



# The Majesty of Vue.js 2

Alex  
Kyriakidis

Kostas  
Maniatis

# The Majesty of Vue.js 2 (Korean)

The Majesty of Vue.js 2의 한국어판입니다.

Alex Kyriakidis, Kostas Maniatis, ChangJoo Park(박창주)

This book is for sale at <http://leanpub.com/vuejs2-korean>

This version was published on 2017-06-26



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 – 2017 Alex Kyriakidis, Kostas Maniatis, ChangJoo Park(박창주)

# 차례

소개 . . . . .	i
Vue.js 소개 . . . . .	ii
Vue.js 개요 . . . . .	ii
Vue.js를 사용하는 사람들의 의견 . . . . .	ii
다른 프레임워크와 비교 . . . . .	iv
환영합니다 . . . . .	xii
The Majesty of Vue 2에 관하여 . . . . .	xii
이 책은 어떤 사람이 읽어야 하나요? . . . . .	xii
저자와 연락하려면 . . . . .	xii
혼자 해보기 . . . . .	xiii
예제 코드 . . . . .	xiii
오판자 . . . . .	xiii
표기 규칙 . . . . .	xiii
Vue.js 기초 . . . . .	1
Vue.js 설치 . . . . .	2
독립 실행 버전 . . . . .	2
NPM 사용 . . . . .	2
Bower 사용하기 . . . . .	3
시작하기 . . . . .	4
안녕하십니까 전하! . . . . .	4

## 차례

양방향 바인딩 . . . . .	6
jQuery와 비교 . . . . .	7
혼자 해보기 . . . . .	9
디렉티브 . . . . .	10
v-show . . . . .	10
v-if . . . . .	13
v-else . . . . .	15
v-if와 v-show의 차이 . . . . .	17
혼자 해보기 . . . . .	18
리스트 렌더링 . . . . .	19
부트스트랩 설치와 사용 . . . . .	19
v-for . . . . .	21
배열 렌더링 . . . . .	23
객체 v-for . . . . .	28
혼자 해보기 . . . . .	30
상호작용 . . . . .	31
이벤트 핸들링 . . . . .	31
이벤트 수식어 . . . . .	35
키 수식어 . . . . .	38
계산된 속성 . . . . .	39
혼자 해보기 . . . . .	45
필터 . . . . .	47
필터링된 결과 . . . . .	47
결과 정렬 . . . . .	56
사용자 정의 필터 . . . . .	60
유ти리티 라이브러리 . . . . .	61
혼자 해보기 . . . . .	65
컴포넌트 . . . . .	66

## 차례

컴포넌트가 무엇인가요? . . . . .	66
컴포넌트 사용하기 . . . . .	66
템플릿 . . . . .	68
속성 . . . . .	69
재사용성 . . . . .	72
함께 사용하기 . . . . .	75
혼자 해보기 . . . . .	82
사용자 정의 이벤트 . . . . .	83
발생과 청취 . . . . .	83
부모-자식 간 통신 . . . . .	86
이벤트에서 전달인자 사용 . . . . .	88
비 부모 자식간 통신 . . . . .	93
이벤트 리스너 제거 . . . . .	96
이야기로 돌아가서 . . . . .	97
혼자 해보기 . . . . .	100
클래스와 스타일 바인딩 . . . . .	102
클래스 바인딩 . . . . .	102
스타일 바인딩 . . . . .	106
이벤트를 이용한 바인딩 . . . . .	109
혼자 해보기 . . . . .	111
API 사용하기 . . . . .	112
머리말 . . . . .	113
CRUD . . . . .	113
API . . . . .	113
실제 데이터를 사용하여 작업하기 . . . . .	118
비동기로 데이터 가져오기 . . . . .	118
리팩토링 . . . . .	121

## 차례

데이터 수정 . . . . .	124
데이터 제거 . . . . .	126
HTTP 클라이언트 . . . . .	129
소개 . . . . .	129
Vue-resource . . . . .	129
axios . . . . .	130
axios 사용하기 . . . . .	131
기능 향상하기 . . . . .	132
자바스크립트 파일 . . . . .	143
소스코드 . . . . .	144
혼자 해보기 . . . . .	149
페이지네이션 . . . . .	152
구현 . . . . .	153
페이지네이션 링크 . . . . .	156
혼자 해보기 . . . . .	159
<b>대규모 애플리케이션 구축 . . . . .</b>	<b>160</b>
ECMAScript 6 . . . . .	161
소개 . . . . .	161
변수 선언 . . . . .	162
화살표 함수 . . . . .	163
모듈 . . . . .	164
클래스 . . . . .	165
전달인자 기본값 . . . . .	166
템플릿 리터럴 . . . . .	167
고급 워크플로우 . . . . .	169
Babel을 이용한 ES6 컴파일 . . . . .	169
Gulp를 이용한 워크플로우 자동화 . . . . .	178

## 차례

Webpack을 이용한 모듈 번들링 . . . . .	182
요약 . . . . .	190
단일 파일 컴포넌트 . . . . .	191
vue-cli . . . . .	191
Webpack 템플릿 . . . . .	195
.vue 파일 만들기 . . . . .	203
중복 상태 제거 . . . . .	216
속성을 이용한 공유 . . . . .	216
전역 저장소 . . . . .	221
컴포넌트 교체 . . . . .	225
동적 컴포넌트 . . . . .	225
Vue 라우터 . . . . .	232
설치하기 . . . . .	232
사용방법 . . . . .	233
이름을 가지는 라우트 . . . . .	235
히스토리 모드 . . . . .	236
중첩 라우트 . . . . .	238
자동 active 클래스 추가 . . . . .	240
라우트 객체 . . . . .	243
동적 세그먼트 . . . . .	244
라우트 별칭 . . . . .	251
라우트 푸시 . . . . .	253
트랜지션 . . . . .	254
네비게이션 가드 . . . . .	258
혼자해보기 . . . . .	260
마무리하며 . . . . .	263
더 배우려면 . . . . .	264

## 차례

튜토리얼 . . . . .	264
비디오 . . . . .	264
책 . . . . .	265
오픈소스 프로젝트 . . . . .	265
Awesome Vue . . . . .	266

# 소개

# Vue.js 소개

## Vue.js 개요

Vue(뷰/vju : /라고 발음합니다. view와 같음)는 사용자 인터페이스를 개발하기 위한 프로그레시브 프레임워크입니다. 다른 프레임워크와 다르게 Vue는 점진적으로 채택할 수 있게 설계하였습니다. 코어 라이브러리는 보이는 부분에만 초점을 맞추어 다른 라이브러리나 기존 프로젝트와 통합하기 매우 쉽습니다. Vue는 현대적인 도구 및 지원하는 라이브러리<sup>1</sup>와 함께 사용하면 정교한 싱글 페이지 애플리케이션을 완벽하게 만들 수 있습니다.

글을 읽는 여러분이 숙련된 프론트엔드 개발자인 경우 Vue.js를 다른 라이브러리/프레임워크 와의 비교한 [공식 가이드의 다른 프레임워크와의 비교](#)<sup>2</sup>를 확인하세요.

Vue.js의 핵심적인 자세한 내용은 [Vue.js 공식 가이드](#)<sup>3</sup>를 참조하세요.

## Vue.js를 사용하는 사람들의 의견

“Vue.js는 자바스크립트를 사랑하게 만들었습니다. 사용법이 매우 쉽고 즐겁습니다. 기본 서비스를 확장할 수 있는 플러그인 및 도구를 포함하는 훌륭한 생태계를 갖추고 있습니다. 작거나 큰 모든 프로젝트에 빠르게 추가할 수 있으며 이를 위해 코드 몇 줄만 작성하면 시작할 수 있습니다. Vue.js는 빠르고 가벼운 프론트엔드 개발의 미래입니다!”

—Alex Kyriakidis

---

“자바스크립트를 시작했을 때 기쁘게도 많은 가능성을 보았습니다. 친구가 Vue.js를 배우라고 제안했을 때 나는 친구의 조언을 따랐습니다. 튜토리얼을 읽는 동안 이전에 Vue에 시간을 투자했다면 지금까지했던 모든 것을 더 잘 할 수 있지 않았을까 생각했습니다. 개인적인 생각이지만 업무를 빠르게 하고 싶다면 Vue.js는 정말 필요했던 멋지고 쉬운 자바스크립트 프레임워크입니다.”

—Kostas Maniatis

---

<sup>1</sup><https://github.com/vuejs/awesome-vue#libraries--plugins>

<sup>2</sup><https://kr.vuejs.org/v2/guide/comparison.html>

<sup>3</sup><http://vuejs.org/guide/overview.html>

“내 말을 기억하세요. Vue.js는 2016년에 하늘에 쏘아올린 로켓과 같은 인기를 얻을 것 입니다. 정말 좋습니다.”

— Jeffrey Way

---

“Vue는 자바스크립트 프레임워크 중 항상 찾았던 것입니다. 이것은 당신과 함께 성장할 프레임워크입니다. 한 페이지에 화면을 보여주거나 Vuex 및 Vue Router를 사용하여 고급 싱글 페이지 애플리케이션을 만들 수 있습니다. 내가 본 것 중에 가장 세련된 자바스크립트 프레임워크입니다.”

— Taylor Otwell

---

“Vue.js는 SPA와 마찬가지로 서버 측 렌더링 된 애플리케이션에서 사용하는 것이 자연스럽다고 느낀 첫 번째 프레임워크입니다. 싱글 페이지 안에 작은 위젯이 필요한지 아니면 복잡한 클라이언트를 만들 것인지 관계없이 과하거나 부족한 느낌이 전혀 없었습니다.”

— Adam Wathan

---

“Vue.js는 사용하기 쉽고 이해하기 쉬운 프레임워크입니다. 다른 사람들이 누가 더 복잡하게 만들 수 있는지 경쟁하는 상황에서 신선한 공기 같습니다.”

— Eric Barnes

---

“내가 Vue.js를 좋아하는 이유는 내가 하이브리드 디자이너/개발자 이기 때문입니다. 나는 React, Angular 및 다른 몇 가지 프레임워크를 살펴 보았지만 학습 곡선과 용어의 어려움 때문에 사용하기 어려웠습니다. Vue.js는 내가 이해한 첫 번째 프레임워크입니다. 또한 나와 같은 자바스크립트 사용자들에게 쉬울 뿐 아니라 Angular 및 React에 대한 경험이 풍부한 개발자도 주목하고 Vue.js를 좋아하고 있는 것을 알고 있습니다.”

— Jack Barham

---

## 다른 프레임워크와 비교

### Angular 1

일부 Vue의 문법은 Angular와 매우 유사합니다. (예를 들어, `v-if` 와 `ng-if`). Angular가 제대로 갖춘 많은 것들을 가지고 있었기 때문에 초기 Vue에 영감을 주었습니다. Angular를 따르는데 많은 어려운 부분들이 있었지만 Vue는 이를 개선하려고 매우 노력했습니다.

#### 복잡성

Vue는 API와 디자인 측면에서 Angular 1보다 훨씬 간단합니다. 간단한 애플리케이션을 작성하기에 필요한 학습기간은 일반적으로 하루면 충분하지만 Angular 1은 그렇지 않습니다.

#### 유연성과 모듈성

Angular 1은 애플리케이션 구성에 대한 강제성이 있지만 Vue는 모듈 방식의 더욱 유연한 해결 방법을 가지고 있습니다. 이것이 [Webpack 템플릿<sup>4</sup>](#)을 제공하는 이유입니다. 이를 통해 핫 모듈 리로딩, 린트 (linting), CSS 추출과 같은 고급 기능에 대한 액세스 권한을 부여하는 동시에 빠르게 설정할 수 있습니다.

#### 데이터 바인딩

Angular 1은 영역간 양방향 바인딩을 사용하는 반면 Vue는 컴포넌트 간 단방향 데이터 흐름을 가집니다. 이로 인해 데이터의 흐름이 단순한 애플리케이션에서 데이터 흐름을 쉽게 파악할 수 있습니다.

#### 디렉티브 vs 컴포넌트

Vue는 디렉티브와 컴포넌트를 명확히 구분합니다. 디렉티브는 DOM조작을 캡슐화하기 위한 것입니다. 그리고 컴포넌트는 자체 뷰와 데이터 로직을 포함하는 하나의 단위입니다. Angular 1에서는 이 둘 사이에 많은 혼란을 보입니다.

#### 성능

Vue는 더 나은 성능을 가지며 변경 감시를 사용하지 않기 때문에 훨씬 쉽게 최적화할 수 있습니다. 감시자가 많으면 Angular 1은 느려집니다. 범위가 변경될 때마다 이러한 모든 감시자를 다시 평가해야 하기 때문입니다. 또한, 일부 감시자가 다른 업데이트를 트리거하는 경우 디제스트 주기를 여러 번 실행하여 “안정화”해야 할 수도 있습니다. Angular 사용자는 디제스트 주기를 벗어나기 위해 종종 숨겨진 기술에 의지해야 하며 때에 따라 많은 감시자와 함께 범위를 최적화할 수 있는 방법이 없습니다.

---

<sup>4</sup><https://github.com/vuejs-templates/webpack>

Vue는 비동기 대기열이 있는 투명한 의존성 추적/관찰 시스템을 사용하기 때문에 이 문제가 전혀 발생하지 않습니다. 모든 변경 사항은 명시적 종속 관계가 없는 한 독립적으로 트리거됩니다.

흥미롭게도 Angular 2와 Vue가 Angular 1 문제를 해결하는 방법에는 몇 가지 유사점이 있습니다.

## Angular 2

Angular 2는 완전히 새로운 프레임워크이기 때문에 별도의 섹션을 만들었습니다. 예를 들어, 1급 컴포넌트 시스템을 가집니다. 그리고 많은 구현 세부 사항을 완전히 다시 작성했으며 API 또한 매우 많이 변경되었습니다.

### 규모와 성능

성능면에서 볼 때 두 프레임워크 모두 매우 빠르며 실제 사용 사례의 데이터가 충분하지는 않아 결론을 내릴 수는 없습니다. 그러나 숫자로 확인하려는 경우 Vue 2.0은 이 [써드파티 벤치 마크](#)<sup>5</sup>에 따르면 Angular 2보다 앞서있는 것으로 보입니다.

규모의 측면에서 Angular 2는 오프라인 컴파일 및 트리세이킹으로 인해 상당히 작아질 수 있지만, 컴파일러가 포함된 완전한 기능을 하는 Vue 2.0이 더 작습니다. (Vue 2.0은 23kb, Angular 2 50kb). 사용하지 않는 기능에 대한 코드를 제거하는 트리세이킹으로 인해 Angular 2 앱의 크기가 작다는 점에 유의하십시오. 프레임워크에서 더 많은 기능을 가져와 사용할 때 결국 실제 크기로 다시 확장됩니다.

### 유연성

Vue는 Angular 2보다 훨씬 덜 강제적이며, 다양한 빌드 시스템에 대한 공식적인 지원을 제공하며 애플리케이션 구조를 제한하지 않습니다. 많은 개발자가 이러한 자유를 누리고 싶어 하지만 일부 개발자는 모든 애플리케이션을 빌드하는 올바른 하나의 방법만 있는 것을 선호합니다.

### 학습 곡선

Vue를 시작하려면 HTML 및 ES5 자바스크립트(즉, 일반 자바 스크립트)에 익숙해야합니다. 기본 기술을 사용하여 하루만에 [가이드](#)<sup>6</sup>를 읽고 작은 애플리케이션을 만들 수 있습니다.

Angular 2의 학습 곡선은 TypeScript가 없는 환경이라도 훨씬 가파릅니다. [빠른 시작 가이드](#)<sup>7</sup>는 Hello World를 설명하기 위해 ES2015 자바스크립트, 18개의 NPM 의존성, 4개의 파일 및 3,000 단어를 사용하는 앱으로 시작합니다. [Vue의 Hello World](#)<sup>8</sup>가 조금 더 간단하다 해도 과언이 아닙니다. 아마 이를 만들기 위해 가이드의 전체 페이지를 소모할 필요조차 없습니다.

<sup>5</sup><http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html>

<sup>6</sup><https://vuejs.org/v2/guide/>

<sup>7</sup><https://angular.io/docs/js/latest/quickstart.html>

<sup>8</sup><https://jsfiddle.net/chrisvfritz/50wL7mdz/>

## React

React와 Vue는 많은 공통점을 공유합니다.

- 가상 DOM을 활용합니다.
- 반응적이고 조합 가능한 컴포넌트를 제공합니다.
- 코어 라이브러리에만 집중하며 라우팅 및 전역 상태를 관리하는 컴패니언 라이브러리가 있습니다.

### 성능 분석

지금까지 테스트한 모든 실제 시나리오에서 Vue는 React보다 훨씬 우수합니다.

#### 렌더링 성능

UI를 렌더링할 때 일반적으로 DOM 조작이 가장 비용이 많이 드는 작업이며 유감스럽게도 라이브러리에서는 이러한 원시 작업을 더 빠르게 만들 수 없습니다. 할 수 있는 최선의 방법은 다음과 같습니다.

1. 필요한 DOM 조작 수를 최소화합니다. React와 Vue는 모두 가상의 DOM 추상화를 사용하여 이 작업을 하며 두 가지 구현 모두 거의 동일하게 작동합니다.
2. DOM 조작에 가능한 적은 오버헤드(순수 자바스크립트 계산)만 가집니다. 이 것은 Vue와 React의 차이입니다.

React에서 엘리먼트 렌더링에 대한 추가적인 오버헤드를 1이라 정하고 평균적인 컴포넌트의 오버헤드가 2라고 가정합니다. Vue에서 엘리먼트의 오버헤드는 0.1과 비슷하지만 반응 시스템에 필요한 설정 때문에 평균적인 컴포넌트 오버헤드는 4입니다.

즉, 렌더링 되는 컴포넌트보다 더 많은 엘리먼트가 있는 일반적인 애플리케이션에서 Vue는 React보다 높은 성능 향상을 보입니다. 그러나 각 엘리먼트를 렌더링하기 위해 1개의 컴포넌트를 사용하는 매우 극한의 상황에서 Vue는 일반적으로 느립니다.

Vue와 React 모두 상태 또는 인스턴스가 없는 함수형 컴포넌트를 제공하므로 오버헤드가 적습니다. 이러한 성능이 중요한 사용되면 Vue가 더 빠릅니다.

#### 갱신 성능

React는 `shouldComponentUpdate`를 구현해야 할 상황이 있으며, 완전히 최적화된 렌더링을 하기 위해 불변의 데이터 구조를 사용해야 합니다. Vue는 컴포넌트의 의존성이 자동으로 추적되므로 해당 의존성 중 하나가 변경될 때만 갱신합니다. Vue에서 도움이 되는 추가적인 최적화로 긴 리스트의 항목에 키 속성을 추가합니다.

즉, 최적화되지 않은 Vue의 갱신은 최적화 되지 않은 React보다 훨씬 빠르며 심지어 완전히 최적화된 React도 기본으로 제공되는 Vue보다 느립니다.

#### 개발에 관하여

프로덕션 환경에서의 성능은 최종 사용자 경험과 직접 관련되어 있으므로 더 중요한 부분이지만 개발에 대한 경험은 개발자와 관련되어 있으므로 여전히 중요합니다.

Vue와 React 모두 대부분의 일반적인 애플리케이션에서 속도가 빠릅니다. 그러나 높은 프레임 속도의 데이터 시각화 또는 애니메이션을 프로토타이핑 할 때 Vue는 개발 시 초당 10프레임을 처리하는 반면 React는 초당 약 1프레임으로 떨어지는 경우를 보았습니다.

이것은 많은 자세한 경고와 오류 메시지를 보여주는 개발 모드의 무질서한 많은 검사 때문입니다.

## Ember

Ember는 높은 찬사를 받는 완전한 기능의 프레임워크입니다. 그것은 많은 기존의 관행을 제공하며, 일단 익숙해지면 생산성을 높일 수 있습니다. 그러나 이는 학습 곡선이 높고 유연성이 떨어지는 것을 의미합니다. 함께 작동하는 느슨하게 결합 된 도구 집합을 사용하여 독창적인 프레임워크와 라이브러리를 선택하려고 하면 트레이드 오프가 발생합니다. 후자는 더 많은 자유를 제공하지만, 더 많은 구조에 관한 결정을 요구합니다.

이는 Vue 코어와 Ember의 객체 모델 레이어를 비교하면 더 정확히 알 수 있습니다.

- Vue는 일반 자바스크립트 객체 및 완전히 자동으로 계산된 속성에 대해 눈에 거슬리지 않는 반응성을 제공합니다. Ember에서는 Ember 객체의 모든 것을 래핑하고 계산된 속성의 종속성을 수동으로 선언 해야 합니다.
- Vue의 템플릿 문법은 자바스크립트 표현식의 모든 기능을 활용하지만, Handlebars의 표현식 및 헬퍼 문법은 매우 제한적입니다.
- 성능 측면에서 Vue는 Ember 2.0의 최신 Glimmer 엔진 업데이트 이후에도 Ember보다 월등히 뛰어납니다. Vue는 업데이트를 자동으로 일괄 처리하지만, Ember에서는 성능이 중요한 상황에서 실행 루프를 수동으로 관리해야합니다.

## Polymer

Polymer는 Google이 후원하는 또 다른 프로젝트이며 실제로 Vue의 영감의 근원입니다. Vue의 컴포넌트는 Polymer의 사용자 지정 엘리먼트와 느슨하게 비교할 수 있으며 둘 다 매우 유사한 개발 스타일을 제공합니다. 가장 큰 차이점은 Polymer는 최신 웹 컴포넌트 기능을 기반으로 하며 이러한 기능을 기본적으로 지원하지 않는 브라우저에서는 성능을 저하시키는 약간의 폴리필을 요구합니다. 대조적으로, Vue는 IE9에 의존성이나 폴리필이 없이 작동합니다.

Polymer 1.0 개발 팀은 성능을 보완하기 위해 데이터 바인딩 시스템을 매우 제한적으로 만들었습니다. 예를 들어, Polymer 템플릿에서 지원되는 유일한 표현식은 Boolean 부정 및 단일 메소드 호출입니다. 계산된 속성 구현 또한 유연하지 않습니다.

Polymer 사용자 정의 엘리먼트는 HTML 파일로 제작되어 일반 자바스크립트/CSS(및 현재 브라우저에서 지원되는 언어 기능)로 제한됩니다. 이에 비해 Vue의 단일 파일 컴포넌트를 사용하면 ES2015+ 및 원하는 모든 CSS 전처리기를 쉽게 사용할 수 있습니다.

프로덕션 환경으로 배포할 때 Polymer는 브라우저에서 스펙을 구현한다고 가정하는 HTML 가져오기 및 서버와 클라이언트 모두에서 HTTP/2 지원을 사용하여 모든 것을 로드하는 것을 권장합니다. 이는 대상 사용자 및 배포 환경에 따라 가능할 수도 있고 그렇지 않을 수도 있습니다. 이것이 바람직하지 않은 경우에는 Vulcanizer라고 하는 특수 도구를 사용하여 폴리머 엘리먼트를 뭉어야 합니다. 앞에서 볼 때 Vue는 비동기 컴포넌트 기능과 Webpack의 코드 분할 기능을 결합하여 애플리케이션 번들의 일부를 자연 로드 되도록 쉽게 분리 할 수 있습니다. 이를 통해 이전 브라우저와의 호환성을 유지하면서 앱 로드 성능을 높일 수 있습니다.

## Riot

Riot 2.0은 작고 아름답게 디자인된 API를 사용하여 유사한 컴포넌트 기반 개발 모델 (Riot에서 “태그”라고 함)을 제공합니다. Riot과 Vue는 디자인 철학에 많은 부분을 공유합니다. 그러나 Vue는 Riot보다 약간 무겁지만 몇 가지 중요한 이점을 제공합니다.

- 진정한 조건부 렌더링. Riot은 조건부 렌더링에서 단순히 보여주고 숨기는 것만 합니다.
- 훨씬 강력한 라우터. Riot의 라우팅 API는 극히 약합니다.
- 보다 성숙한 도구 지원. Vue는 Webpack 및 Browserify를 공식적으로 지원하지만 Riot는 빌드 시스템 통합을 커뮤니티 지원에 의존합니다.
- 트랜지션 시스템. Riot에는 아무 것도 없습니다.
- 더 나은 성능. 가상 DOM을 사용하는 것보다 Riot은 DOM 트리 순회를 하므로 Angular 1과 동일한 성능 문제가 있습니다.

최신 문서는 [Vue.js 가이드](#)를 참고하세요.

# 환영합니다

## The Majesty of Vue 2에 관하여

이 책은 빠르게 유행하고 있는 자바스크립트 프레임워크인 Vue.js에 대한 안내서입니다!

얼마 전에, Laravel과 Vue.js를 기반으로 새 프로젝트를 시작했습니다. Vue.js 가이드와 몇 가지 예제를 모두 읽은 후 웹에 Vue.js에 대한 자료가 부족함을 알게 되었습니다. 프로젝트를 개발하면서 쌓인 경험을 사람들과 나누고자 책을 쓰게 되었습니다. 최근 Vue.js 2 버전이 출시 되었습니다. 그래서 모든 예제와 관련된 내용을 다시 작성해서 새로운 책을 쓰기로 결정했습니다.

이 책은 격식을 차리는 대신, 직관적이며 쉽게 따라 할 수 있게 쓰였으며 모든 예제는 모든 사람에게 적절한 방향을 제공할 수 있도록 자세하게 설명합니다.

가장 기초적인 내용부터 시작하여 많은 예제와 함께 Vue.js의 중요한 내용을 다룹니다.

이 책이 끝날 때쯤이면 Vue.js를 이용해 빠른 프론트엔드 애플리케이션을 만들거나 기존 프로젝트의 성능을 향상할 수 있을 것입니다.

## 이 책은 어떤 사람이 읽어야 하나요?

현대적인 웹 개발에 대해 배워본 사람은 부트스트랩과 많은 자바스크립트 프레임워크를 사용해 봤을 것입니다. 이 책은 가볍고 간단한 자바스크립트 프레임워크를 배우려는 사람을 위해 쓰여졌습니다. 아주 많은 사전지식이 필요하지는 않으나 기본적으로 HTML과 자바스크립트에 익숙해져야 합니다. 지금 시점에서 문자열과 객체의 차이를 설명할 수 없다면 조금 더 공부를 해야할 필요가 있습니다. 이 책은 Vue.js를 처음 사용하는 개발자는 물론 이미 Vue.js를 사용하고 더 깊이 공부하려는 개발자 모두에게 유용할 것입니다. 또한 다른 라이브러리에서 Vue.js 2로 옮기려는 개발자에게도 유용합니다.

## 저자와 연락하려면

이 책에 대해 문의하거나, 피드백을 보내거나, 관심이 필요한 기타 사항에 대해 궁금한 점이 있으면 언제든지 문의 해 주십시오.

Name	Email	Twitter
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Alex Kyriakidis	alex@tmvuejs.com	@hootlex
Kostas Maniatis	kostas@tmvuejs.com	@kostaskafcas

이 책의 번역자는 박창주입니다. 번역에 대한 의견 및 제안은 [pcjpcj2@gmail.com](mailto:pcjpcj2@gmail.com) 또는 트위터(@pcjpcj2)로 문의하십시오.

## 혼자 해보기

코드를 작성하면서 익히는 것이 가장 좋은 방법입니다. 스스로 해결하고 실제로 테스트할 수 있도록 풀어볼 만한 문제를 각 장의 마지막에 준비해 두었습니다. 가능한 한 많이 시도하여 문제를 해결하고 Vue.js에 대해 더 잘 이해하는 것을 적극적으로 권장합니다. 생각하고 있는 것을 시도하는 것을 두려워하지 마세요. 조금만 노력하면 더 많은 것을 할 수 있습니다. 어쩌면 몇 가지 다른 예제나 방법으로 적절한 아이디어를 얻을 수 있습니다. 물론 너무 과한 문제는 없을 것이며 힌트와 잠재적인 해결 방법을 함께 제공합니다.

이제 긴 여행을 시작할 때입니다!

## 예제 코드

이 책에서 사용된 대부분의 예제 코드는 Github에 있습니다. 코드를 살펴보기 위해 [여기<sup>9</sup>](#)에 들어가보세요.

다운받아서 보는 것이 더 편하면 '.zip' 파일을 [내려받으세요<sup>10</sup>](#).

예제 코드를 사용하면 책에 있는 내용을 복사하고 붙여넣으며 테스트하는 끔찍한 일을 피할 수 있습니다.

## 오탈자

정확한 컨텐츠를 제공하기 위해 모든 노력을 기울였음에도 실수가 있을 수 있습니다. 이 책을 읽으시면서 실수를 발견하면 알려주세요. 이는 다른 독자들을 혼란으로 부터 막을 수 있고 이 책의 개선된 후속 버전에 큰 도움이 됩니다. 만약 오탈자를 발견했다면 [Github 저장소<sup>11</sup>](#)에 이슈를 남겨주세요.

## 표기 규칙

이 책에서는 다음의 표기 규칙이 사용됩니다.

코드 블럭은 아래와 같습니다.

---

<sup>9</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2>

<sup>10</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/archive/master.zip>

<sup>11</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2>

JavaScript {lang="javascript"} function(x, y){ // 이 것은 주석입니다 }

글의 코드는 다음과 같이 표시 됩니다 “고정된 폭을 가지는 반응형 컨테이너는 .container를 사용합니다.”

새 용어와 중요 단어는 굵게 표시 합니다.

팁, 노트 및 경고는 다음과 같이 표시됩니다.



### 이 것은 경고 입니다.

이 곳에는 경고나 주의 사항을 표시합니다.



