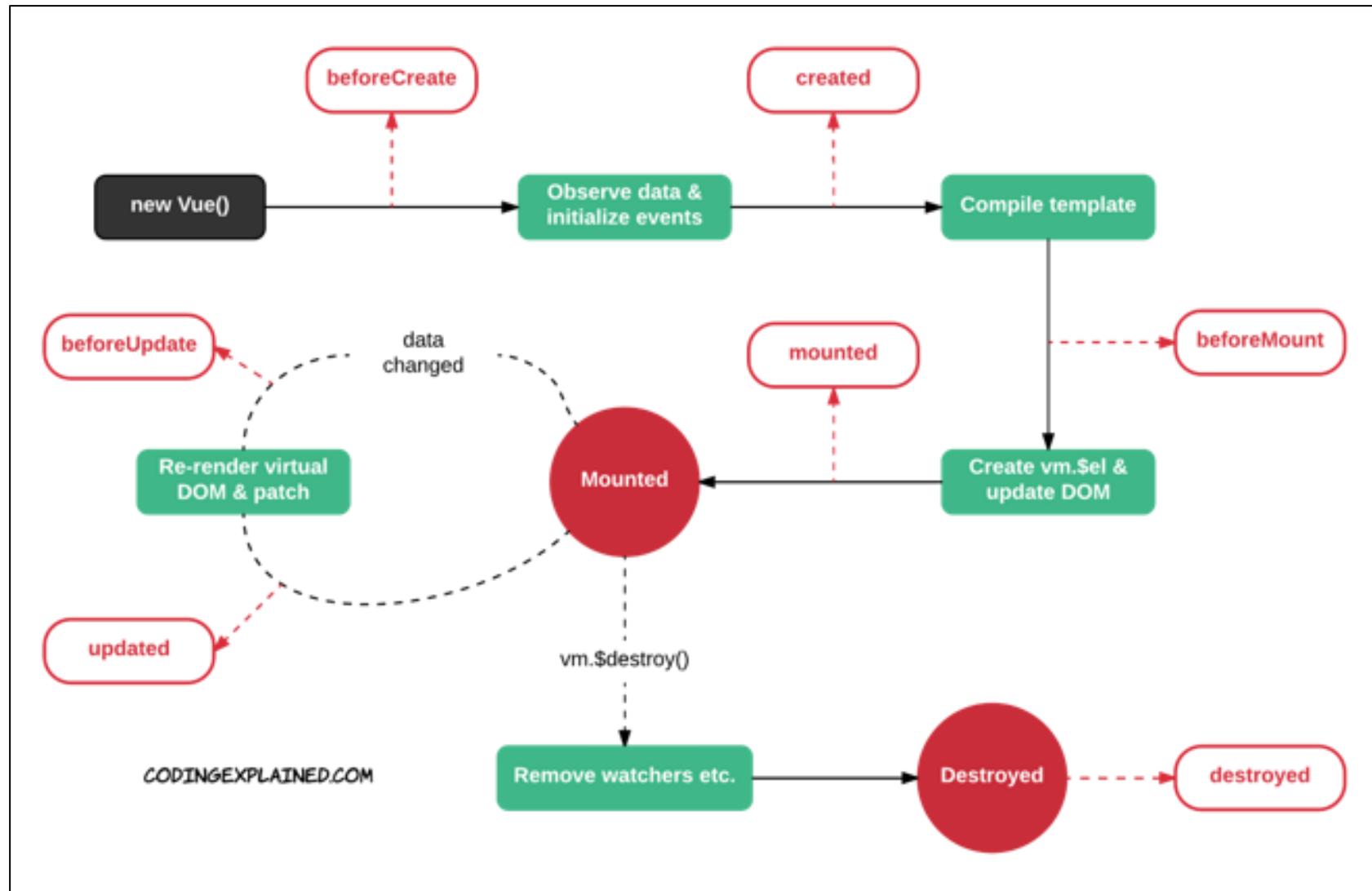
- `created`를 통해 애플리케이션의 초기 데이터를 API 요청을 통해 불러올 수 있음

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

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  const API_URL = 'https://dog.ceo/api/breeds/image/random'
  const app = new Vue({
    el: '#app',
    data: {
      imgSrc: '',
    },
    methods: {
      getImg: function () {
        axios.get(API_URL)
          .then(response => {
            this.imgSrc = response.data.message
          })
    },
    // Vue 인스턴스가 생성되면
    // getImg() 함수를 호출해 처음부터 이미지 데이터를 불러옴
    created: function () {
      this.getImg()
    }
  })
</script>
</body>
```

Basic syntax of Vue.js

Lifecycle Hooks 예시



lodash

- 모듈성, 성능 및 추가 기능을 제공하는 JavaScript 유틸리티 라이브러리
 - array, object등 자료구조를 다룰 때 사용하는 유요하고 간편한 유틸리티 함수들을 제공
 - reverse, sortBy, range, random ...

이어서..

삼성 청년 SW 아카데미

마무리

오늘은

- Things of Vue.js
 - Front-End Framework
 - SPA (Single Page Application)
 - MVVM (Model, View, ViewModel)
- Basic syntax of Vue.js
 - Vue instance
 - Template Syntax
 - Life cycle hooks

사전 준비사항

- 아래 공용 노션 문서에서 “Youtube API key 발급 받기” 설정

<http://bit.ly/05-document>