### 이 것은 팁 입니다.

이 곳에는 팁 또는 추천 사항을 표시합니다.



### 이 것은 정보 입니다.

이 곳에는 약간의 중요한 내용을 표시합니다.



### 이 것은 노트 입니다.

주제에 대한 노트를 표시합니다.



### 이 것은 힌트 입니다.

주제에 대한 힌트를 표시합니다.



### 이 것은 터미널 명령어 입니다.

터미널에서 실행할 명령어를 표시합니다.



### 이 것은 비교를 위한 내용입니다.

주제와 관련된 비교에 대한 내용을 표시합니다.



### 이 것은 Github 링크 입니다.

링크는 이 책의 저장소로 연결되며 코드 샘플 및 각 장의 과제를 찾을 수 있습니다.

# Vue.js 기초

# Vue.js 설치

Vue.js를 다운로드할 수 있는 몇 가지 방법이 있습니다.

## 독립 실행 버전

### vuejs.org에서 다운로드

Vue를 설치하려면 스크립트 태그에 포함하면 됩니다. Vue는 전역 변수로 등록됩니다.

Vue.js의 두 가지 버전을 다운로드할 수 있습니다.

- 개발용 버전은 [http://vuejs.org/js/vue.js<sup>12</sup>](http://vuejs.org/js/vue.js)입니다.
- 배포용 버전은 [http://vuejs.org/js/vue.min.js<sup>13</sup>](http://vuejs.org/js/vue.min.js)입니다.



팁: 개발 중에 배포용 버전을 사용하지 마십시오. 일반적인 실수에 대한 모든 자세한 경고를 놓치게됩니다.

### CDN에서 가져오기

Vue.js는 [jsdelivr<sup>14</sup>](#)와 [cdnjs<sup>15</sup>](#)에서 받을 수 있습니다.



항상 최신 버전을 사용하려면 반영되는데 약간의 시간이 필요하므로 자주 업데이트를 확인해야합니다.

## NPM 사용

NPM은 Vue.js를 사용하여 대규모 앱을 제작할 때 권장되는 설치 방법입니다. [Webpack<sup>16</sup>](#) 또는 [Browserify<sup>17</sup>](#)와 같은 CommonJS 모듈 번들러와 잘 조화를 이룹니다.

<sup>12</sup><http://vuejs.org/js/vue.js>

<sup>13</sup><http://vuejs.org/js/vue.min.js>

<sup>14</sup><https://cdn.jsdelivr.net/npm/vue@2.1.10/dist/vue.min.js>

<sup>15</sup><https://cdn.jsdelivr.net/npm/vue@2.1.10/dist/vue.min.js>

<sup>16</sup><http://webpack.github.io/>

<sup>17</sup><http://browserify.org/>

```
1 # 최신 안정 버전  
2 $ npm install vue  
3 # 최신 안정 버전 + CSP를 준수하는 버전  
4 $ npm install vue@csp  
5 # 개발 빌드 (GitHub에서 직접 설치합니다):  
6 $ npm install vuejs/vue#dev
```

## Bower 사용하기

```
1 # 최신 안정 버전  
2 $ bower install vue
```



Vue.js 설치 가이드<sup>18</sup>에 설치 방법 및 업데이트에 대한 자세한 내용이 있습니다.

대부분의 책 예제에서는 CDN을 이용한 Vue.js를 포함하고 있습니다. 원하는 방법으로 자유롭게 설치할 수 있습니다.

---

<sup>18</sup><http://vuejs.org/guide/installation.html>

# 시작하기

이 장은 Vue의 데이터 바인딩을 간단히 살펴봅니다. 메시지를 입력하고 표시하는 간단한 애플리케이션을 만듭니다. 웹페이지에서 실시간으로 Vue의 양방향 데이터 바인딩 기능을 보게 됩니다. Vue 애플리케이션은 HTML 페이지를 만들고 약간의 설정만 하면 됩니다.

이 과정에서 jQuery 대신 Vue.js를 사용하여 시간과 노력을 아낄 수 있습니다.

## 안녕하십니까 전하!

새로운 파일을 만들 것이고 필요 없는 코드들을 삭제할 것입니다. `hello.html` 처럼 원하는 이름으로 만들면 됩니다.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <h1>Greetings your Majesty!</h1>
7 </body>
8 </html>
```

이 간단한 HTML 파일은 인삿말을 보여줍니다.

이제 Vue.js를 사용하여 동일한 작업을 합니다. 먼저 Vue.js를 웹페이지에 포함하고 새로운 인스턴스를 만듭니다.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <div id= " app " >
7     <h1>Greetings your majesty!</h1>
8   </div>
9 </body>
10 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.10/vue.min.js " ></script>
11 <script>
12   new Vue({
```

```

13     el: '#app'
14   })
15 </script>
16 </html>

```

Vue.js를 사용하려 [cdnjs<sup>19</sup>](#)를 **script** 태그로 포함하였고, 새로운 Vue 인스턴스를 만들었습니다. 참조할 엘리먼트인 **id**가 **#app**인 **div**를 Vue가 알아챌 수 있도록 가리킵니다. Vue가 작동하는 컨테이너로 생각하시면 됩니다. Vue는 가리키고 있는 엘리먼트 밖에 있는 것은 알지 못합니다. 원하는 엘리먼트를 가리키려면 **el** 옵션을 사용하세요.

이제 표시하려는 메시지를 **data** 객체 내부의 변수에 할당합니다. 그 다음 Vue 생성자에 대한 옵션으로 데이터 객체를 전달합니다.

```

1 var data = {
2   message: 'Greetings your majesty!'
3 };
4 new Vue({
5   el: '#app',
6   data: data
7 })

```

페이지에 메시지를 표시하려면 메시지를 두개의 중괄호로 묶어야합니다. 따라서 메시지 안에 있는 내용은 **h1** 태그에 자동으로 나타납니다.

```

1 <div id="app">
2   <h1>{{ message }}</h1>
3 </div>

```

이것과 동일합니다 또 다른 방법으로 메시지 변수를 정의하는 **data** 객체의 Vue 생성자에서 직접하는 것입니다.

```

1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Greetings your Majesty!'
5   }
6 });

```

두가지 방법은 정확하게 같은 결과를 보여줍니다. 그러므로 선호하는 어떤 방법을 선택해도 괜찮습니다.

---

<sup>19</sup> <https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.10/vue.min.js>



## 안내

두개의 중괄호는 HTML이 아니라 스크립트 코드입니다. 이 안에 있는 것은 모두 바인딩 표현식이라고 부릅니다. 자바스크립트는 `{{ message }}`와 같은 표현식을 계산하여 출력합니다. 자바스크립트는 변수의 결과값을 가져옵니다. 이 코드 조각 `{{1+2}}`는 숫자 3을 표시합니다.

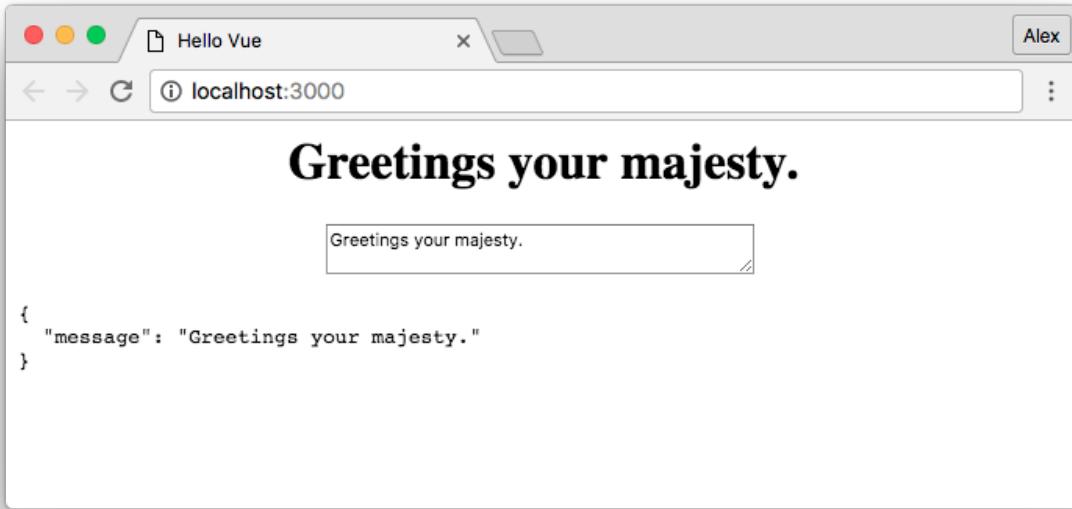
## 양방향 바인딩

Vue의 멋진 것들이 삶을 편하게 해줄 것 입니다. 사용자가 입력을 하면 메시지가 변경된다고 가정합시다. 어떻게 하면 이를 쉽게 할 수 있을까요? 아래의 예제는 `v-model`을 이용합니다. 이 것은 Vue의 디렉티브라고 부릅니다.(다음 장에서 디렉티브에 대한 자세한 내용을 알 수 있습니다.) 그런 다음 양방향 데이터 바인딩을 사용하여 사용자가 입력을 변경할 때 메시지 값을 동적으로 변경합니다. 데이터는 기본적으로 모든 입력 이벤트에 동기화됩니다.

```
1 <div id= "app" >
2   <h1>{{ message }}</h1>
3   <input v-model= "message" >
4 </div>
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Greetings your Majesty!'
5   }
6 })
```

이게 전부입니다. 이제 헤딩 메시지와 사용자 입력이 묶였습니다! `input` 태그 안의 `v-model`은 Vue에게 `message`가 `input`에 바인딩 됨을 알려줍니다.



### 양방향 데이터 바인딩

양방향 데이터 바인딩은 뷰에서 모델의 값을 변경하면 모든 것이 최신으로 유지된다는 것을 의미합니다.

## jQuery와 비교

아마 jQuery에 대한 약간의 경험이 있을 것 입니다. 아니어도 괜찮습니다. 이번에는 jQuery의 일부분만 사용합니다. jQuery 대신 Vue를 이용하는 방법만 구현할 것 입니다. 이 책을 읽는 모두가 Vue.js를 사용할 수 있기를 바랍니다.

데이터 바인딩을 이용하는 방법이 어떻게 애플리케이션을 만드는데 도움이 되는지 더 잘 이해하기 위해 잠시 멈추고 이전의 예제를 만드는 방법을 생각해보세요. 입력 엘리먼트를 만들고 **id** 또는 **class**를 지정하면 됩니다. 이를 가리킨 후 갱신하면 됩니다. 그런 다음 원하는 엘리먼트를 입력 값과 일치하도록 변경하는 함수를 호출합니다. 이는 keyup 이벤트가 발생할 때마다 반복됩니다. 정말 괴로운 일입니다.

더 추가하거나 뺄 것 없이 이렇게 하면 됩니다.

```
1 <html>
2 <head>
3     <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id= " app " >
7     <h1>Greetings your Majesty!</h1>
8     <input id= " message " >
9 </div>
10 </body>
11 <script src= " https://code.jquery.com/jquery-2.1.4.min.js " ></script>
12 <script type= " text/javascript " >
13     $( '#message' ).on( 'keyup' , function(){
14         var message = $( '#message' ).val();
15         $( 'h1' ).text(message);
16     })
17 </script>
18 </html>
```

비교를 위한 간단한 예제입니다. 보시다시피, Vue가 훨씬 더 아름답게 보일 것입니다. 시간이 적게 걸릴 뿐 아니라 파악하기도 쉽습니다. 물론 jQuery는 DOM을 조작하는데 매우 강력한 라이브러리이지만 사용하기에는 약간의 노력이 필요합니다!



## 코드 예제

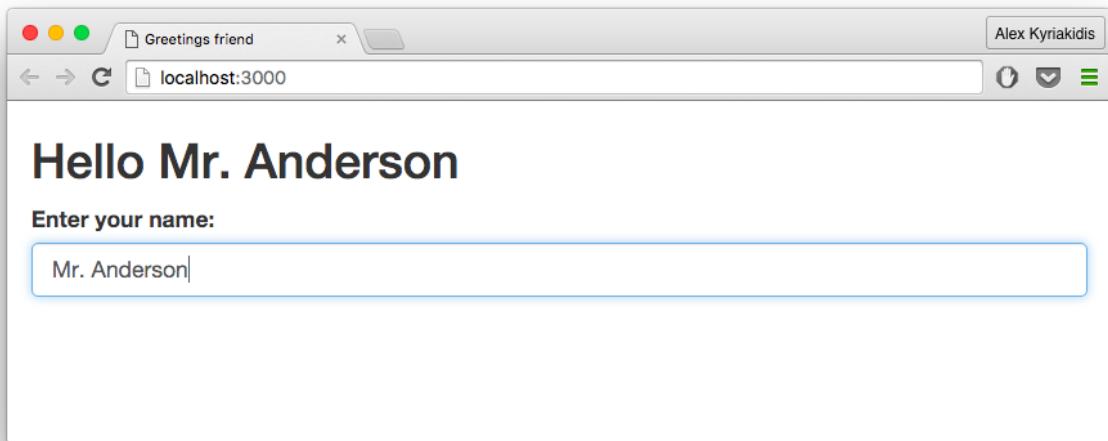
이 장의 예제들은 GitHub<sup>20</sup>에서 찾으실 수 있습니다.

---

<sup>20</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/codes/chapter2.html>

## 혼자 해보기

간단하고 훌륭한 초보자용 예제로 `{{name}}` 헤딩을 만들어 봅시다. 그리고 `name`을 입력 태그에 바인딩하세요. 상상하고 있듯이 헤딩은 입력이 변경될 때마다 즉시 변경되어야 합니다. 한번 해보세요, 행운을 빕니다!



결과 예시



### 노트

예제들은 부트스트랩을 사용합니다. 아직 부트스트랩에 익숙하지 않다면 지금은 무시해도 괜찮습니다. 마지막 장에서 다루고 있습니다.



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>21</sup>](#)를 참조하세요.

---

<sup>21</sup><https://github.com/hoottex/the-majesty-of-vuejs-2/blob/master/homework/chapter2.html>

# 디렉티브

이 장에서는 Vue의 디렉티브의 몇가지 예를 알아보겠습니다. 아직 Vue.js 또는 Angular.js 같은 프레임워크를 사용해 본 경험이 없다면, 아마도 디렉티브를 들어본 적 없을 것 입니다. 디렉티브는 라이브러리에서 DOM 엘리먼트가 무언가를 수행하도록 지시하는 특수한 토큰입니다. 꼭 기억해야 합니다. Vue.js에서 디렉티브의 개념은 Angular 보다 훨씬 더 간단합니다. 디렉티브 중 일부를 살펴보겠습니다.

- **v-show** : 조건이 참인 경우 화면에 출력합니다.
- **v-if** : **v-show**와 동일합니다.
- **v-else** : **v-if**가 거짓이면 출력합니다.

그리고 특별한 구문이 필요한 **v-for**가 있으며 렌더링을 위해 필요합니다(예: 배열 안의 항목들을 출력할 때 사용합니다.) 디렉티브를 사용하는 방법을 자세히 알아볼 예정입니다.

위에서 소개한 디렉티브를 한번 살펴보겠습니다.

## v-show

첫 번째 디렉티브를 설명하기 위해 간단한 예제를 만들어봅니다. 이해하기 쉽고 실제로 사용하기 쉬운 몇 가지 팁을 얻을 수 있습니다. 몇 가지 기준에 따라 엘리먼트를 출력할 필요가 있다고 가정해보겠습니다. 처음에 메시지를 입력하지 않으면 제출 버튼이 표시되지 않아야 합니다. Vue로 어떻게 할 수 있을까요?

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <div id="app">
7     <textarea></textarea>
8   </div>
9 </body>
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
11 <script>
12   new Vue({
13     el: '#app',
14     data: {
```

```

15           message: 'Our king is dead!'
16       }
17   })
18 </script>
19 </html>

```

이제 HTML 파일 하나를 만들었고 `div id= " app "` 와 `textarea`를 추가했습니다. `textarea` 안에 출력할 메시지가 있습니다. 당연히, 아직 바인딩 되지 않았다는 것을 눈치챘을 겁니다. 또한 이 예제는 Vue.js의 최소화 버전을 사용하지 않습니다. 이전에 언급하였듯이, 일반적인 실수에 대한 경고를 놓치지 않기 위해 개발 중에는 최소화 버전을 사용하면 안됩니다. 책에서는 개발용 버전을 사용하지만 원하는 다른 버전을 사용해도 문제되지 않습니다.

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id= " app ">
7   <textarea v-model= " message "></textarea>
8   <pre>
9     {{ $data }}
10  </pre>
11 </div>
12 </body>
13 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
14 <script>
15   new Vue({
16     el: '#app',
17     data: {
18       message: 'Our king is dead!'
19     }
20   })
21 </script>
22 </html>

```

`textarea`은 `message`를 `v-model`에 바인딩하여 메시지를 표시합니다. 이전에 만든 예제처럼 입력하는 동시에 값이 바뀔 것입니다. 추가적으로, `pre` 태그 안에 데이터를 표시하고 있습니다. 이 작업은 Vue 인스턴스의 데이터를 가져옵니다. `json` 형태로 브라우저에 데이터를 보여줍니다. Vue는 문자열, 숫자, 배열 또는 일반 객체 여부에 관계없이 자동으로 정렬하여 출력합니다. 이것이 데이터를 만들고 조작하는데 훨씬 더 좋은 방법이라고 생각합니다. 콘솔을 통해 지속적으로 바라보는 것보다 편합니다.



## 정보

JSON(자바스크립트 객체 표기법) 은 가벼운 데이터 교환 포맷입니다. [여기<sup>22</sup>](#)에서 더 자세한 내용을 확인하세요. `{{ $data }}`의 결과는 Vue 데이터에 바인딩되며 변경 될 때마다 갱신됩니다.

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id= " app " >
7   <h1>You must send a message for help!</h1>
8   <textarea v-model= " message " ></textarea>
9   <button v-show= " message " >
10    Send word to allies for help!
11  </button>
12  <pre>
13    {{ $data }}
14  </pre>
15 </div>
16 </body>
17 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
18 <script>
19   new Vue({
20     el: '#app',
21     data: {
22       message: 'Our king is dead! Send help!'
23     }
24   })
25 </script>
26 </html>
```

계속해서, `h1` 태그에서 몇 가지 기준에 따라 토글할 간단한 경고 메시지를 만들어 보겠습니다. 그 옆에는 조건에 따라 표시되는 버튼을 만듭니다. 이 버튼은 메시지가 있는 경우에만 나타납니다. `textarea`에 아무런 내용이 없으면 데이터의 내용도 사라지며 버튼의 `display` 속성은 자동으로 ‘none’이 되어 사라집니다.



## 정보

`v-show`를 가지는 엘리먼트는 항상 렌더링 되어 DOM에 남아있습니다. `v-show`는 단순히 CSS 속성 `display`을 토글합니다.

---

<sup>22</sup><http://www.json.org/>

```

1 <h1 v-show=" !message " >You must send a message for help!</h1>
2 <textarea v-model=" message " ></textarea>
3 <button v-show=" message " >
4     Send word to allies for help!
5 </button>

```

위 예제를 통해 다른 엘리먼트를 토글하는 방법을 알 수 있습니다. 이 단계에서, **h1**의 경고 메시지를 숨길 필요가 있습니다. 메시지가 있는 경우 **style**을 **display: none**으로 바꿉니다.

## v-if

이 시점에서 “앞서 말한 v-if 는 뭔가요?”라고 궁금해 하실 수 있습니다. 그래서 앞의 예제를 다시 만들어 볼 것입니다. 이번에는 **v-if**를 사용합니다!

```

1 <html>
2 <head>
3     <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id=" app " >
7     <h1 v-if=" !message " >You must send a message for help!</h1>
8     <textarea v-model=" message " ></textarea>
9     <button v-if=" message " >
10        Send word to allies for help!
11    </button>
12    <pre>
13        {{ $data }}
14    </pre>
15 </div>
16 </body>
17 <script src=" https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
18 <script>
19     new Vue({
20         el: '#app',
21         data: {
22             message: 'Our king is dead! Send help!'
23         }
24     })
25 </script>
26 </html>

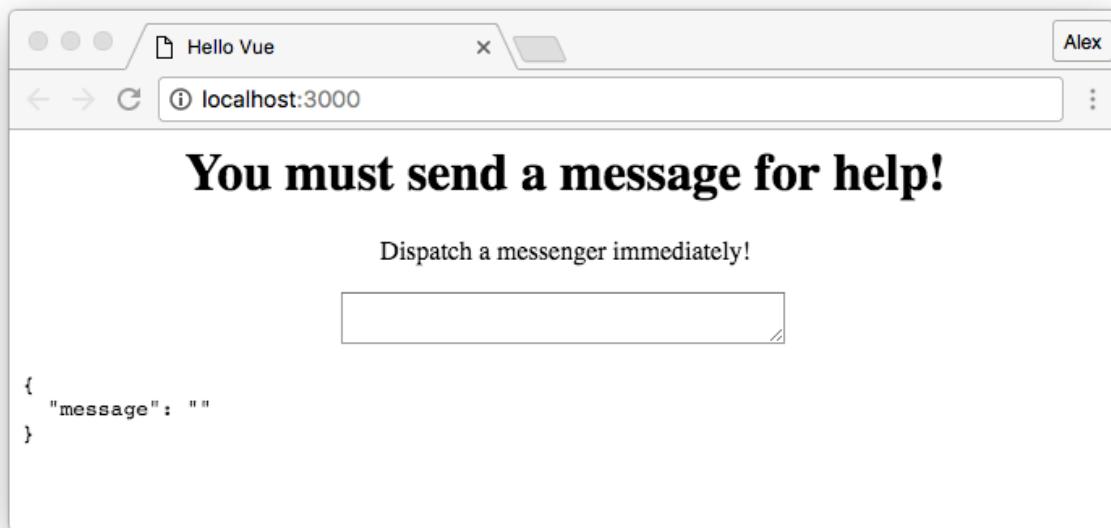
```

보시다시피, **v-show**를 **v-if**로 바꾼 것은 생각한대로 잘 동작합니다. 이를 직접 실험해보고 어떻게 작동하는지 살펴보세요! **v-if**는 DOM에 엘리먼트를 남기지 않는 것이 **v-show**와의 유일한 차이점입니다.

## v-if 템플릿

언제든지, 여러개의 엘리먼트를 한번에 토글해야하는 경우 `<template>`에 `v-if`를 사용하세요. `div` 또는 `span`의 사용이 적절하지 않은 곳에도 `<template>`을 보이지 않는 래퍼로 사용할 수 있습니다. `<template>`는 최종 결과에 렌더링 되지 않습니다.

```
1 <div id= " app " >
2   <template v-if= " !message " >
3     <h1>You must send a message for help!</h1>
4     <p>Dispatch a messenger immediately!</p>
5     <p>To nearby kingdom of Hearts!</p>
6   </template>
7   <textarea v-model= " message " ></textarea>
8   <button v-show= " message " >
9     Send word to allies for help!
10  </button>
11  <pre>
12    {{ $data }}
13  </pre>
14 </div>
```



### v-if 템플릿

이전 예제를 이용하여 `template` 엘리먼트에 `v-if` 디렉티브를 추가했습니다. 중첩된 모든 엘리먼트의 존재 여부를 토글합니다.



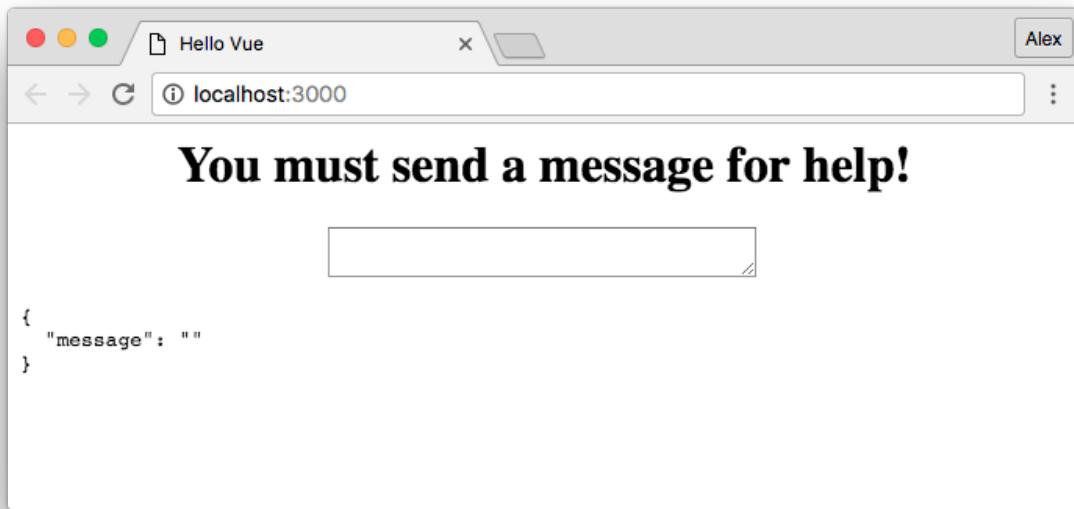
## 주의사항

v-show는 <template>에서 사용할 수 없습니다.

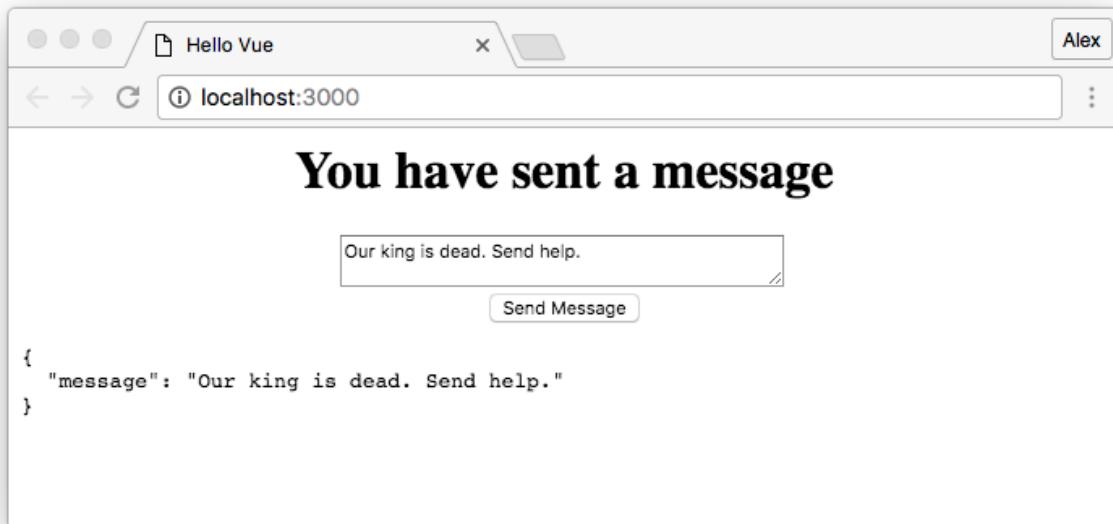
## v-else

이미 예상하셨겠지만 v-if를 사용하는 경우 v-else 디렉티브를 사용하여 “else 블록”을 지정 할 수 있습니다. v-else 디렉티브는 v-if의 다음에 나와야 합니다. 그렇지 않으면 인식되지 않습니다.

```
1 <html>
2 <head>
3     <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id= " app " >
7     <h1 v-if= " !message " >You must send a message for help!</h1>
8     <h2 v-else>You have sent a message!</h2>
9     <textarea v-model= " message " ></textarea>
10    <button v-show= " message " >
11        Send word to allies for help!
12    </button>
13    <pre>
14        {{ $data }}
15    </pre>
16 </div>
17 </body>
18 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
19 <script>
20     new Vue({
21         el: '#app',
22         data: {
23             message: 'Our king is dead! Send help!'
24         }
25     })
26 </script>
27 </html>
```



v-if 사용



v-else 사용

예제를 위해 다른 경고 메시지와 함께 **h2** 태그를 사용했습니다. 이 경고 메시지도 조건에 따라 동작합니다. 메시지가 없으면 **h1** 태그가 표시됩니다. 메시지가 있는 경우에 Vue의 **v-if**와 **v-else**의 간결한 구문을 사용하여 **h2**가 출력되는 것을 볼 수 있습니다.



## 주의사항

Vue 2.0 이후로 **v-else** 디렉티브는 **v-show**와 함께 사용할 수 없습니다.

## v-if와 v-show의 차이

**v-if**와 **v-show**의 차이점을 이미 언급했지만, 더 자세히 살펴보겠습니다. 사용 중에 몇 가지 궁금증이 생기셨을 겁니다. **v-show**와 **v-if**는 대체 어떤 큰 차이점이 있을까요? 성능에 영향을 주는 상황이 있을까요? 둘 중 하나를 사용하는 것이 더 나은 곳이 있나요? 많은 상황에서 **v-show**를 사용하면 페이지 렌더링 중에 더 많은 로드 시간이 발생한다는 것을 경험하실 수 있을 겁니다. 이와 반대로 Vue.js의 가이드에 따르면 **v-if**는 진정한 조건부 렌더링입니다.

**v-if**를 사용할 때 초기 렌더링에서 조건이 거짓이면 아무것도 하지 않습니다.  
\_조건이 true가 될 때까지 블럭안의 내용이 렌더링 되지 않습니다. 일반적으로  
**v-if**는 토큰 비용이 높지만 **v-show**는 초기 렌더링 비용이 높습니다. 매우 자주  
전환해야 하는 경우 **v-show**를 사용하고, 자주 변경할 필요가 없는 경우 **v-if**를  
사용하세요.

그래서, 언제든지 필요에 따라 선택해야 합니다.



## 예제 코드

이 장의 코드 예제는 [GitHub<sup>23</sup>](#)에 있습니다.

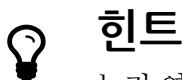
---

<sup>23</sup><https://github.com/hoottex/the-majesty-of-vuejs-2/tree/master/codes/chapter3>

## 혼자 해보기

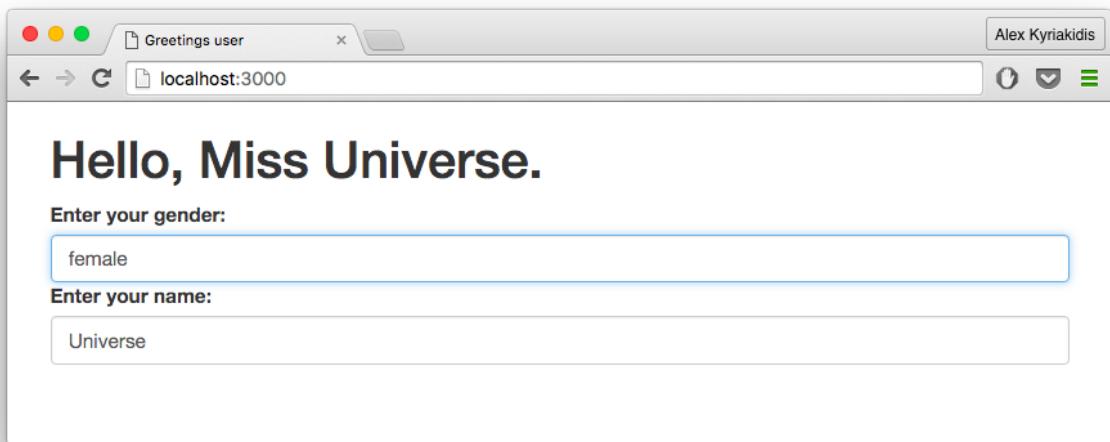
이전의 혼자 해보기에서 했던 것을 조금 더 확장해 봅시다. 이제 사용자는 성별과 함께 자신의 이름을 입력합니다. 사용자가 남성(male)인 경우 “안녕하세요 {{name}}씨. 당신은 남성이군요.” 사용자가 여성(female)인 경우 “안녕하세요 {{name}}씨. 당신은 여성입니다.”라고 출력되어야 합니다.

성별이 남녀가 아닌 경우 사용자는 경고 메시지 “선택할 수 없나요? 괜찮습니다!”



### 힌트

논리 연산자는 사용자의 호칭(Mr 또는 Miss)을 결정할 때 편리합니다.



결과 예시



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>24</sup>](#)를 참조하세요.

---

<sup>24</sup><https://github.com/hoottex/the-majesty-of-vuejs-2/blob/master/homework/chapter3.html>

# 리스트 렌더링

이번 장에서는 리스트 렌더링을 알아보겠습니다.

Vue의 디렉티브를 사용하여 다음 내용들을 할 수 있습니다.

1. 배열 기반의 리스트를 출력할 수 있습니다.
2. 템플릿을 반복할 수 있습니다.
3. 객체의 내부를 순회할 수 있습니다.

## 부트스트랩 설치와 사용

작업을 쉽게 하기위해 부트스트랩을 사용합니다.



### 정보

부트스트랩은 모바일을 우선하는 반응형 웹사이트를 개발하기 위한 프레임워크입니다.

<http://getbootstrap.com/><sup>25</sup>에서 다운로드 버튼을 클릭하세요. 당분간 CDN link<sup>26</sup>에서 부트스트랩을 사용하지만 필요하면 다른 방법으로 설치하셔도 됩니다. 예를 들어 지금은 `css/bootstrap.min.css` 파일 하나만 필요합니다. 이 앱에서 `.css` 파일을 사용하면 보기 좋은 구조와 스타일을 사용할 수 있습니다. 페이지의 `head` 태그 안에 그냥 추가하기만 하면 됩니다.

부트스트랩은 사이트 내용을 그리드 시스템으로 바꾸기 위한 엘리먼트를 제공합니다. 이 프로젝트에서 사용할 두개의 컨테이너 중 하나를 선택할 수 있습니다. `padding` 등으로 인하여 컨테이너는 중첩될 수 없습니다.

- 반응형 고정 네비 컨테이너의 경우에 `.container`를 사용하세요.

```
1 <div class="container">
2 ...
3 </div>
```

- 전체 네비를 사용하는 컨테이너를 위해 `.container-fluid` 사용하세요. 사용자의 뷰포트 전체 네비를 기준으로 표시됩니다.

---

<sup>25</sup><http://getbootstrap.com/>

<sup>26</sup><https://www.bootstrapcdn.com/>

```
1 <div class= " container-fluid " >
2 ...
3 </div>
```

이 시점에서 부트스트랩 클래스를 사용하여 Vue.js 예제를 만듭니다. 물론 Vue와 부트스트랩을 결합하여 사용하는데 많은 연구나 실험이 필요하지는 않습니다.

```
1 <html>
2 <head>
3 <link href= " https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css " rel= " st\
ylesheet " >
4     <title>Hello Bootstrap</title>
5 </head>
6 <body>
7     <div class= " container " >
8         <h1>Hello Bootstrap, sit next to Vue.</h1>
9     </div>
10 </body>
11 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
12 <script type= " text/javascript " >
13     new Vue({
14         el: '.container'
15     })
16 </script>
17 </html>
```

이번에는 **app** id를 가지는 엘리먼트를 Vue 앱으로 사용하는 대신 **el** 옵션 내의 **container** 클래스를 사용하겠습니다.



## 팁

위의 예제에서 **.container** 클래스를 가진 엘리먼트를 대상으로 합니다. 주의하세요  
클래스를 가리키는 엘리먼트로 사용할 때 클래스가 1개 이상 존재하면 Vue.js는  
첫번째 엘리먼트에만 마운트 됩니다.

**el:** 속성은 CSS 셀렉터 또는 실제 HTML 엘리먼트일 수 있습니다. 루트 인스턴스를  
**<html>** 또는 **<body>**에 마운트 하는 것은 권장하지 않습니다.

## v-for

배열의 각 항목을 반복하려면 **v-for** 디렉티브를 사용해야 합니다. 이 디렉티브는 **item in array** 형태의 특수 구문이 필요합니다. 여기서 **array**는 원본 데이터 배열이고 **item**은 반복되는 배열 엘리먼트의 별칭입니다.



### 주의사항

php를 사용하던 개발자에게는 **v-for**가 php의 **foreach**와 비슷해 보일 것입니다. 하지만 **foreach(\$array as \$value)**에 익숙하다면 주의가 필요합니다. Vue의 **v-for**는 정확히 반대입니다. **value in array**로 사용합니다.  
단수는 앞에, 복수는 뒤에 있습니다.

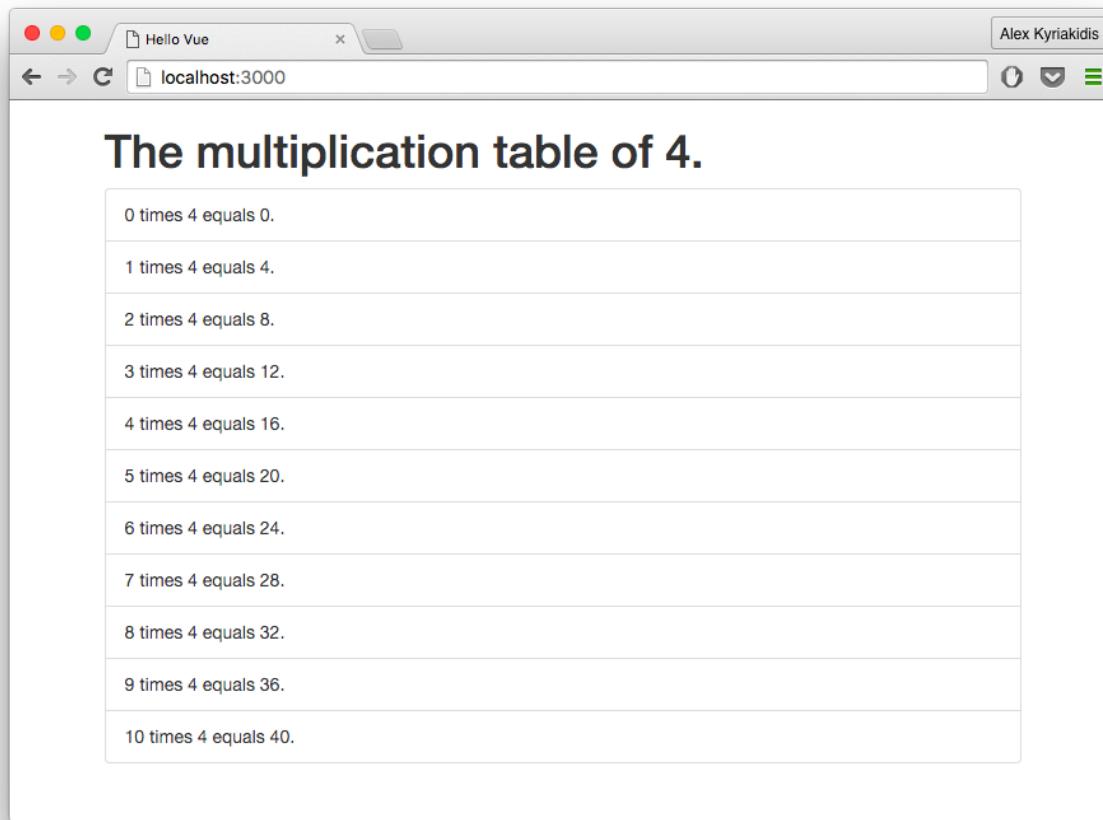
### 범위를 가지는 v-for

디렉티브 **v-for** 또한 정수를 사용할 수 있습니다. 배열/객체 대신에 숫자가 전달될 때마다 템플릿은 주어진 수만큼 반복합니다.

```

1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel\
4 = "stylesheet" >
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>The multiplication table of 4.</h1>
10    <ul class="list-group">
11      <li v-for="i in 11" class="list-group-item">
12        {{ i-1 }} times 4 equals {{ (i-1) * 4 }}.
13      </li>
14    </ul>
15  </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
18 <script type="text/javascript">
19   new Vue({
20     el: '.container'
21   })
22 </script>
23 </html>
```

위 코드는 테이블에 구구단 4단을 출력합니다.



## 구구단 4단



### 노트

구구단 4단을 (40까지) 표시해야 합니다. `i`의 첫 값은 1 이므로 템플릿을 11번 반복합니다.

## 배열 렌더링

### 배열을 통한 반복

다음 예제에서 데이터 객체 내에서 Stories 배열을 만들고 하나씩 차례대로 표시합니다.

```
stories: [
    "I crashed my car today!",
    "Yesterday, someone stole my bag!",
    "Someone ate my chocolate...",
]
```

이제 해야할 일은 리스트를 출력하는 것입니다. 이번에는 특별히 문자열을 가지는 배열입니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>Stories</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Let's hear some stories!</h1>
9         <div>
10            <ul class="list-group">
11                <li v-for="story in stories" class="list-group-item">
12                    Someone said "{{ story }}"
13                </li>
14            </ul>
15        </div>
16        <pre>
17            {{ $data }}
18        </pre>
19    </div>
20 </body>
21 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
22 <script type="text/javascript">
23     new Vue({
24         el: '.container',
25         data: {
26             stories: [
27                 "I crashed my car today!",
28                 "Yesterday, someone stole my bag!",
29             ]
30         }
31     })
32 </script>
```

```

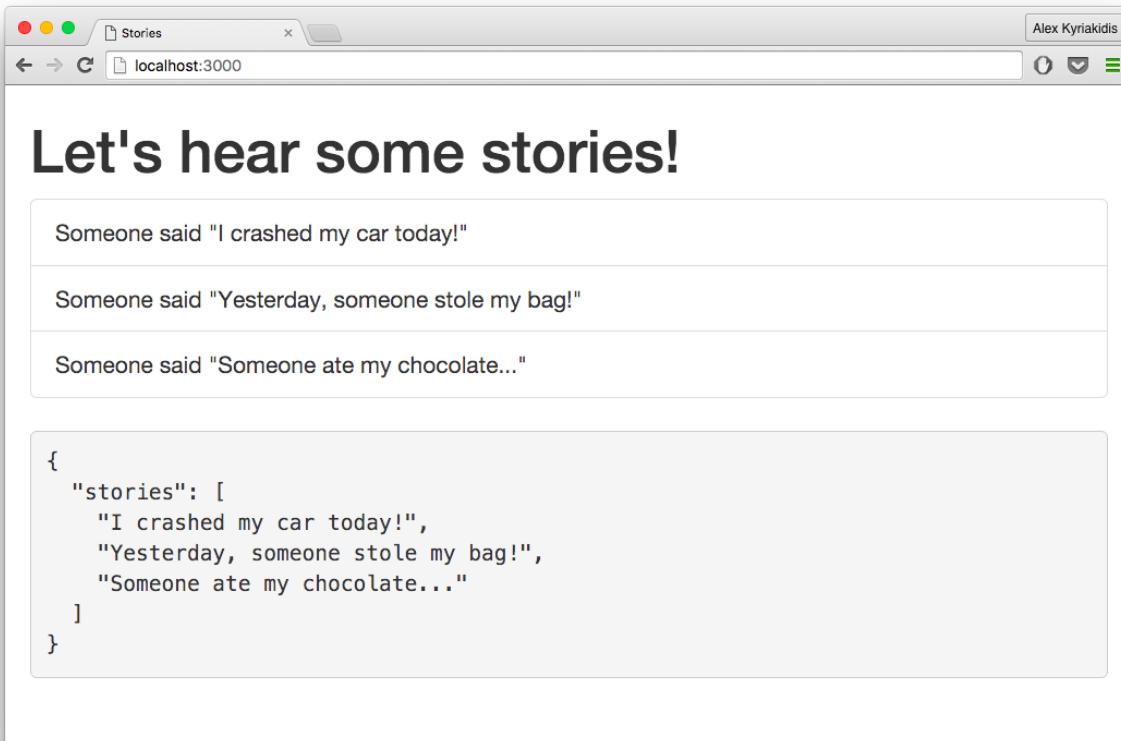
30           " Someone ate my chocolate... " ,
31       ]
32   }
33 })
34 </script>
35 </html>

```



## 정보

**list-group**과 **list-group-item** 클래스는 모두 부트스트랩 클래스입니다. [부트스트랩 리스트 스타일에 대한 자세한 내용을 확인하세요.](#)<sup>27</sup>



v-for로 배열 출력

**v-for**를 사용하여 단순히 순서가 없는 목록에 이야기를 표시할 수 있습니다. 정말 쉽습니다!

<sup>27</sup> <http://getbootstrap.com/css/#type-lists>

## 객체 배열을 통한 반복

Stories 배열에 **story** 객체를 포함하도록 변경합니다. **story** 객체에는 **plot**와 **writer** 속성이 있습니다. 이전 예제와 동일한 작업을 하겠습니다. 그러나 이번에는 **story**를 반복하는 대신 **story.plot** 및 **story.writer**를 각각 출력합니다.

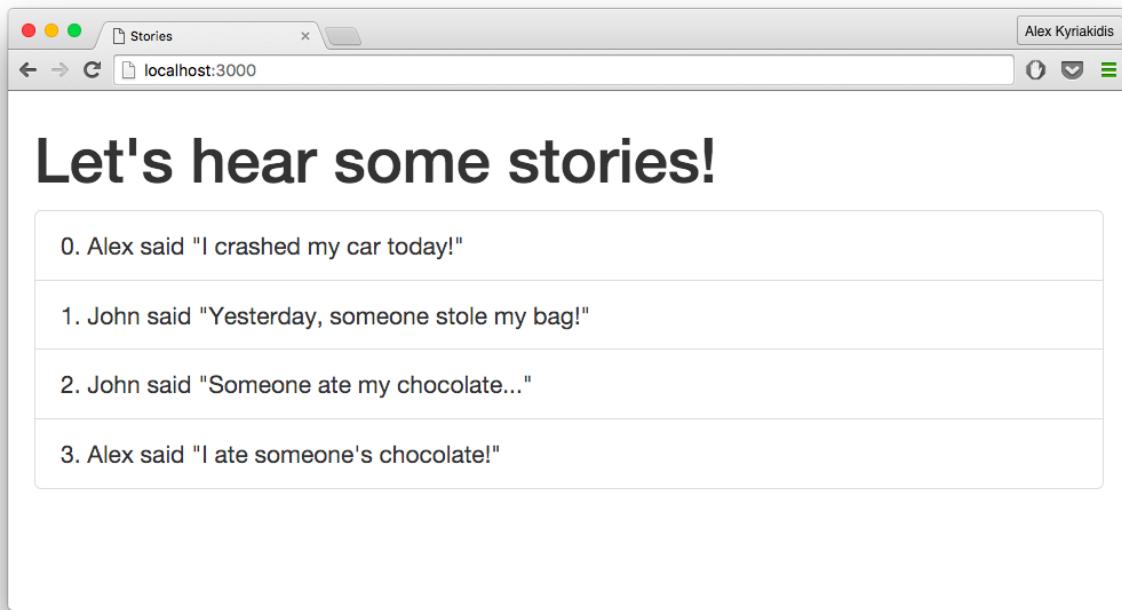
```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>Stories</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Let's hear some stories!</h1>
9         <div>
10             <ul class="list-group">
11                 <li v-for="story in stories"
12                     class="list-group-item">
13                     >
14                     {{ story.writer }} said "{{ story.plot }}"
15                 </li>
16             </ul>
17         </div>
18         <pre>
19             {{ $data }}
20         </pre>
21     </div>
22 </body>
23 <script src="https://cdn.jsdelivr.net/npm/vue@2.3.4/dist/vue.js"></script>
24 <script type="text/javascript">
25 new Vue({
26     el: '.container',
27     data: {
28         stories: [
29             {
30                 plot: "I crashed my car today!",
31                 writer: "Alex"
32             },
33             {
34                 plot: "Yesterday, someone stole my bag!",
35                 writer: "John"
36             },
37             {
38                 plot: "Someone ate my chocolate...",
39             }
40         ]
41     }
42 }
```

```
40         writer: "John"
41     },
42     {
43         plot: "I ate someone's chocolate!",
44         writer: "Alex"
45     },
46   ],
47 }
48 })
49 </script>
50 </html>
```

또한 현재 항목의 인덱스를 표시해야 하는 경우 특수 변수 **index**를 사용할 수 있습니다. 다음과 같이 작성합니다.

```
<ul class="list-group">
  <li v-for="(story, index) in stories"
      class="list-group-item">
    {{index}} {{story.writer}} said "{{story.plot}}"
  </li>
</ul>
```

중괄호 안의 **index**는 위 예제에서 항목에 대한 인덱스입니다.



배열과 인덱스 출력

## 객체 v-for

객체 속성을 반복하기 위해 **v-for**를 사용할 수 있습니다. 앞에서 배열의 **index**를 표시할 수 있지만 객체를 반복할 때에도 동일하게 사용할 수 있습니다. **index** 외에도 각각의 범위에서 **key**라는 특수한 속성에 접근할 수 있습니다.

### 정보

객체를 반복할 때, **index**의 범위는 0 … n-1 입니다. 여기서 n 은 객체 속성의 개수입니다.

이번에는 3개의 속성을 가진 하나의 객체로 데이터를 재구성 했습니다. **plot**, **writer** 그리고 **upvotes** 입니다.

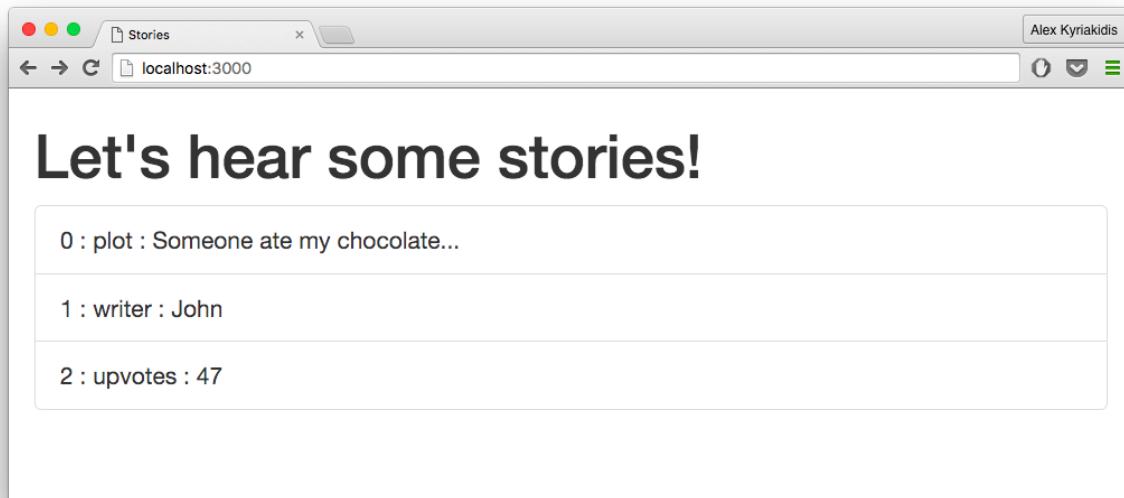
```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="value in story" class="list-group-item">
      {{ value }}
    </li>
  </ul>
</div>
```

```
new Vue({
  el: '.container',
  data: {
    story: {
      plot: "Someone ate my chocolate...",
      writer: 'John',
      upvotes: 47
    }
  }
})
```

**key**와 **index**를 각각 두번째와 세번째 전달인자로 사용할 수 있습니다.

```
1 <div class="container">
2   <h1>Let's hear some stories!</h1>
3   <ul class="list-group">
4     <li v-for="(value, key, index) in story"
5       class="list-group-item">
6       >
7       {{index}} : {{key}} : {{value}}
8     </li>
9   </ul>
10 </div>
```

위의 예제 코드에서 볼 수 있듯이, **key**와 **index**를 사용해 각 쌍의 키-밸류와 **index**를 가져옵니다. 결과는 다음과 같습니다.



객체 속성 출력



## 예제 코드

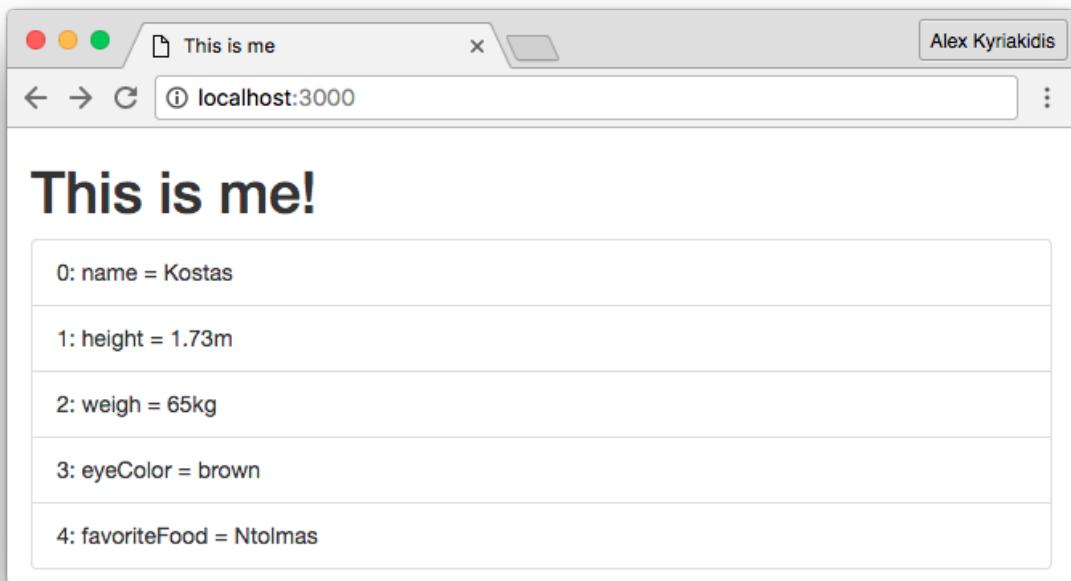
이 장의 예제 코드는 [GitHub<sup>28</sup>](#)에 있습니다.

<sup>28</sup> <https://github.com/hoootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter4>

## 혼자 해보기

이 장에서 살펴본 것을 기억하세요. 그리고 개인정보를 저장하는 객체를 만들어보세요. 개인정보에는 이름(name), 몸무게(weight), 키(height), 눈의 색(eyeColor) 그리고 좋아하는 음식(favoriteFood)이 있습니다.

`v-for`를 이용하여 각 속성을 `index: key = value` 형식으로 반복하여 표시해 보세요.



결과 예시



## 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>29</sup>](#)를 참조하세요.

---

<sup>29</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter4.html>

# 상호작용

이 장에서는 이전 예제를 확장하여 ‘메소드’, ‘이벤트 핸들링’ 및 ‘계산된 속성’과 관련된 새로운 내용들을 알아봅니다. 몇 가지 예제를 만드는데 여러가지 다양한 접근을 해보겠습니다. 계산기와 같이 작은 애플리케이션을 만들며 Vue의 상호작용에 대한 기능을 사용해 보겠습니다.

## 이벤트 핸들링

HTML 이벤트는 DOM 엘리먼트에서 일어납니다. HTML 페이지에서 Vue.js를 사용하면 반응적으로 이벤트가 발생합니다.

이벤트는 기본적인 사용자와의 상호작용에서 렌더링 모델에서 일어나는 일까지 모든 것을 할 수 있습니다. HTML 이벤트의 몇 가지 예입니다.

- 웹 페이지가 완전히 불러진 경우
- 사용자 입력 폼이 변경된 경우
- 버튼이 클릭된 경우
- 폼이 제출된 경우

이벤트 핸들링의 가장 중요한 점은 이벤트가 발생할 때마다 무언가 할 수 있다는 것입니다. Vue.js는 DOM 이벤트를 청취할 수 있게 **v-on** 디렉티브를 사용해야 합니다. **v-on** 디렉티브는 이벤트 리스너를 엘리먼트에 붙입니다. 이벤트의 유형은 전달인자로 표시됩니다. 예를 들어 **v-on:keyup**은 **keyup** 이벤트를 청취합니다.



### 정보

**keyup** 이벤트는 사용자가 키를 놓을 때 발생합니다. HTML 이벤트의 전체 목록<sup>30</sup>입니다.

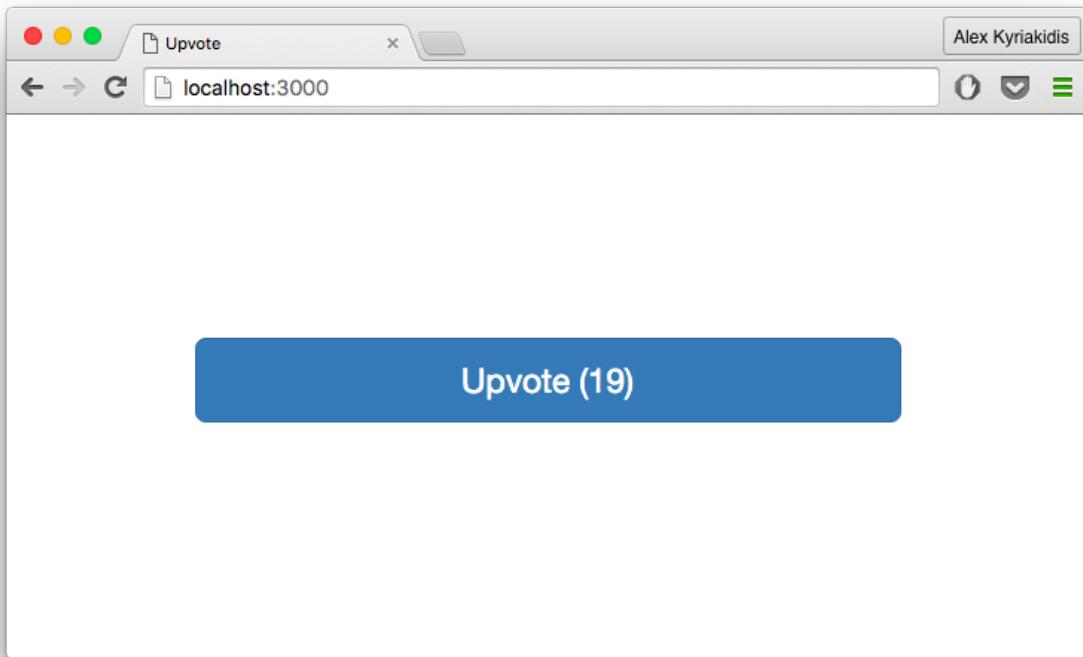
## 인라인 이벤트 핸들링

말하는 것만으로도 충분하지만 좀 더 나아가 이벤트 핸들링에 대해 알아봅시다. 아래에 클릭한 만큼 투표 수를 추가하는 ‘Upvote’ 버튼 예제가 있습니다.

---

<sup>30</sup> [http://www.w3schools.com/tags/ref\\_eventattributes.asp](http://www.w3schools.com/tags/ref_eventattributes.asp)

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Upvote</title>
5 </head>
6 <body>
7   <div class="container">
8     <button v-on:click="upvotes++">
9       Upvote! {{upvotes}}
10    </button>
11  </div>
12 </body>
13 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
14 <script type="text/javascript">
15 new Vue({
16   el: '.container',
17   data: {
18     upvotes: 0
19   }
20 })
21 </script>
22 </html>
```



투표 카운터

데이터에는 **upvotes** 변수가 있습니다. 이 경우, **click** 이벤트 리스너를 그 옆의 코드로 바인딩 합니다. 따옴표 안에 있는 내용은 버튼을 클릭할 때마다 upvotes를 하나씩 증가합니다. 증가 연산자(**upvotes++**)를 이용합니다.

## 메소드를 이용한 이벤트 핸들링

이제 메소드를 사용하여 이전과 똑같은 예제를 만들어 보겠습니다. Vue.js의 메소드는 특정 작업을 하도록 설계된 코드 블럭입니다. 메소드를 실행하려면, 메소드를 정의한 다음 호출해야 합니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Upvote</title>
5 </head>
6 <body>
7   <div class="container">
8     <button v-on:click="upvote">
```

```
10      Upvote! {{upvotes}}
11      </button>
12  </div>
13 </body>
14 <script type= "text/javascript" src= "https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.j\
15 s"></script>
16 <script type= "text/javascript" >
17 new Vue({
18   el: '.container',
19   data: {
20     upvotes: 0
21   },
22   // ** 'methods' ** 객체 아래에 메소드를 정의하세요
23   methods: {
24     upvote: function(){
25       // ** 'this' **는 메소드 안에서 Vue인스턴스를 가리킵니다
26       this.upvotes++;
27     }
28   }
29 })
30 </script>
31 </html>
```

메소드 'upvote'에 클릭 이벤트를 바인딩 했습니다.

이전과 마찬가지로 동작합니다. 하지만 코드를 읽을 때 명확하게 이해하기 쉽습니다.



## 주의사항

이벤트 핸들러는 하나의 명령문만 실행하도록 제한됩니다.

## v-on 축약형

프로젝트에서 v-on을 사용하여 작업을 하고 있다면 HTML이 빠르게 지저분해 질 것입니다. 고맙게도, v-on의 축약형인 @이 있습니다. @은 v-on:을 대체합니다. 코드를 매우 깔끔하게 만들어 줍니다. 이러한 축약형은 선택 사항입니다. @를 사용하면 이전 예제의 버튼이 다음과 같이 됩니다.

v-on:을 이용한 'click' 이벤트 처리

---

```
<button v-on:click= " upvote " >
    Upvote! {{upvotes}}
</button>
```

---

@ 측약형을 이용한 'click' 이벤트 처리

---

```
<button @click= " upvote " >
    Upvote! {{upvotes}}
</button>
```

---

## 이벤트 수식어

이제 계산기 애플리케이션을 만들어 보겠습니다. 이를 위해 2개의 입력과 하나의 드롭다운이 있는 폼을 사용합니다. 다음 코드는 문제 없으나 계산기가 예상한대로 작동하지 않습니다.

```

1 <html>
2 <head>
3     <title>Calculator</title>
4     <link href= " https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css " rel= "stylesheet " >
5 </head>
6 <body>
7     <div class= " container " >
8         <h1>Type 2 numbers and choose operation.</h1>
9         <form class= " form-inline " >
10            <!-- 입력을 숫자로 파싱하기 위해
11               특수 수식어 'number'를 사용합니다. -->
12            <input v-model.number= " a " class= " form-control " >
13            <select v-model= " operator " class= " form-control " >
14                <option>+</option>
15                <option>-</option>
16                <option>*</option>
17                <option>/</option>
18            </select>
19            <!-- 입력을 숫자로 파싱하기 위해
20               특수 수식어 'number'를 사용합니다. -->
21            <input v-model.number= " b " class= " form-control " >
22            <button type= " submit " @click= " calculate "
23                class= " btn btn-primary " >
24                Calculate
25
```

```

26      </button>
27  </form>
28  <h2>Result: {{a}} {{operator}} {{b}} = {{c}}</h2>
29  <pre>
30      {{ $data }}
31  </pre>
32 </div>
33 </body>
34 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
35 <script type="text/javascript">
36     new Vue({
37         el: '.container',
38         data: {
39             a: 1,
40             b: 2,
41             c: null,
42             operator: "+",
43         },
44         methods: {
45             calculate: function() {
46                 switch (this.operator) {
47                     case "+":
48                         this.c = this.a + this.b
49                         break;
50                     case "-":
51                         this.c = this.a - this.b
52                         break;
53                     case "*":
54                         this.c = this.a * this.b
55                         break;
56                     case "/":
57                         this.c = this.a / this.b
58                         break;
59                 }
60             }
61         },
62     });
63 </script>
64 </html>

```

이 코드를 실행하려고 “계산” 버튼을 클릭하면 계산하는 대신 페이지를 다시 로드합니다.

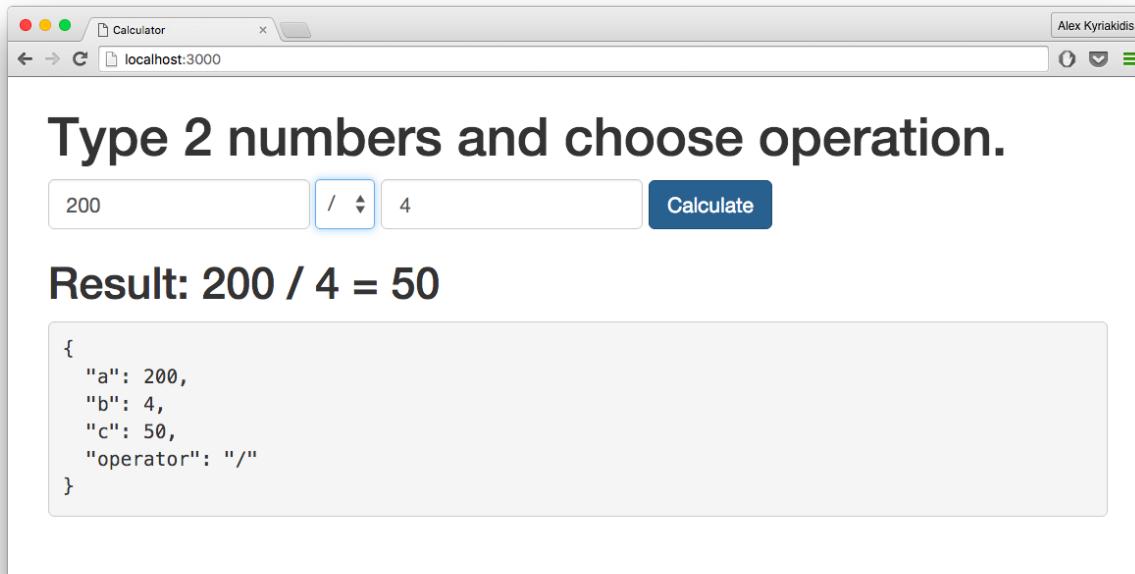
이는 당연합니다. “계산” 버튼을 클릭하면 폼 내용을 제출하기 때문에 페이지가 다시 로드됩니다.

제출을 막으려면 **onsubmit** 이벤트의 기본 동작을 막아야 합니다. 이벤트를 핸들링하는 메

소드에서 `event.preventDefault()`를 호출하는 것은 매우 보편적입니다. 이번 예제의 경우 이벤트 핸들링 메소드는 `calculate`입니다.

따라서 이렇게 될 것입니다.

```
calculate: function(event){  
    event.preventDefault();  
    switch (this.operator) {  
        case "+":  
            this.c = this.a + this.b  
            break;  
        case "-":  
            this.c = this.a - this.b  
            break;  
        case "*":  
            this.c = this.a * this.b  
            break;  
        case "/":  
            this.c = this.a / this.b  
            break;  
    }  
}
```



계산기를 위한 이벤트 수식어

메소드 안에서 더 쉽게 할 수 있지만, 아예 DOM 이벤트를 막는 코드가 없이 데이터를 처리하는 로직만 있다면 더 좋을 것 입니다.

Vue.js는 이벤트의 기본 동작을 방지하기 위해 `v-on`에 대해 네가지 이벤트 수식어를 제공합니다.

1. `.prevent`
2. `.stop`
3. `.capture`
4. `.self`

따라서 `.prevent`를 사용하여 아래의 제출 버튼을 변경하면

```
1 <button type= "submit" @click= "calculate" >Calculate</button>
```

이렇게 됩니다.

```
1 <!-- submit 이벤트는 더이상 페이지를 새로고침하지 않습니다 -->
2 <button type= "submit" @click.prevent= "calculate" >Calculate</button>
```

그리고 `calculate` 메소드에서 안전하게 `event.preventDefault()`를 제거할 수 있습니다.



## 노트

`.capture`와 `.self`는 거의 사용 되지 않으므로 더이상 신경쓰지 않아도 됩니다. 이벤트 순서에 관심이 있으시면 이 [튜토리얼<sup>31</sup>](#)을 보세요.

## 키 수식어

입력 중 하나에 포커스를 주고 엔터 키를 누르면 `calculate` 메소드가 호출 되는 것을 알 수 있습니다. 버튼이 폼에 없거나 버튼이 전혀 없는 경우 키보드 이벤트를 대신 청취할 수 있습니다.

키보드 이벤트를 청취할 때 종종 키 코드를 확인해야 합니다. **엔터** 버튼의 키 코드는 13입니다. 그래서 다음과 같이 사용할 수 있습니다.

---

<sup>31</sup> [http://www.quirksmode.org/js/events\\_order.html](http://www.quirksmode.org/js/events_order.html)

```
1 <input v-model="a" @keyup.13="calculate">
```

모든 키코드를 외우기는 어렵습니다. 그래서 Vue는 가장 일반적으로 사용되는 키의 별칭을 제공합니다.

- enter
- tab
- delete
- esc
- space
- up
- down
- left
- right

따라서 이번 예제에서 엔터 키를 누를 때 calculate 메소드를 실행하려면 입력은 다음과 같이 바꿔야합니다.

```
1 <input v-model="a" @keyup.enter="calculate">
2 <input v-model="b" @keyup.enter="calculate">
```



### 팁

입력과 버튼 등 많은 폼의 종류가 있고 기본 제출 동작을 방지하려면 폼의 `submit` 이벤트를 수정해야 합니다. 예를 들어 이렇게 변경합니다. `<form @submit.prevent="calculate">`

마지막으로 계산기를 실행해 봅시다.

## 계산된 속성

Vue.js의 인라인 표현식은 매우 편리합니다. 하지만 조금 더 복잡한 로직이 필요해지면 반드시 계산된 속성을 사용해야합니다. 계산된 속성이란 다른 변수의 변화에 따라 값이 바뀌는 변수입니다. 계산된 속성은 함수처럼 작동하고 객체의 속성처럼 사용할 수 있습니다. 그러나 상당히 큰 차이점이 있습니다. 계산된 속성의 종속되는 속성이 변경될 때마다 계산된 속성의 값은 다시 계산됩니다.

Vue.js에서는 **Vue** 인스턴스 내부의 `computed` 객체 내에서 계산된 속성을 정의 합니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Hello Vue</title>
5 </head>
6 <body>
7 <div class="container">
8   a={{ a }}, b={{ b }}
9   <pre>
10    {{ $data }}
11   </pre>
12 </div>
13 </body>
14 <script src="https://cdn.jsdelivr.net/npm/vue@2.3.4/dist/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: '.container',
18   data: {
19     a: 1,
20   },
21   computed: {
22     // 계산된 getter
23     b: function () {
24       // ** 'this' **는 Vue 인스턴스입니다
25       return this.a + 1
26     }
27   }
28 });
29 });
30 </script>
31 </html>
```

두개의 변수를 만들었습니다. **a**는 1로 설정했고 **b**는 계산된 속성으로 함수의 반환된 결과로 설정됩니다. 이 예제에서 **b**는 2가 됩니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Hello Vue</title>
5 </head>
6 <body>
7 <div class="container">
8   a={{ a }}, b={{ b }}
9   <input v-model="a" >
10 
```

```

11 <pre>
12     {{ $data }}
13 </pre>
14 </div>
15 </body>
16 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
17 <script type="text/javascript">
18 new Vue({
19     el: '.container',
20     data: {
21         a: 1,
22     },
23     computed: {
24         // 계산된 getter
25         b: function () {
26             // ** 'this' **는 Vue 인스턴스입니다
27             return this.a + 1
28         }
29     }
30 });
31 </script>
32 </html>

```

위의 예제는 이전 예제와 동일하지만 한가지 차이점이 있습니다. 변수 **a**가 사용자 입력에 바인딩 되어있습니다. 원하는 결과는 바인딩 된 속성의 값을 변경하고 **b**를 즉시 수정하는 것입니다. 그러나 우리가 기대했던대로 작동하지 않습니다.

**a**의 값을 5로 입력했다면, 아마 **b**가 6이 될거라 예상했을 것 입니다. 하지만 **b**는 51 이 됩니다. 왜 이런일이 일어났을까요? 이미 아는바와 같이, **b**는 **a**의 값을 문자열로 인식합니다. 그래서 숫자 1을 마지막 위치에 추가합니다.

한가지 방법은 **parseFloat()** 함수를 이용하여 문자열을 부동 소수점 숫자로 변경하여 사용하는 것 입니다.

```

new Vue({
    el: '.container',
    data: {
        a: 1,
    },
    computed: {
        b: function () {
            return parseFloat(this.a) + 1
        }
    }
});

```

또 다른 방법으로는 <input type="number">을 사용하는 것 입니다. 항상 숫자를 사용하게 됩니다.

그러나 이보다 더 좋은 방법을 Vue.js가 제공하고 있습니다. 무엇을 입력하든 숫자로 만들어 줄 수 있습니다. 특별한 수식어인 .number를 붙여주면 됩니다.

```
<body>
<div class="container">
  a={{ a }}, b={{ b }}
  <input v-model.number="a" />
  <pre>
    {{ $data }}
  </pre>
</div>
</body>
```

별 다른 노력이 필요없이 number 수식어를 이용하면 원하는 결과를 얻을 수 있습니다.

계산된 속성의 더 큰 그림을 보여주기 위해 이를 이용해 계산기를 만들어 보겠습니다. 이미 계산기를 만들어 보았지만 이번에는 메소드 대신 계산된 속성을 사용하겠습니다.

간단한 예제로 시작하겠습니다. 계산된 속성 c에는 a와 b의 합을 저장합니다.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel\
4 = "stylesheet" />
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Enter 2 numbers to calculate their sum.</h1>
10    <form class="form-inline">
11      <input v-model.number="a" class="form-control" />
12      +
13      <input v-model.number="b" class="form-control" />
14    </form>
15    <h2>Result: {{a}} + {{b}} = {{c}}</h2>
16    <pre>{{ $data }}</pre>
17  </div>
18 </body>
19 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
20 <script type="text/javascript">
21 new Vue({
22   el: '.container',
23   data: {
```

```
24     a: 1,
25     b: 2
26   },
27   computed: {
28     c: function () {
29       return this.a + this.b
30     }
31   }
32 });
33 </script>
34 </html>
```

기초적인 코드를 준비했습니다. 이 시점에서 사용자가 2개의 숫자를 입력하면 그 합을 얻을 수 있습니다. 목표는 사칙연산을 할 수 있는 계산기입니다. 계속 만들어 보겠습니다.

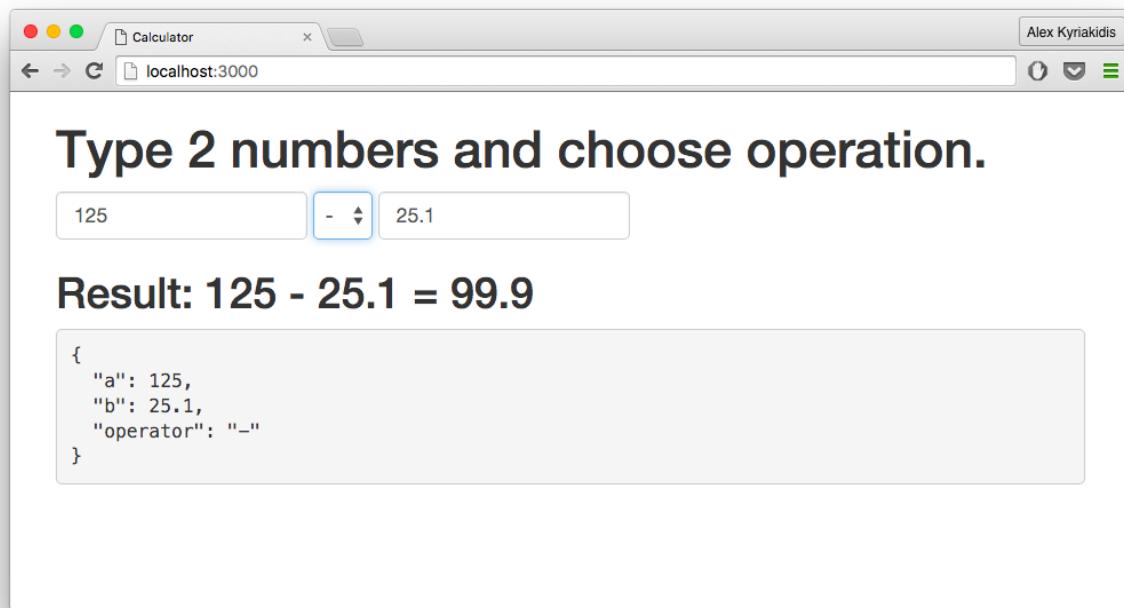
HTML 코드는 [이전 장에서 작성한 계산기](#)와 같으므로 (지금은 버튼이 필요 없습니다.) 자바스크립트 블럭만 보시면 됩니다.

```
1 new Vue({
2   el: '.container',
3   data: {
4     a: 1,
5     b: 2,
6     operator: '+',
7   },
8   computed: {
9     c: function () {
10       switch (this.operator) {
11         case '+':
12           return this.a + this.b
13           break;
14         case '-':
15           return this.a - this.b
16           break;
17         case '*':
18           return this.a * this.b
19           break;
20         case '/':
21           return this.a / this.b
22           break;
23       }
24     }
25   },
26 });
```

계산기를 사용할 준비가 끝났습니다. 할 일은 `calculate` 메소드 안에 있는 것이 무엇이든 계산된 속성 `c`로 옮기는 것 밖에 없습니다! `a`와 `b`가 어떤 값으로 변경되든 실시간으로 결과가 바뀝니다. 버튼이나 이벤트 그 어떤 것도 필요없습니다. 이 얼마나 대단한 일인가요?

## 노트

일반적인 접근 방법은 나누기에 대한 오류를 피하기 위해 `if` 문을 사용하는 것입니다. 그러나 이미 이러한 종류의 결함에 대한 예측을 하고 있습니다. 사용자가 1/0을 입력하면 결과는 자동으로 무한대가 됩니다! 사용자가 텍스트를 입력하면 표시된 결과는 “숫자가 아닙니다.”



계산된 속성을 이용한 계산기 만들기



## 예제 코드

이 장의 예제 코드는 [GitHub](#)<sup>32</sup>에 있습니다.

<sup>32</sup><https://github.com/hoootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter5>

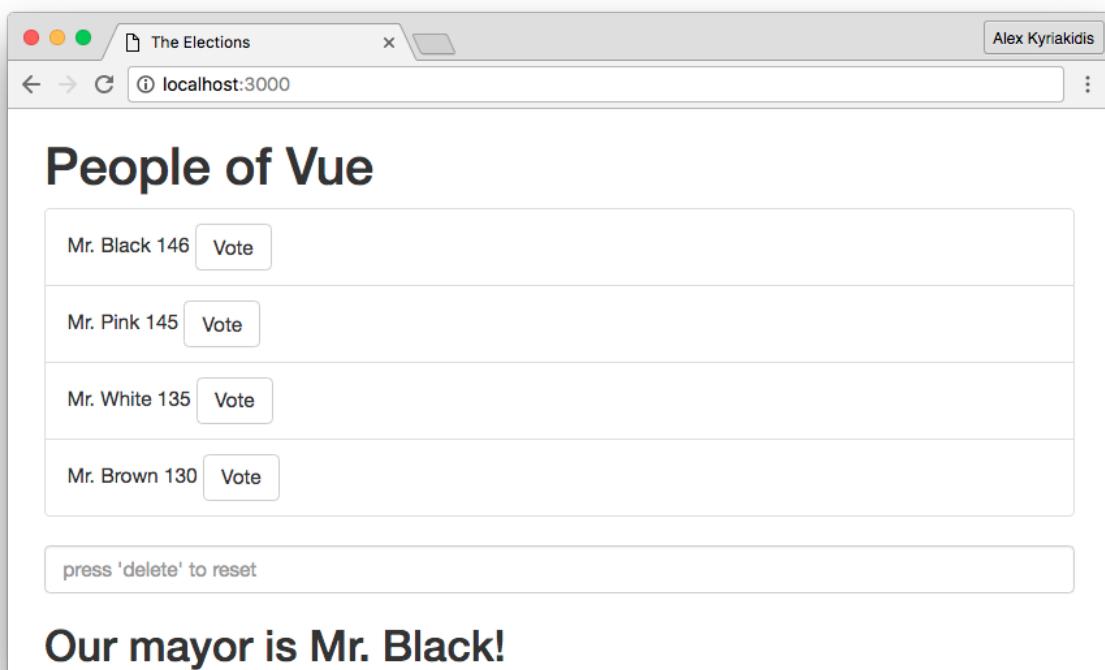
## 혼자 해보기

이제 Vue의 이벤트 처리, 메소드, 계산된 속성 등에 대한 기본적인 이해를 하게 되었으므로 조금 더 도전해보겠습니다. “시장” 후보들의 배열을 만드는 것부터 시작하겠습니다. 각 후보자는 “이름(name)”과 “투표수(votes)”를 가지고 있습니다. 버튼을 사용하여 각 후보자의 득표수를 추가합니다. 계산된 속성을 사용하여 누가 현재 “시장”이 될 수 있는지 결정하고 이름을 표시하세요.

마지막으로 사용자 입력을 추가하세요. 사용자 입력이 포커스 되고 ‘delete’ 키가 눌리면 선거는 처음부터 시작됩니다. 이 말은 모든 투표수를 0으로 만드는 것입니다.

### 힌트

자바스크립트의 내장 함수인 `sort()`와 `map()` 메소드는 매우 유용할 것입니다. 그리고 키 수식어를 사용하면 완성하는데 도움이 될 것입니다.



결과 예시



## 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>33</sup>](#)를 참조하세요.

---

<sup>33</sup><https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter5.html>

# 필터

이전 두개의 장에서 리스트 렌더링, 메소드 및 계산된 속성을 알아보았습니다. 이제 배운 것을 이용해서 몇가지 예제를 만들어 봐도 좋습니다. 이번 장에서는 아래 내용을 알아봅니다.

1. 배열의 항목 필터링
2. 배열의 항목 정렬
3. 사용자 정의 필터 적용
4. 유틸리티 라이브러리 사용

이번 계획은 유사한 예제들을 통해 배웠던 기능의 일부 또는 전부를 결합하는 것입니다.

## 필터링된 결과

원래 데이터를 변경하거나 재설정하지 않고 필터링 된 배열을 표시해야하는 경우가 있습니다. 이전 예제인, [객체 배열을 통한 반복](#)에 이어서 Alex와 John이 쓴 이야기들만 보여주고 싶습니다. 배열을 걸러내어 렌더링 된 결과를 리턴하는 메소드를 만들어 해결해야 합니다.



### 정보

Vue 2.0 부터 **v-for** 안에서 필터를 사용할 수 없습니다. 필터는 오직 텍스트 표현식 (`{} {}`)에서만 사용할 수 있습니다. Vue 팀은 필터 로직을 자바스크립트로 옮겨 컴포넌트 전체에서 재사용하는 것을 권장합니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>User Stories</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Let's hear some stories!</h1>
9         <div>
10            <h3>Alex's stories</h3>
11            <ul class="list-group">
12                <li v-for="story in storiesBy('Alex')"
```

```
14          class= "list-group-item"
15      >
16          {{ story.writer }} said "{{ story.plot }}"
17      </li>
18  </ul>
19  <h3>John ' s stories</h3>
20  <ul class= "list-group" >
21      <li v-for= "story in storiesBy( ' John ' )"
22          class= "list-group-item"
23      >
24          {{ story.writer }} said "{{ story.plot }}"
25      </li>
26  </ul>
27  </div>
28  <pre>
29      {{ $data }}
30  </pre>
31  </div>
32 </body>
33 <script src= "https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js" ></script>
34 <script type= "text/javascript" >
35 new Vue({
36     el: '.container',
37     data: {
38         stories: [
39             {
40                 plot: "I crashed my car today!",
41                 writer: "Alex"
42             },
43             {
44                 plot: "Yesterday, someone stole my bag!",
45                 writer: "John"
46             },
47             {
48                 plot: "Someone ate my chocolate...",
49                 writer: "John"
50             },
51             {
52                 plot: "I ate someone ' s chocolate!",
53                 writer: "Alex"
54             },
55         ],
56     },
57     methods:
58     {
```

```
59      // 작가에 따라 이야기를 필터링하는 메소드
60      storiesBy: function (writer) {
61          return this.stories.filter(function (story) {
62              return story.writer === writer
63          })
64      },
65  }
66  })
67 </script>
68 </html>
```



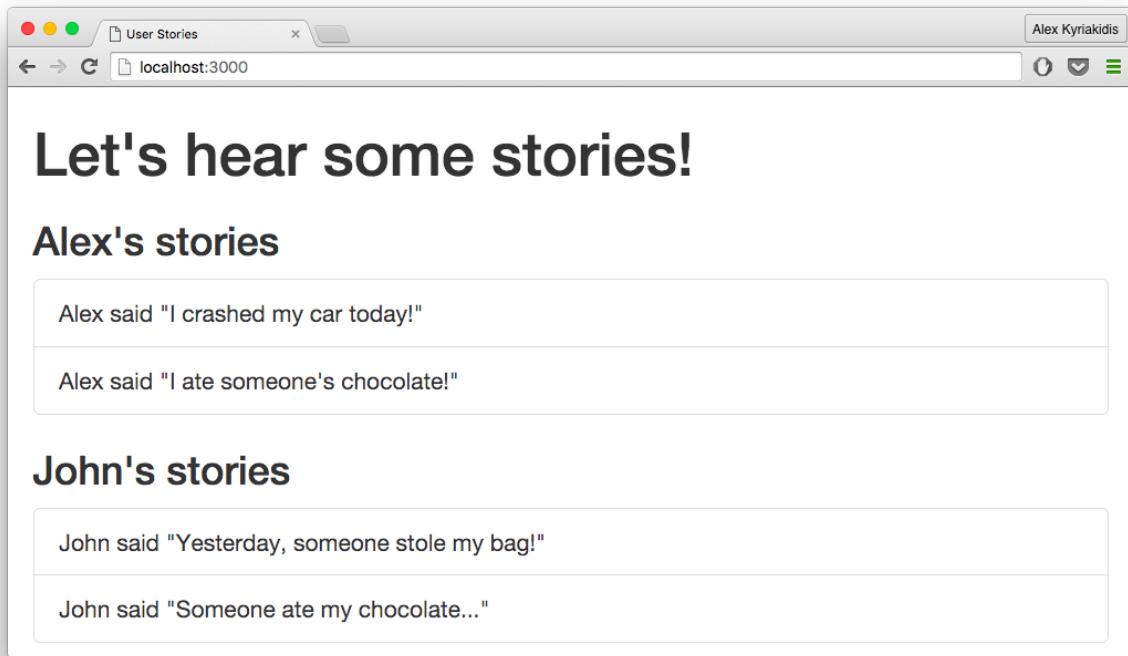
## 정보

storiesBy 메소드에서 자바스크립트에 내장된 `filter` 함수를 사용합니다.<sup>34</sup> `filter()` 함수는 구현된 조건에 맞는 모든 엘리먼트를 가진 새로운 배열을 생성합니다.

위 코드에서 `storiesBy`라는 메소드를 만들었습니다. 이 메소드는 전달인자 `writer`를 가지고 해당 작가로 필터링 된 배열을 반환합니다. 위의 예제처럼 `story in storiesBy( ' Alex ' )` 형식으로 `v-for` 디렉티브를 사용하여 각 작가의 이야기를 표시할 수 있습니다.

---

<sup>34</sup> [https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global\\_Objects/Array/filter](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)



작가로 이야기 필터



## 노트

눈치 채신 것 처럼 **li** 태그 내부는 점점 커지고 있습니다. 그래서 더 많은 줄로 나누었습니다. 실제 결과는 Vue 1.x의 필터 사용법과 동일합니다.

간단하지 않나요?

## 계산된 속성 사용

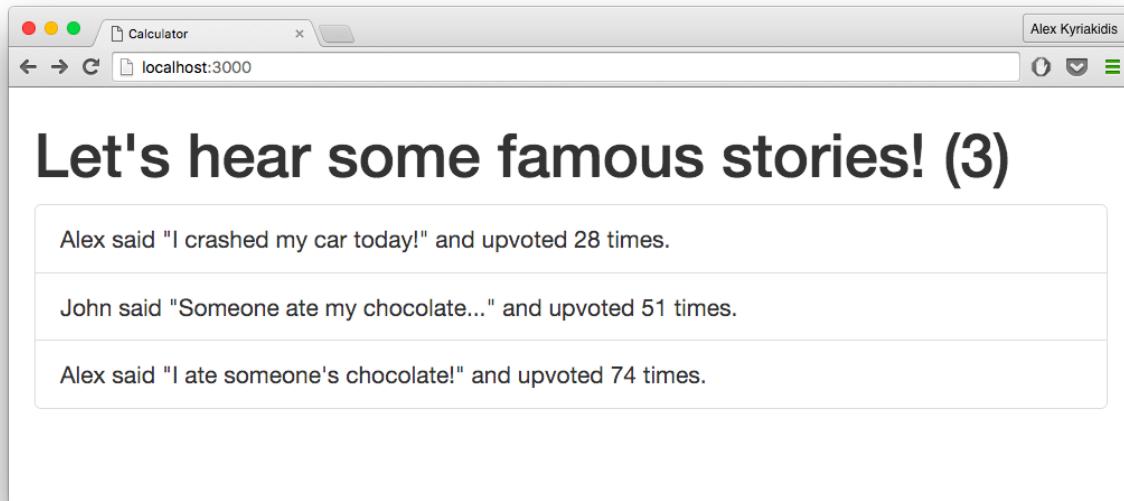
계산된 속성을 사용하여 배열을 필터링 할 수 있습니다. 그러면 다른 곳에서 필터링 된 결과에 접근할 수 있으므로 깊이 있는 제어와 유연함을 제공합니다. 예를 들어 필터링 된 배열의 길이는 코드의 어느 위치에서나 가져올 수 있습니다.

먼저 upvote라는 새 속성으로 이야기에 새로운 기능을 추가합니다. 그 다음 **좋아하는(famous)** 이야기를 필터링 합니다. **famous** 처럼 25개 이상의 추천을 받은 stories를 정의할 수 있습니다. 이번에는 필터링된 배열을 반환하는 대신 계산된 속성을 만듭니다.

```
new Vue({
  el: '.container',
  data: {
    stories: [
      {
        plot: "I crashed my car today!",
        writer: "Alex",
        upvotes: 28
      },
      {
        plot: "Yesterday, someone stole my bag!",
        writer: "John",
        upvotes: 8
      },
      {
        plot: "Someone ate my chocolate...",
        writer: "John",
        upvotes: 51
      },
      {
        plot: "I ate someone's chocolate!",
        writer: "Alex",
        upvotes: 74
      },
    ],
  },
  computed: {
    famous: function() {
      return this.stories.filter(function(item){
        return item.upvotes > 25;
      });
    }
  }
})
```

HTML 코드에서는 **stories** 배열 대신 계산된 속성 **famous**를 사용합니다.

```
<body>
  <div class= " container " >
    <h1>Let ' s hear some famous stories! ({{famous.length}})</h1>
    <ul class= " list-group " >
      <li v-for= " story in famous "
          class= " list-group-item "
        >
          {{ story.writer }} said " {{ story.plot }} "
          and upvoted {{ story.upvotes }} times.
        </li>
    </ul>
  </div>
</body>
```



### 계산된 속성을 이용한 배열 필터링

이게 끝입니다. 계산된 속성을 이용해서 배열을 필터링 했습니다. 이야기의 투표수를 관리하기가 얼마나 쉬운지 확인하셨나요? 이제 헤딩 메시지를 {{famous.length}}를 이용해 출력하겠습니다.

다음으로 아주 기본적이지만 멋진 검색 기능을 구현하겠습니다. 사용자가 이야기의 일부를 입력하면 실시간으로 어떤 이야기인지, 누가 썼는지 알 수 있도록 합니다. 비어있는 값을 가지는 **query**에 바인딩 된 **input**을 추가하여 이야기 배열을 동적으로 필터링 할 수 있습니다.

```

1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     ...
5     <div class="form-group">
6       <label for="query">
7         What are you looking for?
8       </label>
9       <input v-model="query" class="form-control" />
10      </div>
11      <h3>Search results:</h3>
12      <ul class="list-group">
13        <li v-for="story in search"
14          class="list-group-item">
15          >
16            {{ story.writer }} said "{{ story.plot }}"
17          </li>
18        </ul>
19      </div>
20    </div>

```

그런 다음 이름이 `search`인 계산된 속성을 만듭니다. 내장된 `filter` 함수와 함께 하나의 문자열이 다른 문자열에 있는지 확인할 수 있는 자바스크립트 내장 함수 `includes35`를 사용할 것입니다.

```

1 new Vue({
2   el: '.container',
3   data: {
4     stories: [...],
5     query: '',
6   },
7   methods: {
8     storiesBy: function (writer) {
9       return this.stories.filter(function (story) {
10         return story.writer === writer
11       })
12     }
13   },
14   computed: {
15     search: function () {
16       var query = this.query
17       return this.stories.filter(function (story) {
18         return story.plot.includes(query)

```

---

<sup>35</sup> [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/String/includes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes)

```
19      })
20    }
21  }
22 })
```

The screenshot shows a web application window titled "User Stories" with the URL "localhost:3000". The title bar includes the name "Alex Kyriakidis". The main content area displays three sections: "Alex's stories", "John's stories", and a search bar.

- Alex's stories:** Contains two items:
  - Alex said "I crashed my car today!"
  - Alex said "I ate someone's chocolate!"
- John's stories:** Contains two items:
  - John said "Yesterday, someone stole my bag!"
  - John said "Someone ate my chocolate..."
- What are you looking for?**: A search input field with a placeholder character "|".

Below the search results, there is a section labeled "Search results:" which lists the same four items from the "Alex's stories" and "John's stories" sections.

이야기 찾기

The screenshot shows a web browser window titled "User Stories" with the URL "localhost:3000". The page displays a search interface for user stories. At the top, it says "Lets hear some stories!" followed by "Alex's stories" and two story snippets:

- Alex said "I crashed my car today!"
- Alex said "I ate someone's chocolate!"

Below that is "John's stories" with two more snippets:

- John said "Yesterday, someone stole my bag!"
- John said "Someone ate my chocolate..."

Underneath, there is a search bar with the placeholder "What are you looking for?" containing the text "choco". The search results section shows:

Search results:

- John said "Someone ate my chocolate..."
- Alex said "I ate someone's chocolate!"

‘choco’ 찾기

멋지지 않나요?

## 결과 정렬

경우에 따라, 특정 기준에 따라 정렬한 배열을 표시 해야합니다. 계산된 속성을 사용하여 각 이야기의 upvotes 수에 따라 정렬된 배열을 표시할 수 있습니다. 배열을 정렬하기 위해 자바스크립트 내장 함수인 `sort`<sup>36</sup>를 사용합니다. 이 함수는 정렬된 배열을 반환합니다. 좋아하는 이야기일수록 위에 있어야 합니다.

```

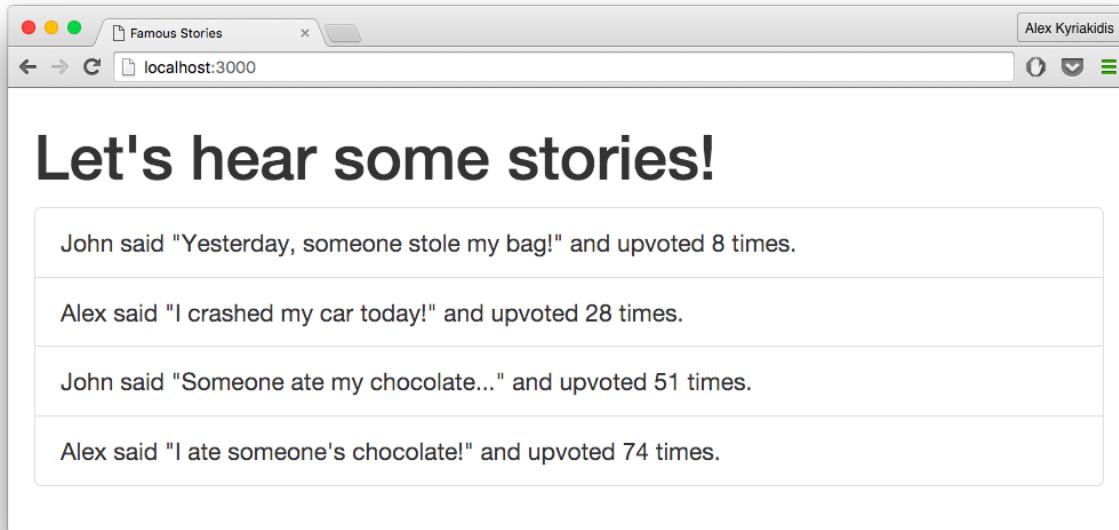
1 <html>
2 <head>
3 <link href=" https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css " rel=" st\ 
4 ylesheet " >
5     <title>Famous Stories</title>
6 </head>
7 <body>
8     <div class= " container " >
9         <h1>Let ' s hear some stories!</h1>
10        <ul class= " list-group " >
11            <li v-for= " story in orderedStories "
12                class= " list-group-item "
13            >
14                {{ story.writer }} said " {{ story.plot }} "
15                and upvoted {{ story.upvotes }} times.
16            </li>
17        </ul>
18        <pre>
19            {{ $data }}
20        </pre>
21    </div>
22 </body>
23 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
24 <script type= " text/javascript " >
25 new Vue({
26     el: '.container',
27     data: {
28         stories: [...]
29     },
30     computed: {
31         orderedStories: function () {
32             return this.stories.sort(function(a, b){
33                 return a.upvotes - b.upvotes;
34             })
35         }
36     }
37 }

```

---

<sup>36</sup>[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/sort](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort)

```
36      }
37  })
38 </script>
39 </html>
```



### 투표 수에 따른 이야기 정렬

음.. 이상하게도 배열이 정렬되긴 했지만 기대한 것과는 다릅니다. 인기 있는 이야기가 처음에 나와야 합니다.

정렬된 배열의 순서를 변경하기 위해 `sort` 함수를 살펴봅시다. 자바스크립트의 `sort(compareFunction)`에서 `compareFunction`이 있으면 `compareFunction`의 반환 값에 따라 정렬됩니다. `a`와 `b`가 비교되는데 사용하는 두 엘리먼트라면,

- `compareFunction(a, b)`가 0 보다 작으면 `a`가 `b` 보다 인덱스가 낮습니다.
- `compareFunction(a, b)`가 0 이면 `a`와 `b`는 변경되지 않습니다.
- `compareFunction(a, b)`가 0 보다 크면 `b`가 `a` 보다 인덱스가 낮습니다.

`compareFunction`은 다음과 같아야 합니다.

compareFunction

---

```
function(a, b){
    return a.upvotes - b.upvotes;
}
```

---

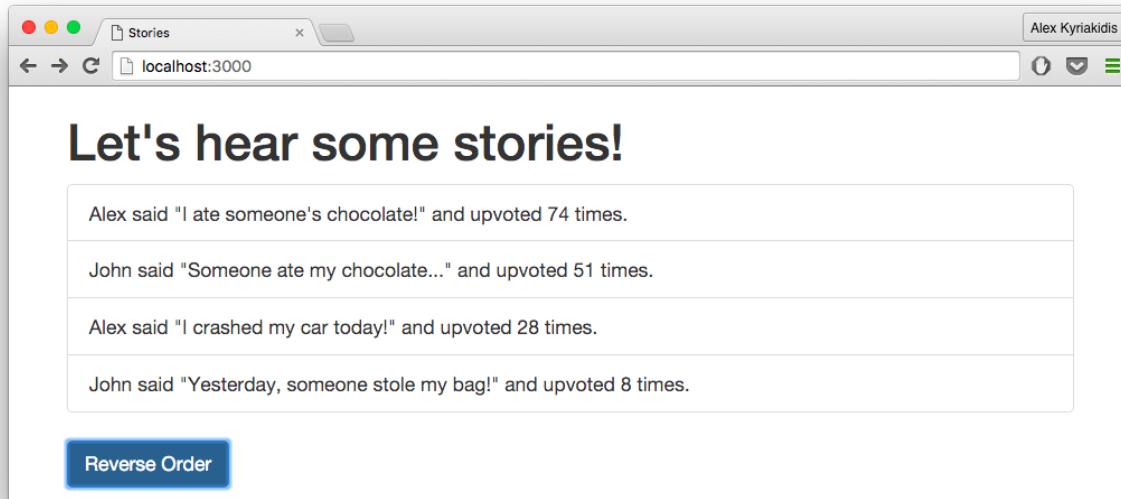
따라서 오름차순에서 내림차순으로 순서를 변경하려면 **-1**을 곱하여 반환 값을 뒤집을 수 있습니다. (**return (a.upvotes - b.upvotes) \* -1**)

변수 **order**를 사용하여 정렬을 동적으로 변경할 수 있습니다. 새로운 변수의 값을 **-1**과 **1**로 토글하는 **button**을 사용하면 됩니다.

```
1 <div class= "container" >
2     <h1>Let ' s hear some stories!</h1>
3     <ul class= "list-group" >
4         <li v-for= "story in orderedStories "
5             class= "list-group-item"
6         >
7             {{ story.writer }} said " {{ story.plot }} "
8             and upvoted {{ story.upvotes }} times.
9         </li>
10    </ul>
11    <button @click= " order = order * -1 " >Reverse Order</button>
12 <pre>
13     {{ $data }}
14 </pre>
15 </div>
```

```
1 new Vue({
2     el: '.container',
3     data: {
4         stories: [...],
5         order : -1
6     },
7     computed: {
8         orderedStories: function () {
9             var order = this.order;
10            return this.stories.sort(function(a, b) {
11                return (a.upvotes - b.upvotes) * order;
12            })
13        }
14    }
15 })
```

**order** 변수를 **-1** 값으로 초기화 한 다음 계산된 속성에 전달하므로 버튼을 클릭할 때마다 변수의 값이 변경되고 배열의 순서가 변경됩니다.



내림차순 정렬

## 사용자 정의 필터

사용자 정의 필터를 사용하기 위해 새로운 예를 만들겠습니다. 고담시의 신문인 “고담 신문”을 담당하고 있다고 가정해봅시다. 첫 업무는 영웅들의 숨겨진 정체성을 알리는 것 입니다. 이미 그들의 성과 이름을 알고 있습니다. 그리고 숨겨진 신원을 드러낼 멋진 목록을 만들고 싶습니다. 전역 `Vue.filter()`를 이용해 `hero`를 가지고 모든 정보를 가져올 필터를 만듭니다. 물론 HTML 코드를 더럽히면 안됩니다. 필터를 등록하려면 `filterID`와 `filterFunction` 전달하면 됩니다. 필터는 처리된 값을 반환합니다.

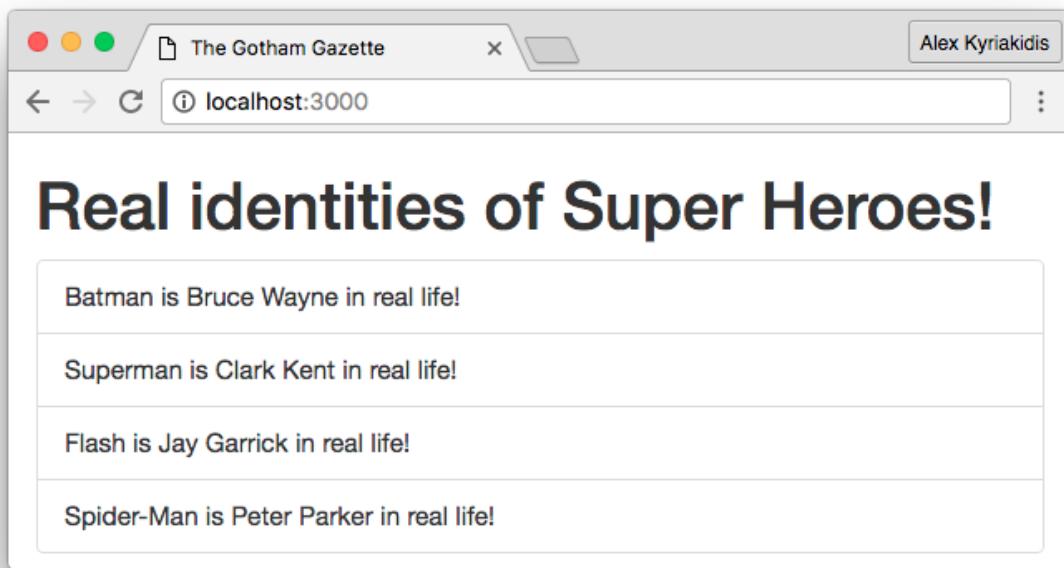
그런 다음 필터를 텍스트 표현식 내부에서 사용합니다.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>The Gotham Gazette</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Real identities of Super Heroes!</h1>
9         <ul class="list-group">
10            <li v-for="hero in heroes"
11                class="list-group-item">
12                >
13                {{ hero | snitch }}
14            </li>
15        </ul>
16    </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js">
19 </script>
20 <script>
21     Vue.filter('snitch', function (hero) {
22         return hero.secretId + ' is '
23         + hero.firstname + ', '
24         + hero.lastname + ' in real life!'
25     })
26 }
27
28 new Vue({
29     el: '.container',
30     data: {
31         heroes: [
32             { firstname: 'Bruce', lastname: 'Wayne', secretId: 'Batman' },
33             { firstname: 'Clark', lastname: 'Kent', secretId: 'Superman' },
34         ]
35     }
36 })
```

```

34      { firstname: 'Jay', lastname: 'Garrick', secretId: 'Flash' },
35      { firstname: 'Peter', lastname: 'Parker', secretId: 'Spider-Man' }
36  ]
37 }
38 })
39 </script>
40 </html>

```



사용자 정의 필터 'famous'

## 유틸리티 라이브러리

이 시점에서 조금 더 고급의 방법으로 데이터를 정렬/필터/색인 하기 위해 자바스크립트 유틸리티 라이브러리 사용을 고려해 보겠습니다. [Lodash<sup>37</sup>](#)와 [Underscore<sup>38</sup>](#), 그리고 [Sugar<sup>39</sup>](#) 등의 훌륭한 라이브러리가 이미 준비되어 있습니다.

더 나은 이해를 위해 Lodash를 추가하고 이전의 예제를 수정하겠습니다.

Lodash 사용하기 위해 CDN을 통해 **HTML** 파일에 추가해야합니다.

<sup>37</sup><https://lodash.com>

<sup>38</sup><http://underscorejs.org/>

<sup>39</sup><https://sugarjs.com/>

Lodash의 **orderBy** 메소드는 새로운 정렬된 배열을 반환합니다. 따라서 Stories 배열을 정렬하기 위해 계산된 속성 내에서 이 메소드를 사용합니다.

## 문법

Lodash의 **orderBy** 문법입니다:

```
_.orderBy(collection, [iteratees=[_.identity]], [orders])
```

두번째 전달인자가 혼동을 줄 수 있습니다. 정말 간단합니다. 첫번째 전달인자는 정렬할 배열입니다. 두번째 전달인자는 정렬을 위한 배열의 키입니다. 세번째 전달인자는 각 키에 대한 orders 배열입니다.

배열이 있는 경우에 대한 예제:

```
var kids = [
  { name: 'Stan', strength: 70, intelligence: 70},
  { name: 'Kyle', strength: 40, intelligence: 80},
  { name: 'Eric', strength: 45, intelligence: 80},
  { name: 'Kenny', strength: 100, intelligence: 70}
]
```

이제 실행해보면,

```
_.orderBy(kids, [ 'intelligence', 'strength' ], [ 'desc', 'asc' ])
```

배열의 내용은 이렇게 정렬 됩니다.

```
var kids = [
  { name: 'Kyle', strength: 40, intelligence: 80},
  { name: 'Eric', strength: 45, intelligence: 80},
  { name: 'Stan', strength: 70, intelligence: 70},
  { name: 'Kenny', strength: 100, intelligence: 70}
]
```

배열은 아이의 **내림차순인** 지능을 기준으로 정렬 됩니다. 그리고 아이의 **오름차순인** 힘을 두번째 기준으로 사용합니다.

다음과 같이 계산된 속성에서 **\_.orderBy**를 사용합니다.

```
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes')
  }
}
```

잘 작동하지만 orders 전달인자가 없으면 배열은 오름차순으로 정렬 됩니다. 또한 이전과 마찬가지로 버튼을 사용하여 배열의 순서를 변경할 수 있어야 합니다. 이를 동적으로 수행하기 위해 `order : 'desc'` 데이터 속성을 설정하고 이 값을 변경하는 메소드를 만들 수 있습니다.

```
methods: {
  reverseOrder: function () {
    this.order = (this.order === 'desc') ? 'asc' : 'desc'
  }
},
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes', [order])
  }
}
```

버튼은 거의 같지만 이번에는 매우 적은 코드입니다.

```
<button v-on:click="reverseOrder">
```

다 되었습니다. Lodash를 이용해 똑같이 만들었습니다.

## 팁

유틸리티 라이브러리를 사용하여 데이터를 필터/정렬할 때 계산된 속성을 사용하지 않고 결과 배열을 반복할 수 있습니다.

이제 예제를 수정하여 아이디어를 얻을 수 있습니다. 정렬된 배열을 렌더링하는 HTML 코드는 다음과 같습니다.

```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="story in _.orderBy(stories, [ 'upvotes' ], [ 'desc' ]) ">
      {{ story.writer }} said "{{ story.plot }}"
      and upvoted {{ story.upvotes }} times.
    </li>
  </ul>
</div>
```

끝입니다. 어떠한 자바스크립트 코드도 필요 없습니다.



## 예제 코드

이 장의 예제 코드는 [GitHub<sup>40</sup>](#)에 있습니다.

---

<sup>40</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter6>

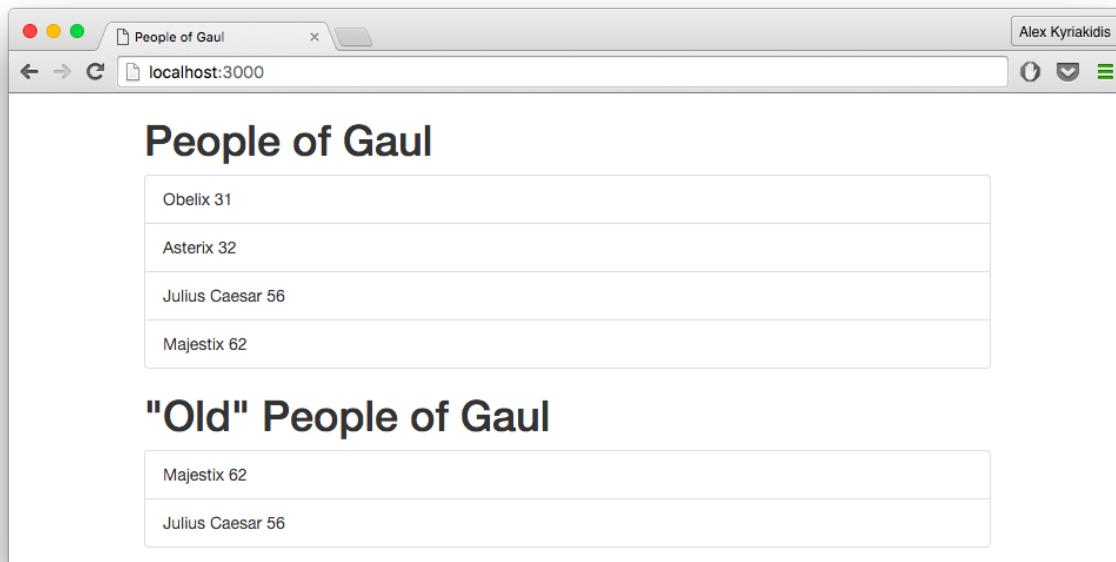
## 혼자 해보기

이 장에서 배운 것을 연습하기 위해 아래 내용을 확인해야합니다. 사람들을 가지고 있는 배열을 만든 후 시작하겠습니다. 각 사람은 이름과 나이를 가지고 있습니다. 지금까지 배운 것을 사용하여 배열을 출력하고 “나이(age)”별로 정렬 해봅니다. 그 다음에 계산된 속성인 “old”를 가지고 65세 이상의 사람들만 출력하는 두번째 리스트를 만듭니다.

가볍게 주변 사람들로 배열을 채우세요. 주의하세요. 필터가 제대로 작동하는지 확인하려면 65세 이상인 사람들을 추가해야 합니다! 시작하세요!

### 💡 힌트

자바스크립트에 내장된 `.filter`가 여기 필요합니다.



결과 예시



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>41</sup>](#)를 참조하세요.

<sup>41</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter6.html>

# 컴포넌트

## 컴포넌트가 무엇인가요?

컴포넌트는 Vue.js에서 가장 강력한 기능 중 하나입니다. 기본 HTML 엘리먼트를 확장하여 재사용할 수 있도록 캡슐화하는데 도움을 줍니다. 상위 레벨에서 컴포넌트는 Vue.js의 컴파일러가 지정된 동작을 추가할 사용자 정의 엘리먼트입니다. 어떤 경우에는 특별한 **is** 속성으로 확장한 기본 HTML 엘리먼트로 나타낼 수 있습니다.

새로운 기능을 위해 HTML을 확장하는 것은 정말 강력한 방법입니다. 이 장에서는 아주 간단한 예제부터 시작하려고 합니다. 다음 장에서는 컴포넌트가 어떻게 이 장에서 작성한 코드를 개선하는데 도움이 되는지 살펴 보겠습니다.

## 컴포넌트 사용하기

간단한 컴포넌트를 만듭니다. 우선 컴포넌트를 사용하기 위한 등록절차가 필요합니다.

컴포넌트를 등록하는 한가지 방법은 **Vue.component** 메소드를 사용하고, **태그** 및 **생성자**를 전달하는 것입니다. **태그**를 컴포넌트의 이름으로, **생성자**를 옵션으로 생각하시면 됩니다. 이번에는 컴포넌트 이름을 **story**로 짓고, 속성을 정의할 것 입니다. **template** 옵션(**story**)은 **생성자** 내부에 있으며 다른 옵션도 추가할 수 있습니다.

story 컴포넌트는 이와 같이 등록합니다.

```
1 Vue.component('story', {  
2   template: '<h1>My horse is amazing!</h1>'  
3 });
```

이제 컴포넌트를 잘 등록했으므로 사용해보겠습니다. 이야기를 표시하기 위해 HTML안에 사용자 정의 엘리먼트인 **<story>**를 추가합니다.

```
1 <html>
2 <head>
3   <link href= "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel\
4 = "stylesheet" >
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class= "container" >
9     <story></story>
10    </div>
11 </body>
12 <script src= "https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js" ></script>
13 <script type= "text/javascript" >
14 Vue.component( 'story' , {
15   template: ' <h1>My horse is amazing!</h1> '
16 });
17
18 new Vue({
19   el: '.container'
20 })
21 </script>
22 </html>
```

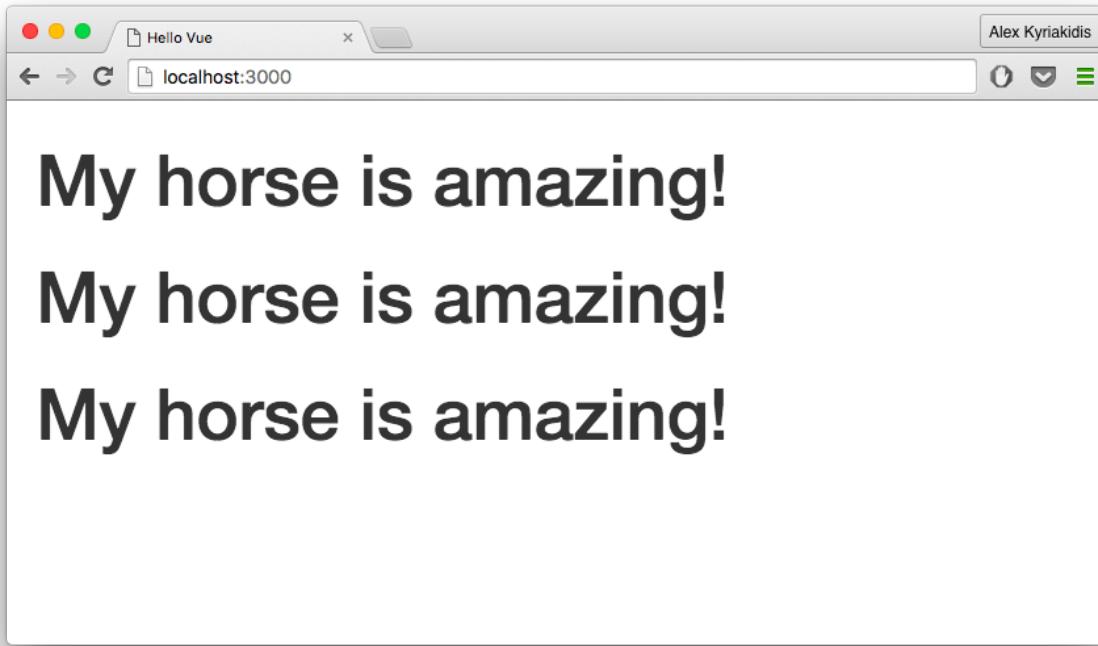


## 노트

여기서 사용자 정의 컴포넌트에 원하는 이름을 지정할 수 있지만 일반적으로 고유한 이름을 사용하여 나중에 추가될 수 있는 실제 HTML 태그와 충돌하지 않도록 예방하는 것이 좋습니다.

이 장의 시작 부분에서 언급했듯이 컴포넌트는 재사용할 수 있습니다. 원하는대로 **<story>** 엘리먼트를 추가할 수 있다는 것을 의미합니다. 다음 HTML 코드 조각은 story 컴포넌트를 3번 표시합니다.

```
<body>
  <div class= "container" >
    <story></story>
    <story></story>
    <story></story>
  </div>
</body>
```



이야기 컴포넌트 출력

## 템플릿

컴포넌트의 템플릿 선언 방법은 여러가지가 있습니다. 이전에 사용한 인라인 템플릿 방식을 사용하면 템플릿 코드가 쉽게 지저분해집니다.

또 다른 방법은 `type= "text/template"` 으로 설정된 `script` 태그를 `story-template`의 `id`로 생성하는 것입니다. 이 템플릿을 사용하려면 이 스크립트에 대한 컴포넌트의 `template`에서 셀렉터를 참조해야 합니다.

```
<script type= "text/template" id= "story-template" >
  <h1>My horse is amazing!</h1>
</script>

<script type= "text/javascript" >
  Vue.component( 'story' , {
    template: "#story-template"
  });
</script>
```



## 정보

”**text/template**” 은 브라우저가 이해할 수 있는 스크립트가 아니므로 브라우저는 이를 무시합니다. 이와 같이 하면 추출할 수 있는 모든 내용을 넣을 수 있고 HTML 코드 조각을 생성할 수 있습니다.

**template** (그리고 이 책의 예제에서 사용할 템플릿)을 정의하는데 있어 저의 추천은 **template** HTML 태그를 만들고 **id**를 지정하는 것입니다. 그런 다음 이전과 마찬가지로 셀렉터를 참조할 수 있습니다. 아래 컴포넌트는 위의 방법을 사용한 예입니다.

```
<template id= "story-template" >
  <h1>My horse is amazing!</h1>
</template>

<script type= "text/javascript" >
  Vue.component( 'story' , {
    template: "#story-template"
  });
</script>
```

## 속성

이제 **story** 컴포넌트의 여러 인스턴스를 사용하여 이야기 목록을 표시하는 방법을 알아보겠습니다. **template**은 항상 같은 이야기를 표시하지 않고 원하는 이야기의 줄거리를 표시해야 합니다.

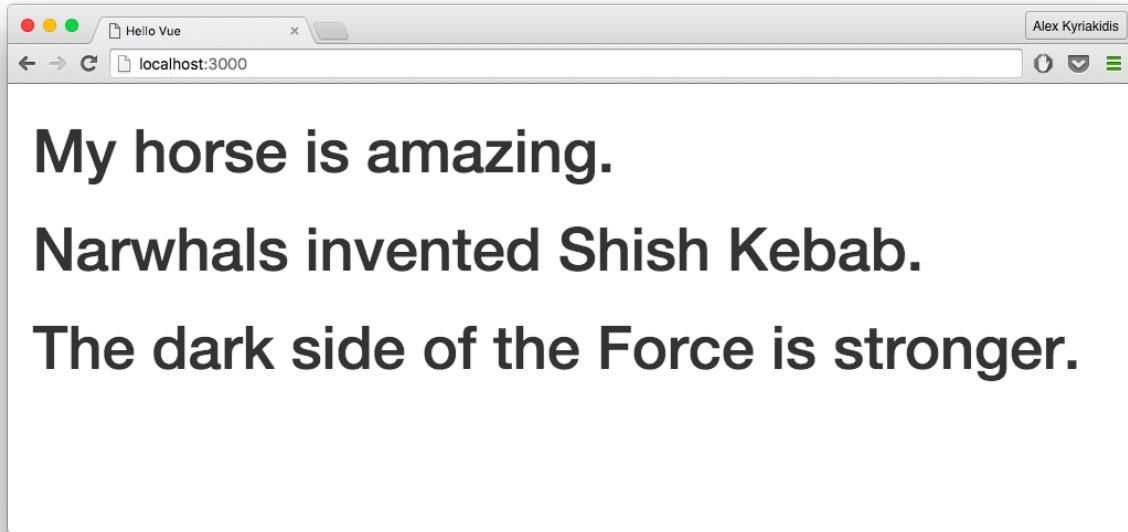
```
<template id= "story-template" >
  <h1>{{ plot }}</h1>
</template>
```

이 속성을 사용하려면 컴포넌트를 수정해야 합니다. 이를 위해 새로운 속성인 ‘plot’을 **props** 속성에 추가할 것입니다.

```
Vue.component( 'story' , {
  props: [ 'plot' ],
  template: "#story-template"
});
```

이제 **<story>** 엘리먼트를 사용할 때마다 **plot**을 전달할 수 있습니다.

```
<div class="container">
  <story plot="My horse is amazing."></story>
  <story plot="Narwhals invented Shish Kebab."></story>
  <story plot="The dark side of the Force is stronger."></story>
</div>
```



서로 다른 이야기들



## 주의사항

HTML 속성은 대소문자를 구분하지 않습니다. camelCase를 사용한 속성을 이름으로 사용하는 경우 이를 kebab-case(하이픈으로 구분됨)로 사용해야 합니다.

그래서, camelCase는 자바스크립트에서, kebab-case는 HTML에서 사용해야 합니다. 예를 들어 `props: [ 'isUser' ]`는 HTML 속성으로 사용할 때 `<story is-user="true"></story>`로 써야 합니다.

이미 예상한 것 처럼 컴포넌트는 둘 이상의 속성을 가질 수 있습니다. 예를 들어, 모든 이야기에 대한 플롯과 함께 작가를 표시하려면 `writer`도 전달해야 합니다.

```
<story plot="My horse is amazing." writer="Mr. Weebly"></story>
```

속성이 많아져 엘리먼트가 더러워지면 객체를 속성으로 전달할 수 있습니다.  
한번 더 예제를 리팩토링하여 마무리 하겠습니다.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4   <title>Awesome Stories</title>
5 </head>
6 <body>
7   <div class="container">
8     <story v-bind:story= " {plot: ' My horse is amazing. ', writer: ' Mr. Weebly ' } " >
9     </story>
10    <story v-bind:story= " {plot: ' Narwhals invented Shish Kebab. ', writer: ' Mr. Weebly ' } " >
11    </story>
12    <story v-bind:story= " {plot: ' The dark side of the Force is stronger. ', writer: ' Darth Vader ' } " >
13    </story>
14  </div>
15  <template id="story-template">
16    <h1>{{ story.writer }} said {{ story.plot }}</h1>
17  </template>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
20 <script type="text/javascript">
21   Vue.component('story', {
22     props: [ 'story' ],
23     template: "#story-template"
24   });
25
26 new Vue({
27   el: '.container'
28 })
29 </script>
30 </html>
```



## 정보

**v-bind**는 하나 이상의 속성 또는 컴포넌트 속성을 표현식에 동적으로 바인딩하는데 사용합니다.

**story** 속성은 문자열이 아니라 자바스크립트 객체이므로 **story= " ... "** 대신 **v-bind:story= " ... "**를 사용하여 전달된 객체에 **story** 속성을 바인딩 합니다.

**v-bind**의 축약형은 :입니다. 그러므로 이제 **:story= " ... "** 와 같이 사용할 수 있습니다.

## 재사용성

**정렬된 결과** 예제를 다시 한번 살펴 보겠습니다. 이번에는 http 호출을 하여 외부 API에서 **stories** 변수 데이터를 가져옵니다. API 개발자는 이야기의 **plot** 속성을 **body**로 바꾸기로 결정했습니다. 이제 코드를 검토하고 수정해야 합니다.

### 정보

이 책의 뒷부분에서는 **Vue**를 사용하여 웹의 HTTP 요청 방법을 설명합니다.

```
1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     <h3>Alex's stories</h3>
5     <ul class="list-group">
6       <li v-for="story in storiesBy('Alex')"
7           class="list-group-item">
8         >
9         {{ story.writer }} said "{{ story.plot }}"
10        {{ story.writer }} said "{{ story.body }}"
11      </li>
12    </ul>
13    <h3>John's stories</h3>
14    <ul class="list-group">
15      <li v-for="story in storiesBy('John')"
16          class="list-group-item">
17        >
18        {{ story.writer }} said "{{ story.plot }}"
19        {{ story.writer }} said "{{ story.body }}"
20      </li>
21    </ul>
22    <div class="form-group">
23      <label for="query">
24        What are you looking for?
25      </label>
26      <input v-model="query" class="form-control" />
27    </div>
28    <h3>Search results:</h3>
29    <ul class="list-group">
30      <li v-for="story in search"
31          class="list-group-item">
32        >
33        {{ story.writer }} said "{{ story.plot }}"
34      </li>
35    </ul>
```

```
34         {{ story.writer }} said "{{ story.body }}"
35     </li>
36   </ul>
37 </div>
38 </div>
```



## 노트

이 특별한 예제의 구문 강조 표시는 끈 상태입니다.

보신대로, 정확하게 같은 코드 변경을 3번이나 작성해야 했습니다. 이 책을 읽는 분의 성향은 모르지만 저는 반복하는 것을 싫어합니다. 이 일이 그다지 크게 불편하게 느껴지지 않는다면, 위의 코드 블록을 100개의 위치에서 사용해야한다고 상상해보십시오. 어떻게 하실건가요? 운 좋게도 Vue 는 이에 대한 해결책을 마련해 두었습니다. 바로 컴포넌트입니다



## 팁

기능을 반복할 때마다 가장 효율적인 방법은 전용 컴포넌트를 만드는 것입니다.

다행히 앞의 예에서 이야기의 작가와 본문을 표시하는 story 컴포넌트를 만들었습니다. HTML 안에 사용자 정의 엘리먼트 <story>를 사용할 수 있으며 :story 태그를 사용하여 이전과 같이 각 이야기를 전달할 수 있습니다. 이번에는 v-for 디렉티브와 함께 사용합니다.

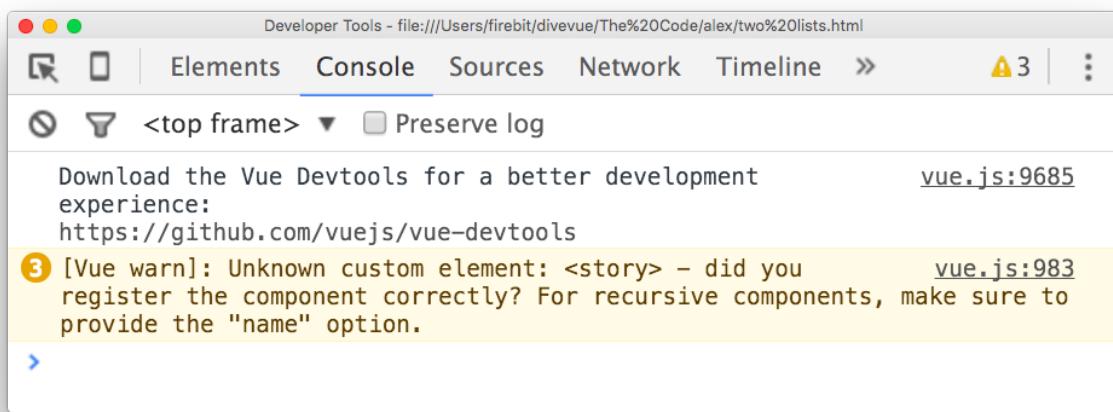
다음 코드를 보세요.

```
<div class="container">
  <h1>Lets hear some stories!</h1>
  <div>
    <h3>Alex's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('Alex')"
             :story="story"></story>
    </ul>
    <h3>John's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('John')"
             :story="story"></story>
    </ul>
    <div class="form-group">
      <label for="query">What are you looking for?</label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Search results:</h3>
```

```
<ul class="list-group">
  <story v-for="story in search"
    :story="story"></story>
</ul>
</div>
</div>
```

이 코드를 실행하려고 하면 다음의 경고 메시지가 표시됩니다.

Vue warn: Unknown custom element: <story> – did you register the component correctly?  
For recursive components, make sure to provide the “name” option.



### Vue 경고

이 문제를 해결을 위해 다시 컴포넌트를 등록해야 합니다. 이번에는 컴포넌트의 템플릿을 약간 변경합니다. 그리고 **plot** 속성을 **body** 및 **<h1>** 태그로 변환하고 필요에 맞게 **<li>**로 바꿉니다. **story** 컴포넌트의 템플릿이 아래에 있습니다.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.body }}"
  </li>
</template>
```

컴포넌트는 동일하게 유지됩니다.

```
1 Vue.component( 'story' , {  
2     props: [ 'story' ],  
3     template: '#story-template'  
4 });
```

위의 사용자 정의 컴포넌트로 만든 코드를 실행하면 모든 것이 이전과 동일합니다.  
괜찮지 않나요?



## 주의사항

책임은 당신에게 있습니다. 너무 해매지 마세요.

## 함께 사용하기

새로 얻은 지식을 사용해 조금 더 복잡한 것을 만들 수 있어야 합니다. 위 예제의 구조에 따라 **이야기들**에 대한 투표 시스템을 만들고 좋아요 기능을 추가해 봅시다. 이를 할 수 있는 방법은 메소드, 디렉티브 그리고 당연히 컴포넌트가 필요합니다.

이제 시작 해보겠습니다.

```
1 <html>  
2   <head>  
3     <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">  
4   </head>  
5   <body>  
6     <div id="app">  
7       <div class="container">  
8         <h1>Let's hear some stories!</h1>  
9         <ul class="list-group">  
10           <story v-for="story in stories" :story="story"></story>  
11         </ul>  
12         <pre>{{ $data }}</pre>  
13       </div>  
14     </div>  
15   </body>  
16 </html>
```

```
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
24 <script type="text/javascript">
25 Vue.component('story', {
26     template: "#story-template",
27     props: [ 'story' ],
28 });
29
30 new Vue({
31     el: '#app',
32     data: {
33         stories: [
34             {
35                 plot: 'My horse is amazing.',
36                 writer: 'Mr. Weebl',
37             },
38             {
39                 plot: 'Narwhals invented Shish Kebab.',
40                 writer: 'Mr. Weebl',
41             },
42             {
43                 plot: 'The dark side of the Force is stronger.',
44                 writer: 'Darth Vader',
45             },
46             {
47                 plot: 'One does not simply walk into Mordor',
48                 writer: 'Boromir',
49             },
50         ]
51     }
52 })
53 </script>
54 </html>
```

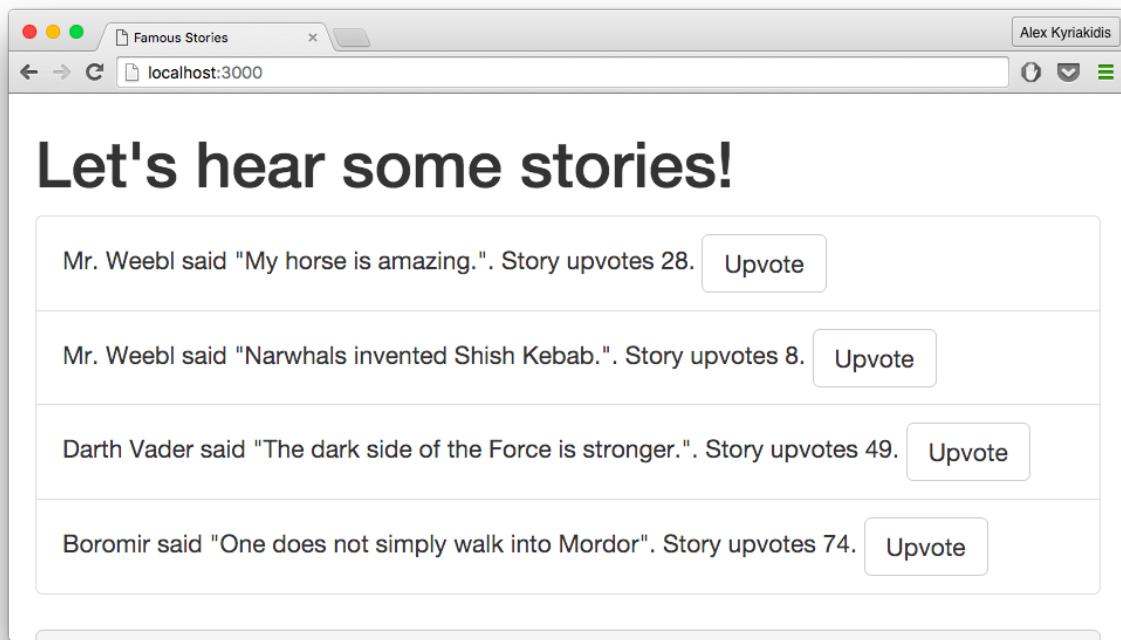
다음 단계로 사용자가 원하는 이야기에 투표할 수 있게 만듭니다. 한 이야기당 1표만 투표할 수 있게 제한하려면 아직 투표하지 않은 사용자에게 ‘투표’ 버튼을 표시해주어야 합니다. 모든 이야기는 **voted** 속성을 가져야 하며 **upvote** 기능이 실행되면 참이 되어야 합니다.

```
<template id= "story-template" >
  <li class= "list-group-item" >
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show= "!story.voted" @click= "upvote"
            class= "btn btn-default" >
      >
        Upvote
    </button>
  </li>
</template>

Vue.component( 'story' , {
  template: "#story-template",
  props: [ 'story' ],
  methods:{
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
  }
});

new Vue({
  el: '#app',
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weebl',
        upvotes: 28,
        voted: false,
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weebl',
        upvotes: 8,
        voted: false,
      },
      {
        plot: 'The dark side of the Force is stronger.',
        writer: 'Darth Vader',
        upvotes: 49,
        voted: false,
      }
    ]
  }
});
```

```
        },
        {
          plot: 'One does not simply walk into Mordor',
          writer: 'Boromir',
          upvotes: 74,
          voted: false,
        },
      ],
    }
  })
}
```



### 투표 준비 끝!

이제 투표 시스템이 만들어졌습니다. 잘 만든 것 같습니다. 이제 '좋아하는 이야기' 부분을 진행하겠습니다. 이번에는 사용자마다 단 하나의 이야기만 좋아하도록 선택할 수 있기를 원합니다. 가장 먼저 할 일은 새로운 빈 객체(favorite)를 추가하고 사용자가 한 이야기를 좋아한다고 선택하면 **favorite** 변수를 업데이트 하는 것입니다. 이렇게 하면 이야기가 사용자가 좋아한다고 표시한 이야기와 같은지 확인할 수 있습니다.

시작합니다.

```
<template id= " story-template " >
  <li class= " list-group-item " >
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show= " !story.voted " @click= " upvote "
      class= " btn btn-default " >
      Upvote
    </button>
    <button v-show= " !isFavorite " @click= " setFavorite "
      class= " btn btn-primary " >
      Favorite
    </button>
    <span v-show= " isFavorite "
      class= " glyphicon glyphicon-star pull-right " aria-hidden= " true " >
    </span>
  </li>
</template>

Vue.component( ' story ' , {
  template: "#story-template",
  props: [ ' story ' ],
  methods: {
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
    setFavorite: function(){
      this.favorite = this.story;
    },
  },
  computed: {
    isFavorite: function(){
      return this.story == this.favorite;
    },
  }
});

new Vue({
  el: '#app',
  data: {
    stories: [
      ...
    ],
    favorite: {}
  }
})
```

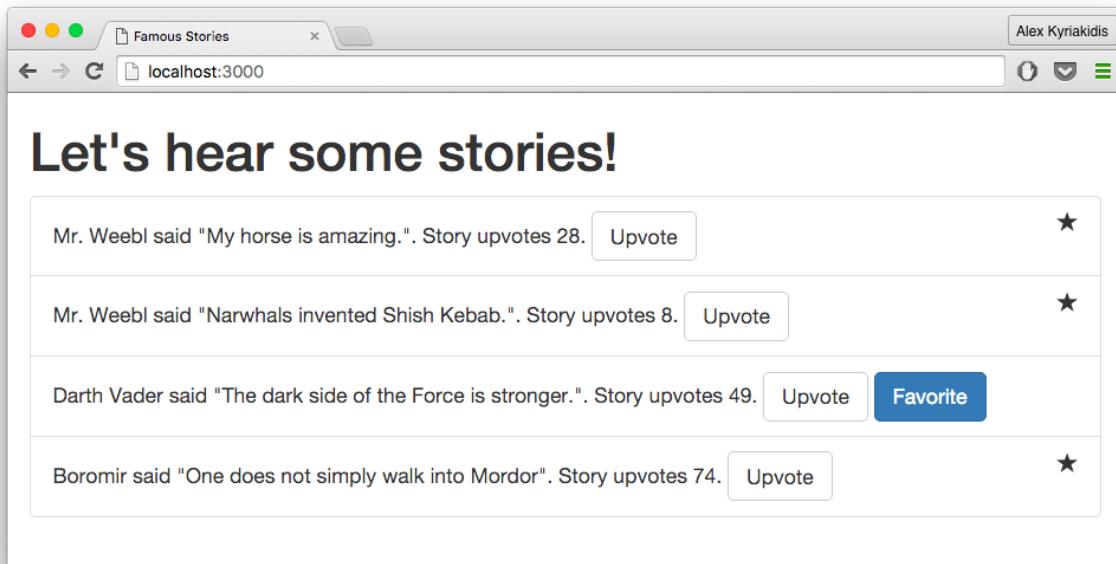
```
    }
})
```

위 코드를 실행하려 하면 코드가 제대로 작동하지 않습니다. 이야기에 좋아요 표시를 할 때마다 `$data` 안에 있는 `favorite` 변수가 null입니다.

`story` 컴포넌트는 `favorite` 객체를 수정할 수 없으므로 각 이야기에 `favorite`를 전달하기 위해 컴포넌트의 속성에 추가해야 합니다.

```
<ul class="list-group">
  <story v-for="story in stories"
         :story="story"
         :favorite="favorite">
  </story>
</ul>
```

```
Vue.component('story', {
  ...
  props: ['story', 'favorite'],
  ...
});
```



`setFavorite` 메소드 오작동

이상하게도 **favorite**는 **setFavorite**가 실행되어도 여전히 바뀌지 않습니다. 버튼이 예상대로 사라지고 별 모양 아이콘이 나타나지만 favorite 변수는 여전히 null입니다. 이 문제로 인해 사용자는 모든 이야기에 좋아요 표시를 할 수 있습니다.

이 것의 문제는 동기화하고 있지 않는 것입니다. 기본적으로 모든 속성은 하위 속성과 부모 사이에 단방향 바인딩을 만듭니다. 부모 속성이 변경되면 자식에게 전달하지만 반대 방향으로는 전달하지 않습니다. 지금까지 알고 있는 것과 다르게 자녀의 데이터를 부모와 동기화 할 수 없습니다.

잠시 쉬었다가 Vue의 사용자 정의 이벤트를 익힌 후 더 많은 것들을 해보겠습니다.



## 예제 코드

이 장의 예제 코드는 [GitHub<sup>42</sup>](#)에 있습니다.

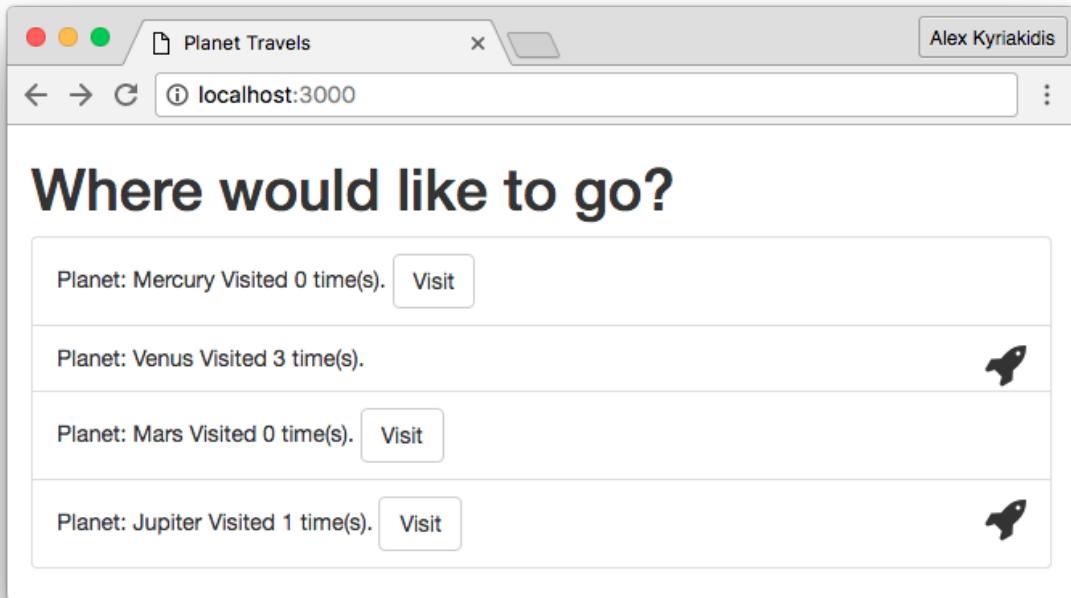
---

<sup>42</sup><https://github.com/hoootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter7>

## 혼자 해보기

행성에 대한 배열을 만들어 보세요. 각각의 행성들은 반드시 이름(name)과 방문 횟수(number of visits)를 가져야 합니다. 우리는 어느 행성이라도 방문할 수 있습니다. 하지만 연료의 제한으로 인해 행성당 3번 밖에 방문할 수 없습니다. 적절한 메소드 또는 계산된 속성을 가진 Planet 컴포넌트를 만들어야 합니다. 렌더링 되면, 각 행성은 다음을 출력합니다.

- 해당 행성의 이름
- 방문 횟수
- 방문 버튼 (최대 방문 횟수에 도달하지 않은 경우에만 출력됨)
- 행성에 적어도 한번이라도 방문했는지 나타내는 아이콘



결과 예시



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>43</sup>](#)를 참조하세요.

<sup>43</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter7.html>

# 사용자 정의 이벤트

Vue로 작업하다 보면 사용자 정의 이벤트가 필요한 경우가 있습니다. 이를 위해 Vue 인스턴스 메소드를 이용합니다. 모든 Vue 인스턴스는 [이벤트 인터페이스<sup>44</sup>](#)를 가지고 있습니다. 이는 다음을 사용할 수 있음을 의미합니다.

- `$on(event)`를 이용하여 이벤트 청취.
- `$emit(event)`를 이용하여 이벤트 발생.

또한,

- 한번만 이벤트를 청취해야 하는 경우 `$once(event)`를 사용합니다.
- 이벤트 청취를 멈추기 위해 `$off()`를 사용합니다.

## 발생과 청취

정말 간단한 예제로 시작해 보겠습니다.

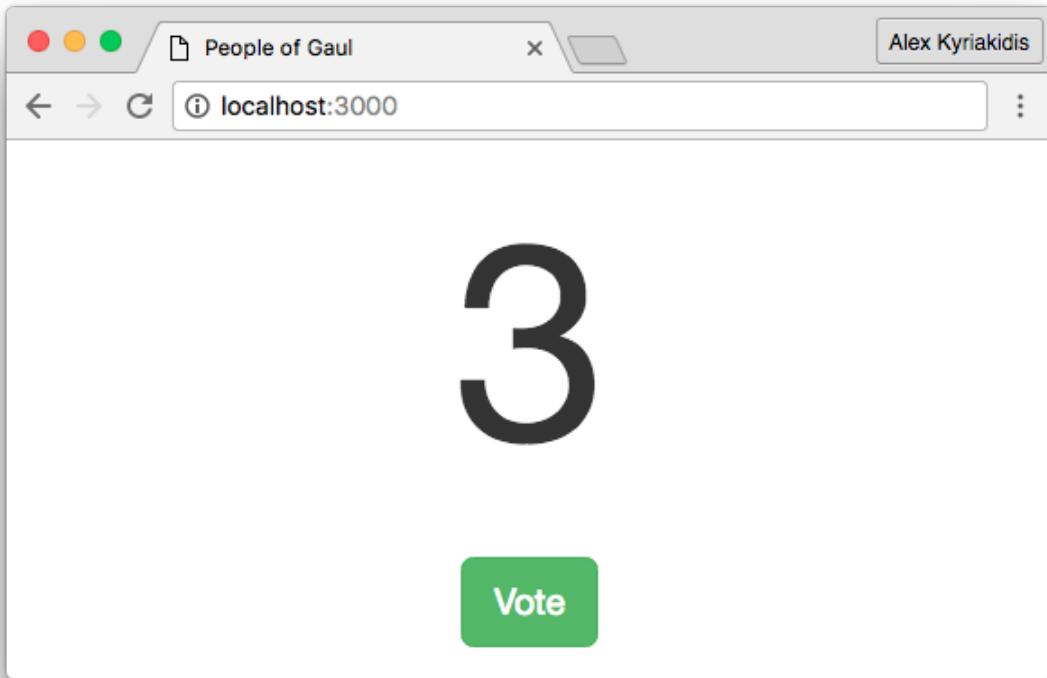
다음 코드 블럭은 카운터와 Vote 버튼이 있는 페이지를 나타냅니다. 버튼을 클릭하면 이름이 ‘voted’인 이벤트를 발생합니다. 이벤트에 대한 이벤트 리스너도 있어 이벤트가 발생하면 투표 수를 증가시킵니다.

```
1 <html>
2 <head>
3   <title>Emit and Listen</title>
4   <link href= "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel= "stylesheet" >
5 </head>
6 <body>
7   <div class= " container text-center" >
8     <p style= "font-size: 140px; ">
9       {{ votes }}
10      </p>
11      <button class= " btn btn-primary " @click= " vote " >Vote</button>
12    </div>
13  </body>
```

---

<sup>44</sup><http://vuejs.org/api/#Instance-Methods-Events>

```
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     votes: 0
21   },
22   methods:
23   {
24     vote: function (writer) {
25       this.$emit('voted')
26     },
27   },
28   created () {
29     this.$on('voted', function(button) {
30       this.votes++
31     })
32   }
33 })
34 </script>
35 </html>
```



### 결과 예시

리스너를 생명주기 혹은 중에 `created`에 등록합니다. `this`는 `vote` 메소드와 `created` 혹은 안의 Vue 인스턴스에 바인딩 되어있습니다. 그래서 `this.$on`과 `this.$emit`을 사용하여 `$on`과 `$emit` 함수에 접근할 수 있습니다.

## 생명주기 혹은

생명주기 혹은 Vue 관련 이벤트가 발생할 때 실행되는 함수입니다.

Vue 2의 혹은 목록입니다.

훅	시점
<b>beforeCreate</b>	인스턴스 초기화 후 데이터 감시 및 이벤트/감시자 설정 전
<b>created</b>	인스턴스가 생성된 직후.
<b>beforeMount</b>	마운트 시작 직전.
<b>mounted</b>	이제 막 DOM에 마운트 된 직후.
<b>beforeUpdate</b>	데이터 변경시, 가상 DOM이 다시 렌더링되고 패치되기 전
<b>updated</b>	데이터 변경 후 가상 DOM이 다시 렌더링되고 된 후
<b>activated</b>	keep-alive 상태의 컴포넌트가 활성화 될 때
<b>deactivated</b>	keep-alive 상태의 컴포넌트가 비활성화 될 때
<b>beforeDestroy</b>	Vue 인스턴스가 파괴되기 직전
<b>destroyed</b>	Vue 인스턴스가 파괴된 후

모두 기억해야 할 필요는 없지만 훅의 존재를 알고 있는 것이 좋습니다. 더 자세히 알아보려면 [Vue API<sup>45</sup>](#)를 확인하세요.

## 부모-자식 간 통신

부모 컴포넌트가 자식 컴포넌트의 이벤트를 수신할 필요가 있을 때 상황이 약간 달라집니다. `this.$on`/`this.$emit`를 사용할 수 없습니다. 왜냐하면 `this`는 각각 다른 인스턴스에 바인딩될 것이기 때문입니다.

**v-on (@)** [이벤트 리스너](#)를 기억하십니까? 부모 컴포넌트는 자식 컴포넌트가 사용되는 템플릿에서 **v-on**을 사용하여 자식 컴포넌트에서 방출된 이벤트를 직접 청취할 수 있습니다. 앞의 예제에 따라, name 속성을 가진 음식 컴포넌트를 생성합니다. 템플릿에서 name을 표시하는 버튼을 추가합니다. 버튼을 클릭하면 vote 이벤트를 방출하게 됩니다.

Food 컴포넌트

---

```
Vue.component('food', {
  template: '#food',
  props: [ 'name' ],
  methods: {
    vote: function () {
      this.$emit('voted')
    }
  },
})
```

---

<sup>45</sup> <http://vuejs.org/api/#Options-Lifecycle-Hooks>

## Food 컴포넌트의 템플릿

```
<template id= " food " >
  <button class= " btn btn-default " @click= " vote ">{{ name }}</button>
</template>
```

부모 인스턴스에서 <button>은 <food @voted= " countVote " ></food>로 대체 됩니다. **@voted= " countVote "** 는 자식의 voted 이벤트가 발생할 때 countVote 메소드가 실행됨을 의미합니다. **this.\$on** 리스너는 더이상 필요없기 때문에 제거할 수 있습니다.

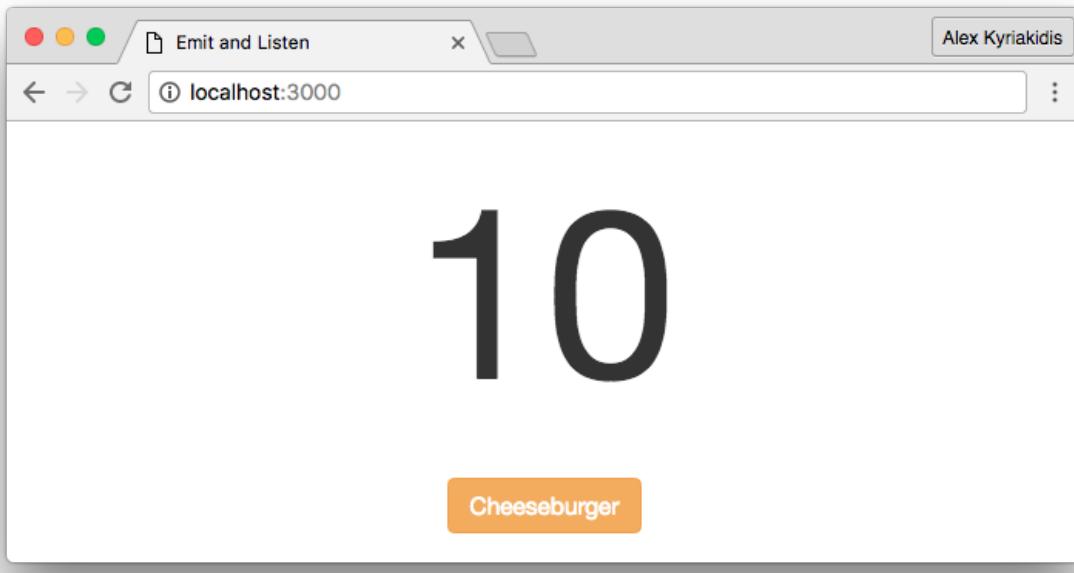
## 부모 컴포넌트

```
new Vue({
  el: '.container',
  data: {
    votes: 0
  },
  methods: {
    countVote: function () {
      this.votes++
    }
  }
})
```

## 부모 컴포넌트의 템플릿

```
<div class= " container text-center " >
  <p style= " font-size: 140px; ">
    {{ votes }}
  </p>
  <food @voted= " countVote " name= " Cheeseburger " ></food>
</div>
```

브라우저에서 실행해보면, 결과가 같은 것을 볼 수 있습니다.



부모 자식간 통신

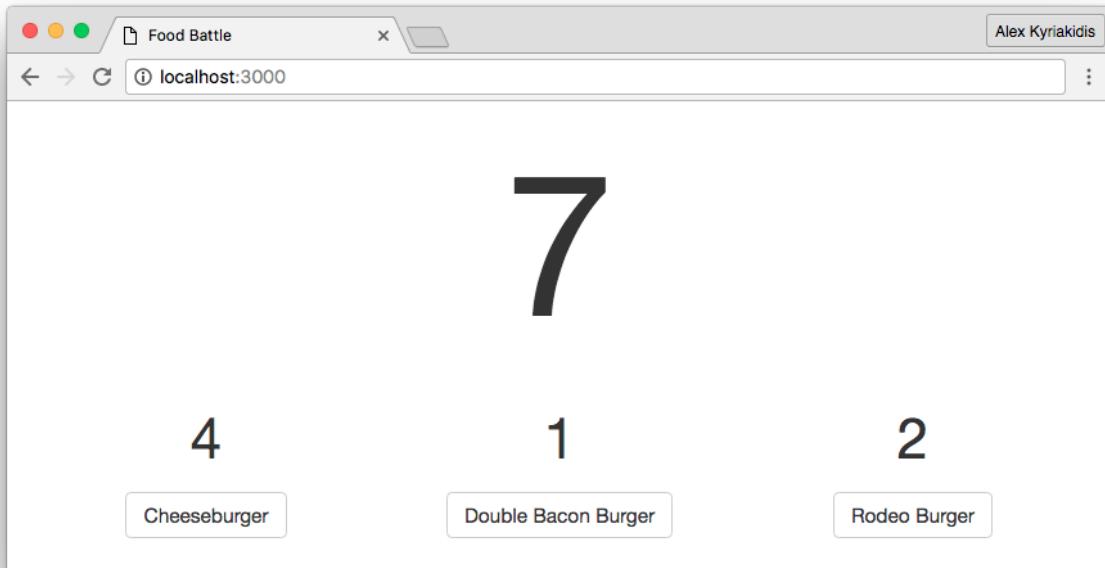
## 이벤트에서 전달인자 사용

3개의 food 컴포넌트를 만들겠습니다. 각 인스턴스는 투표 수를 가질 것입니다. 음식 중 하나가 투표를 받으면 투표 수가 증가하고 부모 컴포넌트에 있는 총 투표 수를 수정하기 위한 이벤트가 방출됩니다.

```
1 <html>
2 <head>
3   <title>Food Battle</title>
4   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
5 </head>
6 <body>
7   <div class="container text-center">
8     <p style="font-size: 140px;">
9       {{ votes }}
10      </p>
11    </div>
12    <div class="row">
13      <food @voted="countVote" name="Cheeseburger" ></food>
14      <food @voted="countVote" name="Double Bacon Burger" ></food>
```

```
16      <food @voted= " countVote " name= " Rodeo Burger " ></food>
17    </div>
18  </div>
19
20 </body>
21 <template id= " food " >
22   <div class= " text-center col-lg-4 " >
23     <p style= " font-size: 40px; " >
24       {{ votes }}
25     </p>
26     <button class= " btn btn-default " @click= " vote " >{{ name }}</button>
27   </div>
28 </template>
29 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
30 <script type= " text/javascript " >
31 var bus = new Vue()
32
33 Vue.component( ' food ' , {
34   template: '#food',
35   props: [ ' name ' ],
36   data: function () {
37     return {
38       votes: 0
39     }
40   },
41   methods: {
42     vote: function () {
43       this.votes++
44       this.$emit( ' voted ' )
45     }
46   }
47 })
48 new Vue({
49   el: '.container',
50   data: {
51     votes: 0
52   },
53   methods: {
54     countVote: function () {
55       this.votes++
56     }
57   }
58 })
59 </script>
```

61 &lt;/html&gt;



### 여러개의 컴포넌트 인스턴스

지금까지 특별한 것은 없습니다. 앱을 더 멋지게 만드려면, 투표 기록을 추가하세요. 이 기록은 음식에 투표할 때마다 수정됩니다. `voted` 이벤트를 방출할 때 음식 이름을 전달하기 위해 자식 컴포넌트를 수정해야합니다.



### 정보

`$emit` 함수는 event name 전달인자와 함께 리스너의 콜백 함수에 추가로 전달합니다.  
예제: `vm.$emit('voted', 'Alex', 'Sunday', 'Bob Ross')`

음식 이름을 얻기 위한 두개의 방법을 얻었습니다. 하나는 당연히 컴포넌트의 `name` 속성에서 얻을 수 있습니다. 두번째는 이벤트를 방출한 엘리먼트에 접근하여 텍스트 내용을 찾는 것입니다. 두번째 방법을 알아보겠습니다.

클릭한 엘리먼트에 어떻게 접근할 수 있는지 알아보기 위해 `Food`의 `vote` 메소드 내에서 `event` 변수를 콘솔에 로그를 남길 수 있습니다.

## Food 컴포넌트

```
Vue.component( 'food' , {  
    ...  
    methods: {  
        vote: function (event) {  
            console.log(event)  
            this.votes++  
            this.$emit( 'voted' )  
        }  
    }  
})
```



## event.srcElement

위 코드를 보면 `event.srcElement` 속성 안에서 클릭한 엘리먼트에 접근이 가능한 것을 확인할 수 있습니다. 이름은 `event.srcElement.outerText`와 `event.srcElement.textContent`에서 찾을 수 있습니다.

따라서, 이들 중 하나를 `$emit` 함수로 전달하세요.

## Food 컴포넌트

---

```
Vue.component( 'food' , {
  ...
  methods: {
    vote: function () {
      this.votes++
      this.$emit( 'voted' , event.srcElement.textContent)
    }
  }
})
```

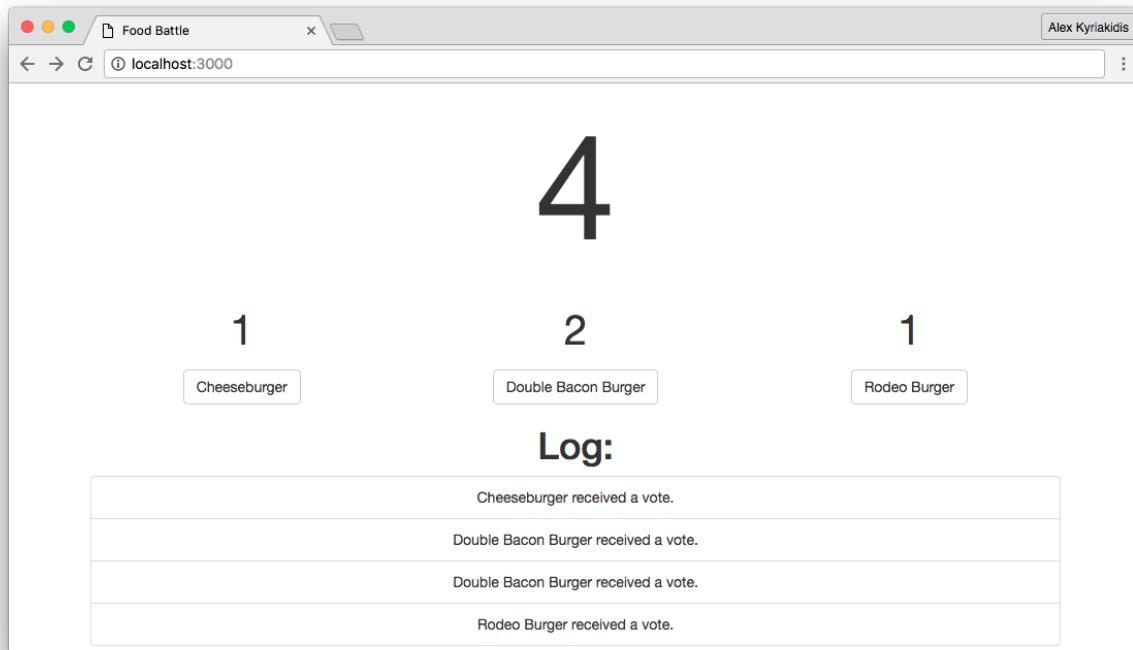
---

부모 내부에서, 투표가 발생하면 내용을 log 배열에 추가할 것입니다.

```
new Vue({
  el: '.container',
  data: {
    votes: 0,
    log: []
  },
  methods:
  {
    countVote: function (food) {
      this.votes++
      this.log.push(food + ' received a vote.')
    }
  }
})
```

log를 표시하기 위해 HTML 템플릿에 리스트를 추가합니다.

```
<h1>Log:</h1>
<ul class="list-group">
  <li class="list-group-item" v-for="vote in log">{{ vote }}</li>
</ul>
```



투표 기록

## 비 부모 자식간 통신

앞의 예제에서 한걸음 더 나아가 리셋 버튼을 추가합니다. 리셋 버튼을 클릭하면 모든 투표 카운터가 재설정 됩니다. 그리고 재설정 버튼은 모든 컴포넌트가 처리할 이벤트를 방출합니다. 하지만 자식 컴포넌트에서 이 이벤트를 어떻게 청취할 수 있을까요?

<food @voted= "countVote" >와 같은 방법을 이용하면 이벤트를 청취할 수는 있지만 자식 컴포넌트에 이벤트를 보낼 방법은 없습니다.

모든 컴포넌트가 서로 통신할 수 있게 하려면 빈 Vue 인스턴스를 중앙의 이벤트 버스로 사용하면 됩니다. 그 다음 created 흑 내에서 `this.$on` 대신 `**bus.$on**`을 사용하여 이벤트 리스너를 등록합니다. 따라서 모든 이벤트를 실행하기 위해 `**bus.$emit**`을 사용하면 됩니다.

## HTML

---

```

1 <body>
2   <div class= " container text-center " >
3     <h1>Food Battle</h1>
4     <p style= " font-size: 140px; " >
5       {{ votes.count }}
6     </p>
7     <button class= " btn btn-danger " @click= " reset " >Reset votes</button>
8     <hr>
9
10    <div class= " row " >
11      <food name= " Cheeseburger " ></food>
12      <food name= " Double Bacon Burger " ></food>
13      <food name= " Whooper " ></food>
14    </div>
15    <hr>
16
17    <h1>Log:</h1>
18    <ul class= " list-group " >
19      <li class= " list-group-item " v-for= " vote in votes.log " >{{ vote }} </li>
20    </ul>
21  </div>
22 </body>

```

---

## JavaScript

---

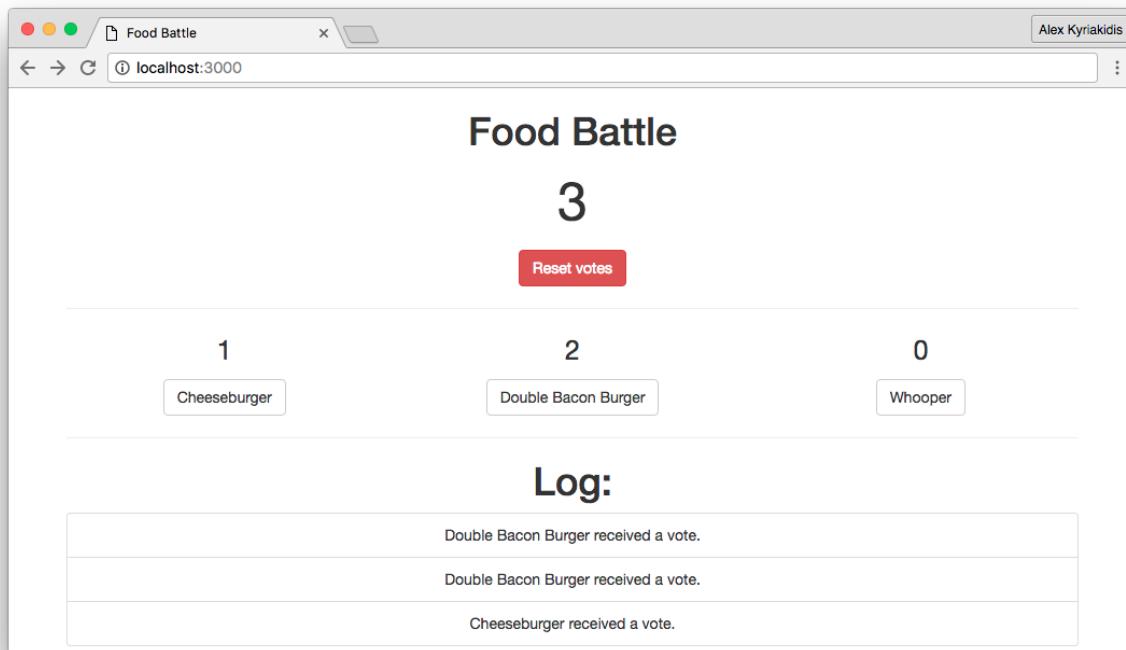
```

1 var bus = new Vue()
2
3 Vue.component( ' food ' , {
4   template: '#food',
5   props: [ ' name ' ],
6   data: function () {
7     return {
8       votes: 0
9     }
10   },
11   methods: {
12     vote: function (event) {
13       // this.name을 사용하는 대신
14       // 이벤트의 엘리먼트의 문자열을 사용할 수 있습니다
15       var food = event.srcElement.textContent;
16       this.votes++
17       bus.$emit( ' voted ' , food)
18     },

```

```
19     reset: function () {
20       this.votes = 0
21     }
22   },
23   created () {
24     bus.$on( 'reset' , this.reset)
25   }
26 })
27 new Vue({
28   el: '.container',
29   data: {
30     votes: {
31       count: 0,
32       log: []
33     }
34   },
35   methods:
36   {
37     countVote: function (food) {
38       this.votes.count++
39       this.votes.log.push(food + ' received a vote.')
40     },
41     reset: function () {
42       this.votes = {
43         count: 0,
44         log: []
45       }
46       bus.$emit( 'reset' )
47     }
48   },
49   created () {
50     bus.$on( 'voted' , this.countVote)
51   }
52 })
```

---



비 부모 자식간 통신



## 주의사항

여기서 사용하는 방법을 주의하세요.

```
bus.$on( 'voted' , this.countVote)
```

이 대신에 다음과 같이 사용해 봅시다

```
bus.$on( 'voted' , function(){
    this.vote(food)
})
```

**this**가 현재 컴포넌트의 인스턴스가 아닌 bus 인스턴스에 바인딩 되어 있기 때문에 오류가 발생했습니다.

## 이벤트 리스너 제거

하나 이상의 이벤트 리스너를 제거하기 위해 **\$off**를 사용할 수 있습니다. **\$off([event, callback])** 메소드는 여러가지 방법으로 사용할 수 있습니다.

1. `$off()`를 전달인자 없이 사용하면 모든 이벤트 리스너를 제거합니다.
2. `$off([event])`는 지정된 이벤트에 대한 모든 이벤트 리스너를 제거합니다.
3. `$off([event, callback])`는 특정 콜백에 대한 이벤트 리스너를 제거합니다.

실제로 동작하는지 확인하기 위해 Stop 버튼을 추가하여 투표가 카운트/기록 되지 않도록 합니다. `stop` 메소드를 Vue 인스턴스에 추가할 것입니다.

```
new Vue({
  ...
  methods:
  {
    ...
    stop: function () {
      bus.$off(['voted'])
    }
  }
})
```

`bus.$off(['voted'])`와 함께 사용한다면, Stop 버튼을 클릭하면 새로 들어오는 득표 수가 총 합에 추가되지 않는다는 것을 알 수 있습니다. 또한 로그에 표시되지 않습니다. 하지만 Reset Votes 버튼을 클릭하면 음식 컴포넌트의 투표 수가 모두 0으로 재설정 됩니다.

reset 리스너를 사용하지 않으려면 `bus.$off()`를 사용하여 모든 이벤트 리스너를 제거할 수 있습니다.

## 이야기로 돌아가서

이전 장의 [이야기 예제](#)를 기억하시나요? 컴포넌트 데이터를 부모 데이터와 동기화 하려고 했습니다. 해결책이 이제 명백히 보입니다.

이제 Story 컴포넌트를 다음과 같이 사용합니다.

```
<story v-for="story in stories"
:story="story"
:favorite="favorite"
@update="updateFavorite"></story>
```

Story 컴포넌트 내에서 이야기를 좋아한다고 표시하면 `update` 이벤트가 발생합니다. 방출 이벤트의 전달인자는 좋아하는 이야기입니다.

## Story 컴포넌트

---

```
Vue.component( 'story' , {  
    ...  
    methods:{  
        ...  
        updateFavorite: function(){  
            // 'update' 는 사용자 이벤트의 이름입니다  
            // 무엇이든 지정할 수 있습니다. 예: fav-update  
            this.$emit( 'update' , this.story)  
        }  
    }  
}); ...
```

---

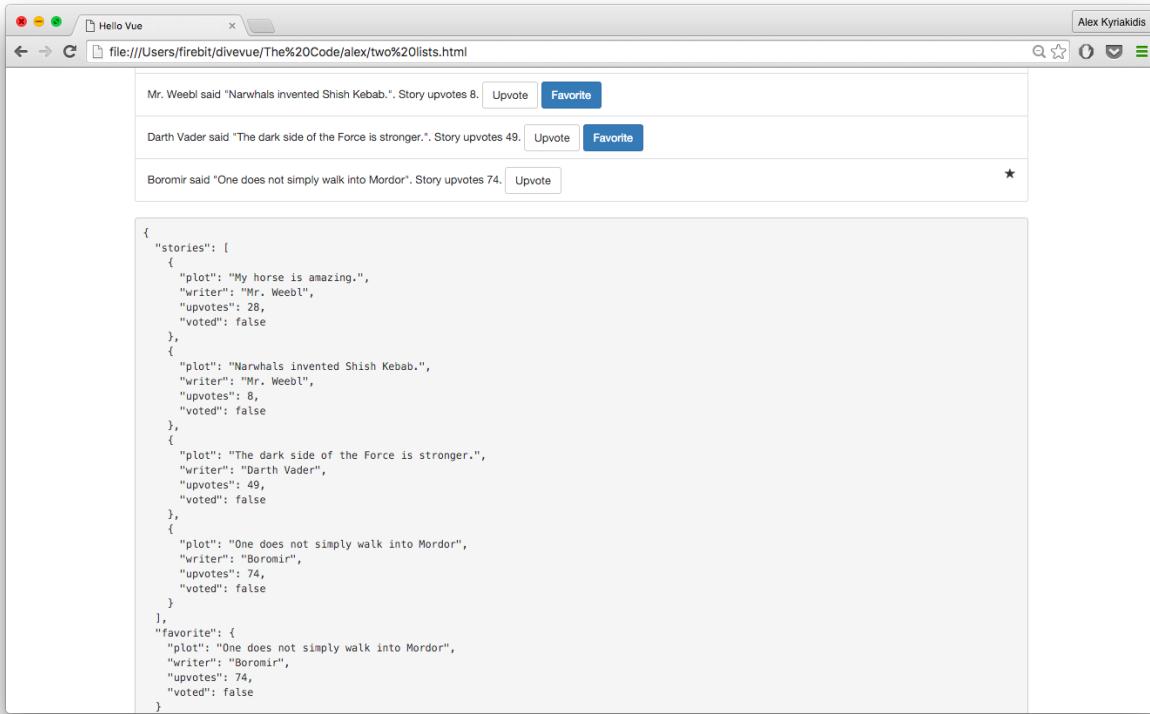
부모 인스턴스에서 favorite 변수를 추가할 것입니다. 또한 호출될 때 favorite 변수를 수정할 새로운 메소드를 생성할 것입니다.

## 부모 인스턴스

---

```
new Vue({  
    ...  
    data: {  
        ...  
        favorite: {}  
    },  
    methods: {  
        updateFavorite: function(story) {  
            this.favorite = story;  
        }  
    },  
})
```

---



### 한 이야기만 좋아할 수 있어요

이제 원하는 결과를 얻을 수 있습니다. 사용자는 한 이야기만 좋아하도록 선택할 수 있게 되었습니다. 그리고 원하는 많큼 많은 이야기에 투표할 수 있게 되었습니다.



### 정보

Vue 2에서 바인딩은 항상 단방향입니다. 부모와 자식간 데이터를 동기화 하려면 이벤트를 사용해야 합니다.



### 예제 코드

이 장의 예제 코드는 [GitHub<sup>46</sup>](#)에 있습니다.

<sup>46</sup> <https://github.com/hoottex/the-majesty-of-vuejs-2/tree/master/codes/chapter8>

## 혼자 해보기

이것은 지금까지 예제 중 가장 어렵습니다. 이 책에서 배운 모든 것을 사용해보세요. 4마리가 끄는 전차를 만듭니다. 각 전차에는 “이름(name)”과 “말(horses)” (1에서 4까지)이 있습니다. “chariot”을 이름으로 가지는 컴포넌트를 만드세요. “chariot” 컴포넌트에는 전차의 이름과 가지고 있는 말의 수를 표시해야 합니다. 액션 버튼도 있어야 합니다. 버튼의 글씨는 현재 선택된 전차에 달려있습니다.

더 자세히 설명하자면, 버튼의 글씨는 다음과 같아야 합니다.

- ‘Pick Chariot’, 사용자가 어떤 전차를 선택하기 전
- ‘Dismiss Horses’, 전차가 선택된 전차보다 말이 적은 경우
- ‘Hire Horses’, 전차가 선택된 전차보다 말이 많은 경우
- ‘Riding!’, 전차가 선택된 전차인 경우 (이 버튼은 사용하지 못하게 됨)

사용자는 전차를 골라 원하는 전차를 선택할 수 있어야 합니다.

**예상 시나리오:** 사용자는 두마리 말과 함께 전차를 선택했으며 버튼에 ‘Riding!’이라고 표시 됩니다. 세마리 말을 가진 전차에는 말이 한마리 더 많으므로 버튼에 ‘Hire Horses’라고 표시 됩니다. 한마리 말을 가진 전차는 사용자의 전차보다 말의 수가 적기 때문에 ‘Dismiss Horses’라고 표시 됩니다.



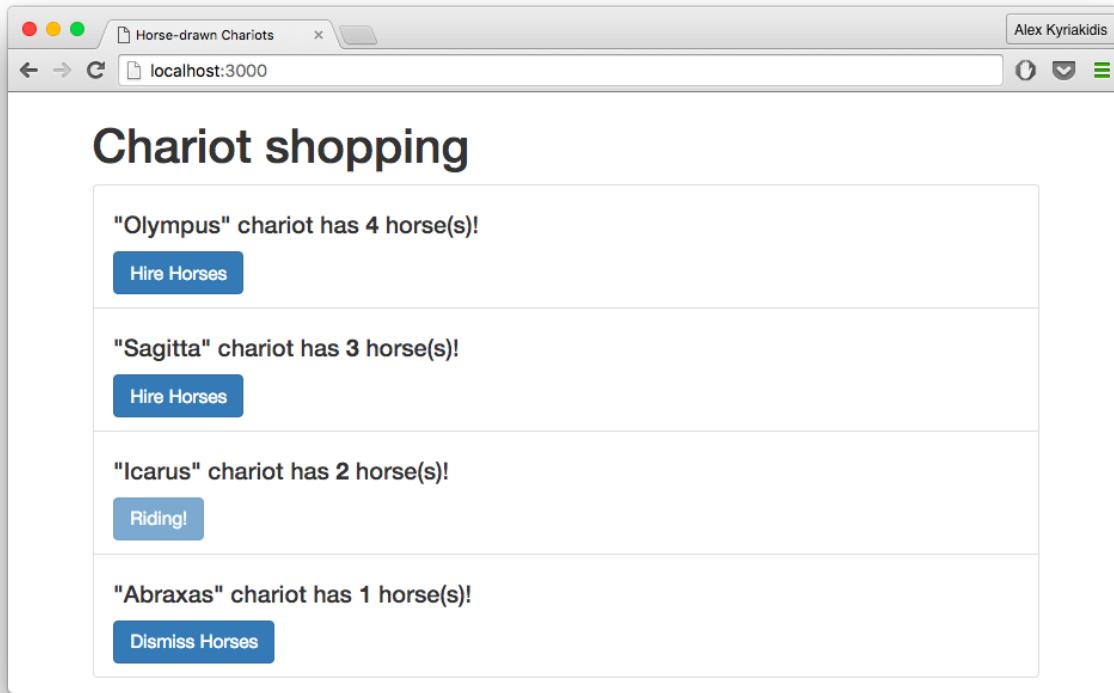
### 힌트

부모와 자식 컴포넌트간 **currentChariot** 속성을 동기화 하여야 합니다.



### 힌트

버튼을 사용할 수 없게 만드려면 **disabled= "true"** 를 사용합니다. 조건에 따라 위 내용을 적용해야 합니다.



결과 예시



## 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>47</sup>](#)를 참조하세요.

---

<sup>47</sup> <https://github.com/hoottex/the-majesty-of-vuejs-2/blob/master/homework/chapter8.html>

# 클래스와 스타일 바인딩

## 클래스 바인딩

### 객체 문법

엘리먼트의 클래스와 스타일을 조작하는데 데이터 바인딩을 사용할 수 있습니다. 이 경우 **v-bind:class**를 사용합니다. 조건에 따라 클래스를 적용하고 토글하거나 하나의 바인딩 되어있는 개체를 사용해 많은 클래스를 한 번에 적용하는데 사용할 수 있습니다.

**v-bind:class** 디렉티브는 다음 형태의 객체를 전달인자로 사용합니다.

```
{  
  'classA' : true,  
  'classB' : false,  
  'classC' : true  
}
```

그러면 **true** 값을 가지는 모든 클래스를 적용합니다. 예를 들어, 다음 엘리먼트의 클래스는 **classA** 및 **classC**가됩니다.

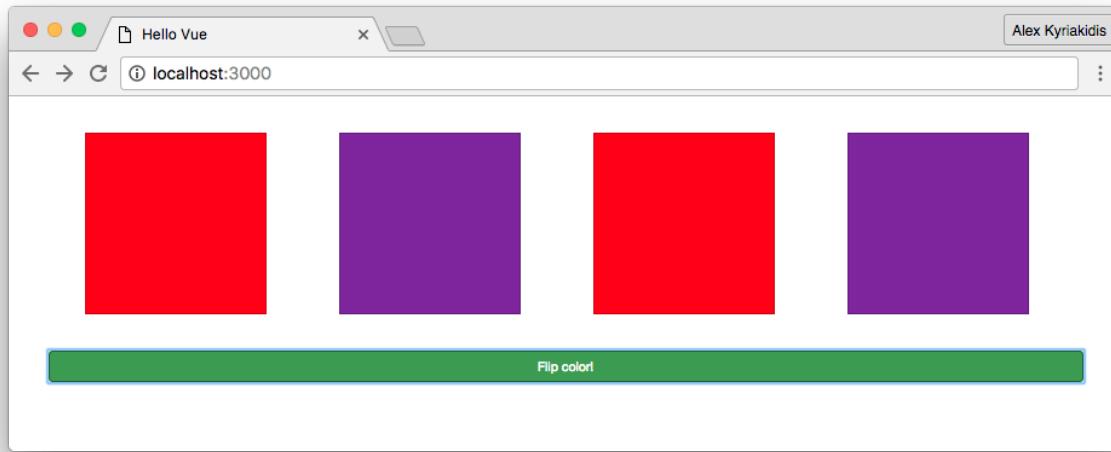
```
<div v-bind:class="elClasses" ></div>
```

```
data: {  
  elClasses:  
  {  
    'classA' : true,  
    'classB' : false,  
    'classC' : true  
  }  
}
```

클래스 속성과 함께 **v-bind** 사용하는 방법을 알아보기 위해 클래스를 토글하는 예제를 만들어 보겠습니다. **v-bind:class** 디렉티브를 사용하여 **div** 엘리먼트의 클래스를 동적으로 토글합니다.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4   <title>Hello Vue</title>
5 </head>
6 <body>
7   <div class="container text-center">
8     <div class="box" v-bind:class="{'red': color, 'blue': !color}"></div>
9     <div class="box" v-bind:class="{'purple': color, 'green': !color}"></div>
10    <div class="box" v-bind:class="{'red': color, 'blue': !color}"></div>
11    <div class="box" v-bind:class="{'purple': color, 'green': !color}"></div>
12    <button v-on:click="flipColor" class="btn btn-block btn-success">
13      Flip color!
14    </button>
15  </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
18 <script type="text/javascript">
19 new Vue({
20   el: '.container',
21   data: {
22     color: true
23   },
24   methods: {
25     flipColor: function() {
26       this.color = !this.color;
27     }
28   }
29 });
30 });
31 </script>
32 <style type="text/css">
33 .red {
34   background: #ff0000;
35 }
36 .blue {
37   background: #0000ff;
38 }
39 .purple {
40   background: #7B1FA2;
41 }
42 .green {
43   background: #4CAF50;
44 }
45 .box {
```

```
46 float: left;  
47 width: 200px;  
48 height: 200px;  
49 margin: 40px;  
50 border: 1px solid rgba(0, 0, 0, .2);  
51 }  
52 </style>  
53 </html>
```



박스의 색 토글



박스의 색 토글

편의상 **div**에 **box**를 이름으로 하는 클래스를 적용했습니다. 이 코드는 버튼을 누르면 박스의

색을 “뒤집는” 것 입니다. 버튼을 누를 때 `flipColor` 메소드가 호출되며, 처음 `true`로 설정된 `color`의 값을 반대로 바꿉니다. 그 다음 `v-bind:class` 클래스 이름을 ‘red’에서 ‘blue’로 또는 ‘purple’에서 ‘green’으로 토글합니다. `color`의 값은 참 거짓 여부에 따릅니다. 주어진 스타일은 각 클래스에 적용되어 원하는 결과를 출력합니다.

## 정보

`v-bind:class` 디렉티브는 일반 클래스 속성과 함께 있을 수 있습니다.

그리고 예제에서 `div`은 항상 `box` 클래스를 가지고 있고 조건에 따라 `red`, `blue`, `purple` 또는 `green`로 변경됩니다.

## 배열 문법

클래스 이름을 가지는 배열을 사용해 엘리먼트 목록에 클래스 목록을 적용할 수도 있습니다.

```
<div v-bind:class=" [ 'classA' , 'classB' , anotherClass ] " ></div>
```

조건부로 클래스를 적용하는 것은 배열 내부에서 `if`를 사용하면 됩니다.

```
<div v-bind:class=" [ 'classA' , condition ? 'classB' : '' ] " ></div>
```

## 정보

인라인 `if`는 일반적으로 삼항 연산자 또는 조건부 연산자라고 부릅니다.

조건부(삼항) 연산자는 세개의 피연산자를 사용하는 유일한 자바스크립트 연산자입니다.

삼항 연산자의 문법은 `조건 ? 표현식1 : 표현식2`입니다. 조건이 참이면 연산자는 표현식1의 값을 반환하고 그렇지 않으면 표현식2의 값을 반환합니다.

인라인 `if`를 사용한 색상 뒤집기 예제는 다음과 같습니다.

```

1 <div class="container text-center">
2   <div class="box" v-bind:class=" [ color ? 'red' : 'blue' ] "></div>
3   <div class="box" v-bind:class=" [ color ? 'purple' : 'green' ] "></div>
4   <div class="box" v-bind:class=" [ color ? 'red' : 'blue' ] "></div>
5   <div class="box" v-bind:class=" [ color ? 'purple' : 'green' ] "></div>
6   <button v-on:click="flipColor" class="btn btn-block btn-success">
7     Flip color!
8   </button>
9 </div>

```

```

1 new Vue({
2   el: '.container',
3   data: {
4     color: true
5   },
6   methods: {
7     flipColor: function() {
8       this.color = !this.color;
9     }
10  }
11 });

```



## 팁

실제로 클래스 배열 내에서 변수 대신 클래스 이름을 사용하려면 작은 따옴표를 사용하십시오. `v-bind:class=" [ variable, 'classname' ] "`

## 스타일 바인딩

### 객체 문법

`v-bind:style`에 사용하는 객체 구문은 매우 간단합니다. 자바스크립트 객체라는 점을 제외하면 CSS와 거의 유사합니다. 이번에 Vue.js에서 제공하는 `v-bind`의 축약형 지시어(:)를 사용할 것입니다.

```

1 <!-- 축약형 -->
2 <div :style=" niceStyle " ></div>

```

```

1 data: {
2   niceStyle:
3   {
4     color: 'blue',
5     fontSize: '20px'
6   }
7 }
```

:style="..." 인라인 객체 내에서 스타일 속성을 선언할 수도 있습니다.

```
<div :style=" { 'color': 'blue', 'fontSize: '20px' } " >...</div>
```

스타일 객체 내부에 참조 변수를 넣을 수도 있습니다.

```
<!-- 'niceStyle' 은 이전 예제와 같습니다 -->
<div :style=" { 'color': niceStyle.color, 'fontSize: niceStyle.fontSize' } " >
</div>
```



### 스타일 객체 바인딩

스타일 객체를 사용하여 바인딩 하는 것이 좋습니다. 그러면 템플릿이 더 간결해집니다.

## 배열 문법

**v-bind:style**에 인라인 배열 구문을 사용하면 동일한 엘리먼트에 여러 스타일 객체를 사용할 수 있습니다. 즉, 모든 리스트 항목에 **niceStyle**의 **color**와 **fontsize**를 적용할 수 있습니다. 글꼴 스타일은 **badStyle**입니다.

```
1  <!-- 축약형 -->
2  <div :style= " [niceStyle, badStyle] " ></div>

1  data: {
2      niceStyle:
3      {
4          color: 'blue',
5          fontSize: '20px'
6      }
7      badStyle:
8      {
9          fontStyle: 'italic'
10     }
11 }
```



## 중급자를 위한 정보

`v-bind:style`를 이용하는 경우 CSS 속성에 브라우저 공급자 접두어가 필요한 transform과 같은 속성을 사용할 때, Vue.js 는 자동으로 적절한 접두어를 감지하여 적용된 스타일에 추가합니다.

여기<sup>48</sup>에서 브라우저 공급자 접두사에 대한 자세한 정보를 찾을 수 있습니다.

---

<sup>48</sup> [https://developer.mozilla.org/en-US/docs/Glossary/Vendor\\_Prefix](https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix)

## 이벤트를 이용한 바인딩

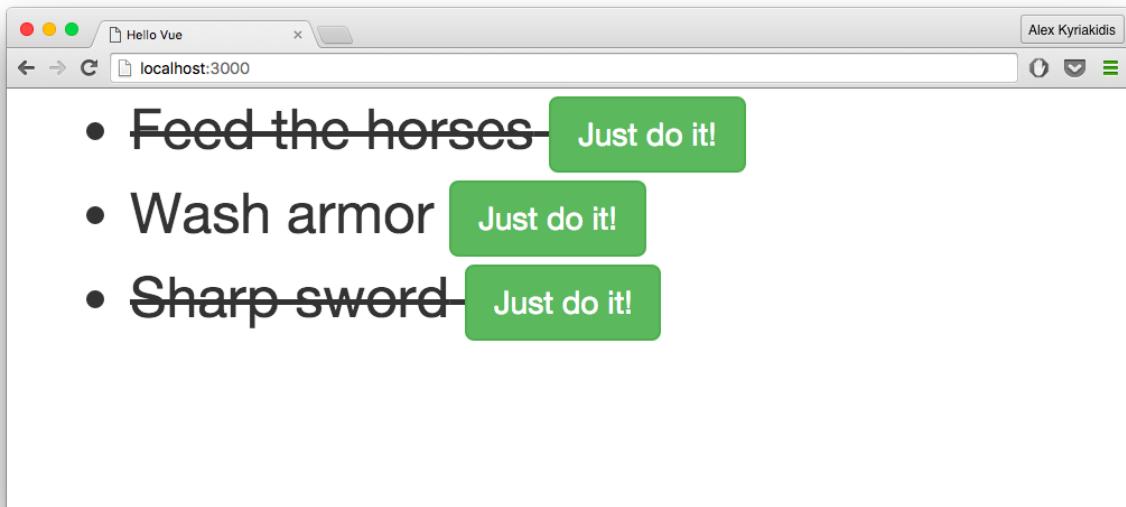
```
1 <html>
2 <head>
3 <link href= "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet" >
4 <title>Hello Vue</title>
5 </head>
6 <body class= "container-fluid" >
7   <div id= "app" >
8     <ul>
9       <li :class= "' completed' : task.done" 
10        :style= "styleObject"
11        v-for= "task in tasks" >
12          {{task.body}}
13          <button @click= "completeTask(task)" class= "btn" >
14            Just do it!
15          </button>
16          </li>
17        </ul>
18      </div>
19    </body>
20  </script type= "text/javascript" src= "https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js" ></script>
21 <script type= "text/javascript" >
22 new Vue({
23   el: '#app',
24   data: {
25     tasks: [
26       {body: "Feed the horses", done: true},
27       {body: "Wash armor", done: true},
28       {body: "Sharp sword", done: false},
29     ],
30     styleObject: {
31       fontSize: '25px'
32     }
33   },
34   methods: {
35     completeTask: function(task) {
36       task.done = !task.done;
37     }
38   },
39 });
40 </script>
```

```

43 <style type="text/css">
44   .completed {
45     text-decoration: line-through;
46   }
47 </style>
48 </html>

```

위의 예제는 tasks라는 객체 배열과 하나의 속성만 가지고 있는 **styleObject**를 가지고 있습니다. **v-for**를 사용하면 할일 목록이 렌더링되고 각각의 할일에는 불린 값이 있는 done 속성이 있습니다. done 값에 따라 클래스는 전과 같이 조건부로 적용됩니다. 할일이 완료되면 css 스타일이 적용되고 **text-decoration** 속성이 **line-through**가 됩니다. 각 할일에는 click 이벤트를 받는 버튼이 있습니다. 이 이벤트는 메소드를 실행하고 할일의 완료 상태를 변경합니다. **style** 속성은 **styleObject**에 바인딩되어 모든 할일의 font-size를 변경합니다. 보시다시피 **completeTasks** 메소드는 **task** 매개변수를 취합니다.



완료된 할일 스타일링



## 예제 코드

이 장의 예제 코드는 [GitHub<sup>49</sup>](#)에 있습니다.

---

<sup>49</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter9>

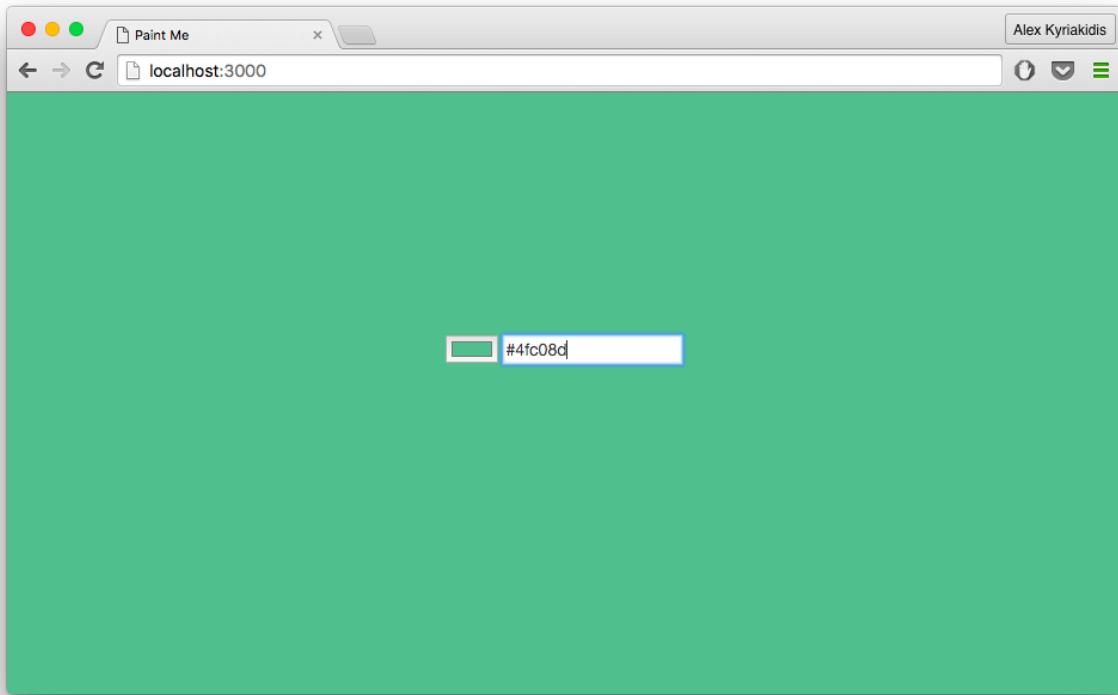
## 혼자 해보기

이 장은 재미있거나 어쩌면 힘든 과제일 수 있습니다. 색상을 입력할 수 있는 사용자 입력을 만들어 봅니다. 색상이 선택되면 그 색상을 엘리먼트에 지정해 보겠습니다. 이게 끝입니다. 색칠을 시작해보세요!



### 힌트

대부분의 브라우저에서 지원하는 `input type= "color"` 를 사용하면 더욱 쉽습니다.



결과 예시



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제<sup>50</sup>](#)를 참조하세요.

<sup>50</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter9.html>

# API 사용하기

# 머리말

이 장에서는, 약간 더 깊이 들어가서 Vue.js에서 API 사용 방법을 알아봅니다.

이전 장에서 작성한 이야기(story) 예제가 외부의 실제 데이터를 사용하도록 바꿉니다.

실제 데이터를 사용하려면, 데이터베이스를 사용해야 합니다. 이미 데이터베이스를 만드는 방법을 알고 있다고 가정하여 이 책에서는 다루지 않습니다. 예제와 함께 작업하기 위한 데이터베이스는 이미 사용할 수 있도록 만들었습니다. 아래에서 확인하세요

## CRUD

데이터베이스가 있다고 가정합시다. 데이터베이스는 CRUD(Create, Read, Update, Delete)를 수행할 수 있습니다.

조금 더 자세한 내용입니다.

- Create 새 이야기를 데이터베이스에 추가합니다.
- Read 이미 가지고 있는 이야기들을 가져옵니다.
- Update 이미 가지고 있는 이야기의 세부 내용을 수정합니다. ('upvotes' 같은 것들)
- Delete 이제는 좋아하지 않는 이야기를 삭제합니다.

Vue.js는 프론트엔드 자바스크립트 프레임워크이므로 데이터베이스에 직접 접근할 수 없습니다. 데이터베이스에 접근하려면 Vue.js와 데이터베이스를 이어주는 부분이 필요합니다. 이 부분을 API(Application Program Interface)라고 합니다.

## API

이 책은 API 설계에 대해 다루고 있지 않으므로 Laravel<sup>51</sup>로 작성된 데모 API를 제공합니다. Laravel은 Symfony2, Nette, CodeIgniter 및 Yii2와 함께 가장 강력한 PHP 프레임워크 중 하나입니다. 원하는 언어나 프레임워크를 사용하여 API를 자유롭게 작성하실 수 있습니다. 저는 Laravel을 사용합니다. Laravel은 단순하고, 훌륭한 커뮤니티를 가지고 있으며 정말 멋집니다! :)

따라서 이 책의 예제를 위한 데모 API를 사용하는 것이 좋습니다.

---

<sup>51</sup> <https://laravel.com/>

## 책의 코드 내려받기

API를 사용하려면, 책의 코드를 내려받아 서버를 시작해야 합니다. 이를 위해 아래 설명을 따라하세요.

터미널을 열고 디렉토리를 만드세요 (예 : ‘~/themajestyofvuejs2’)

```
>_ mkdir ~/themajestyofvuejs2
```

github에서 소스코드를 내려받으세요

```
>_ cd ~/themajestyofvuejs2  
>_ git clone https://github.com/hootlex/the-majesty-of-vuejs-2 .
```

다른 방법으로는, github의 소스코드를 저장소<sup>52</sup>에서 내려받으세요 그리고 원하는 위치에 압축을 풀어주시면 됩니다.

새로 생성 된 디렉토리의 ‘apis’ 아래에 있는 현재 챕터로 이동하세요.

```
>_ cd ~/themajestyofvuejs2/apis/stories
```

설치 스크립트를 실행하세요

```
>_ sh setup.sh
```

이제 더미데이터로 채워져 있는 잘 갖추어진 서버를 얻었습니다. <http://localhost:3000>로 접속하세요!

서버의 설정을 변경하려면 (호스트, 포트 등) 직접 수정해야 합니다.

아래는 스크립트의 소스코드입니다.

---

<sup>52</sup><https://github.com/hootlex/the-majesty-of-vuejs-2>

설치 스크립트: setup.sh

---

```
# navigate to chapter directory
$ cd ~/themajestyofvuejs2/apis/stories

# install dependencies
$ composer install

# Create the database
$ touch database/database.sqlite;

# Migrate & Seed
$ php artisan migrate;
$ php artisan db:seed;

# Start server
$ php artisan serve --port=3000 --host localhost;
```

---

훌륭해요! 이제 완전한 기능을 갖춘 API를 가지게 되었습니다.



## 노트

Vagrant를 사용하고 있다면, 서버는 host '0.0.0.0'에서 작동합니다.

테스트를 위해 Vagrant의 IP에 접속해야 합니다. 예를 들어 Vagrant의 박스 IP가 192.168.10.10 이면

```
$ php artisan serve - port=3000 - host 0.0.0.0;
```

브라우저를 열어 192.168.10.10:3000로 접속하세요.

데모 API를 내려받았다면 읽지 않고 다음 장으로 넘어가도 괜찮습니다.

직접 API를 만드는 경우, 이야기(stories)를 저장할 데이터베이스 테이블을 만들어야 합니다. 아래의 칼럼들은 반드시 있어야 합니다.

칼럼 이름	타입
<code>id</code>	Integer, Auto Increment
<code>plot</code>	String
<code>writer</code>	String
<code>upvotes</code>	Integer, Unsigned

다음 장의 예제를 위해 가짜 데이터를 추가하는 것을 잊지 마세요.

## API 엔드포인트

엔드포인트는 단순히 URL입니다. `http://example.com/foo/bar`는 엔드포인트이고 도메인은 모든 엔드포인트에 대해 동일하므로 `/foo/bar`를 호출하면 됩니다.

**Story** 리소스를 관리하기 위해 5개의 엔드포인트가 필요합니다. 엔드포인트는 지정된 동작을 수행합니다

HTTP 메소드	URI	동작
GET/HEAD	api/stories	모든 이야기를 가져옵니다
GET/HEAD	api/stories/{id}	특정 이야기를 가져옵니다
POST	api/stories	새 이야기를 만듭니다
PUT/PATCH	api/stories/{id}	기존 이야기를 수정합니다
DELETE	api/stories/{id}	특정 이야기를 삭제합니다

위의 표에서 알 수 있듯이 모든 ‘이야기들(stories)’이 있는 목록을 얻기 위해 `api/stories`에 대한 HTTP GET 또는 HEAD 요청을 해야 합니다. 기존의 `story`를 수정하려면 `api/stories/{storyID}`에 HTTP PUT 또는 PATCH 요청을 해야 합니다. 이 때 `{storyID}`를 이야기의 id로 바꾸고 수정하려는 데이터를 제공해야 합니다. 동일한 로직이 모든 엔드포인트에 적용됩니다.

서버 주소가 `http://localhost:3000`라고 가정하면 웹 브라우저에서 `http://localhost:3000/api/stories` 주소에서 모든 이야기의 목록이 JSON 형식으로 출력됩니다.

The screenshot shows a browser window with the URL `localhost:3000/api/stories`. The page displays a JSON array of six story objects. Each object has properties: id, plot, upvotes, and writer. The stories are from Star Wars and include quotes from Anakin, Padme, C-3PO, Princess Leia, Darth Vader, and Senator Palpatine.

```

[{"id": 2, "plot": "Anakin, you're breaking my heart! And you're going down a path I cannot follow!", "upvotes": "586", "writer": "Padme"}, {"id": 3, "plot": "Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.", "upvotes": "384", "writer": "C-3PO"}, {"id": 4, "plot": "Help me Obi-Wan Kenobi, you're my only hope.", "upvotes": "896", "writer": "Princess Leia"}, {"id": 5, "plot": "I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.", "upvotes": "680", "writer": "Darth Vader"}, {"id": 6, "plot": "The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.", "upvotes": "787", "writer": "Senator Palpatine"}]
  
```

JSON 응답

## Tip

브라우저에서 원본 JSON 데이터를 보는 것은 괴로운 일입니다. 잘 정리된 JSON을 보고 싶다면 크롬 브라우저는 원본 JSON 데이터를 쉽게 읽을 수 있는 트리 뷰 형식으로 보여주는 몇 가지 훌륭한 확장 기능을 제공합니다.

[JSONFormatter<sup>53</sup>](#)는 구문 강조를 지원하고 각 노드의 왼쪽에 있는 삼각형 아이콘을 클릭하여 트리의 노드를 축소하거나 확장할 수 있는 트리 뷰를 제공합니다. 또한 원본 데이터로 전환하기 위한 버튼도 제공합니다.

원하는 확장 프로그램을 선택할 수 있지만 무조건 사용하는 것을 추천합니다.

<sup>53</sup> <https://chrome.google.com/webstore/detail/json-formatter/bcjindccaaagfpapjjmafapmmgkhhgoa>

# 실제 데이터를 사용하여 작업하기

실제 데이터베이스를 가지고 작업하려면 앞서 말한 기능(CRUD)을 추가해야 합니다.

컴포넌트 장의 마지막 예제를 활용하겠지만 이번에는 외부에서 가져온 데이터를 사용하게 됩니다. 서버와 데이터를 교환하려면 비동기 HTTP 요청(Ajax)을 사용해야 합니다.

## 정보

AJAX는 백그라운드에서 서버와 소량의 데이터를 주고 받아 웹 페이지를 비동기적으로 수정할 수 있게 해주는 기술입니다.

## 비동기로 데이터 가져오기

잠시 시간을 내여 컴포넌트 장의 마지막 예제를 살펴 보겠습니다. 여기서 볼 수 있듯 Vue 인스턴스의 data 객체 안에 stories 배열의 내용이 하드코딩 되어있습니다.

하드코딩된 이야기 배열

---

```
new Vue({
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weebly',
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weebly',
      },
      ...
    ]
  }
})
```

---

이번에는 서버에서 기존 이야기를 가져오려고 합니다.

HTTP GET 요청을 해야합니다. 우선 jQuery를 이용하겠습니다. 이 장의 뒷부분에서 vue-resource<sup>54</sup>로 변경하는 과정에 대해 알아봅니다.

---

<sup>54</sup><https://github.com/vuejs/vue-resource>

AJAX 요청을 위해 `$.get()`를 사용합니다. 이는 jQuery 함수이며 HTTP GET 요청으로 서버에서 데이터를 가져옵니다. `$.get()`의 전체 문서는 [여기<sup>55</sup>](#)에 있습니다.

## 정보

`vue-resource`은 서버에 요청하고 결과를 전달 받을 수 있도록 도와주는 `Vue.js`의 플러그인입니다.

`$.get()` 함수의 문법입니다.

```
$.get(
  url,
  success
);
```

위 내용은 아래 코드의 축약형입니다.

```
$.ajax({
  url: url,
  success: success
});
```

무엇을 해야 할까요? 이야기를 `$.get( '/api/stories' )` 요청을 이용해 서버에서 가져와야 합니다. 그리고 전달받은 데이터를 `stories` 배열에 넣어야 합니다.

이제 공통적으로 해야할 일이 있습니다. 요청을 할 때에는 인스턴스가 준비되어 있어야 합니다. `Vue`의 [라이프사이클](#) `훅`을 기억하시나요?

`mounted` `훅`에서 호출해야 인스턴스가 마운트된 직후에 데이터를 가져올 수 있습니다.

## 주의사항

`mounted` `훅`은 jQuery의 `$( document ).ready()`와 완전히 같지는 않습니다. `mounted` 상태는 문서 안에 있는 것을 완전히 보장하지 않습니다. 만약 DOM이 문서 안에 완전히 준비된 상태임을 보장하려면 아래와 같이 해야합니다.

```
mounted: function () {
  this.$nextTick(function () {
    // this.$el이 문서 안에 있을 때 작동합니다.
  })
}
```

이제 한번 해보겠습니다.

---

<sup>55</sup> <https://api.jquery.com/jquery.get/>

```

1 <div id= " app " >
2   <div class= " container " >
3     <h1>Let ' s hear some stories!</h1>
4     <ul class= " list-group " >
5       <story v-for= " story in stories " :story= " story " >
6         </story>
7       </ul>
8       <pre>{{ $data }}</pre>
9     </div>
10    </div>
11  <template id= " template-story-raw " >
12    <li class= " list-group-item " >
13      {{ story.writer }} said "{{ story.plot }}"
14      <span>{{story.upvotes}}</span>
15    </li>
16  </template>

1 <script src= " https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js " ></script>
2 <script src= " https://code.jquery.com/jquery-2.1.4.min.js " ></script>
3 <script type= " text/javascript " >
4 Vue.component( ' story ' , {
5   template: "#template-story-raw",
6   props: [ ' story ' ],
7 });
8
9 var vm = new Vue({
10   el: '#app',
11   data: {
12     stories: []
13   },
14   mounted: function(){
15     $.get( '/api/stories' , function(data){
16       vm.stories = data;
17     })
18   }
19 })
20 </script>

```

jQuery를 [cdnjs<sup>56</sup>](https://cdnjs.com/libraries/jquery/)에서 가져와서 작업하였습니다. mounted 흑에서 GET 요청 하면 됩니다. 요청이 성공적으로 완료 되면 전달 받은(콜백 안에 있는) 데이터를 stories 배열에 넣어야 합니다.

---

<sup>56</sup><https://cdnjs.com/libraries/jquery/>

#	Plot	Writer	Upvotes
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	583
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787
7	Aren't you a little short for a storm trooper?	Princess Leia	387
9	The force is strong with this one.	Darth Vader	836
10	There is good in him. I've felt it.	Luke Skywalker	318
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867
16	Now, young Skywalker... you will die.	The Emperor	899

Here's a list of all your stories.

### 이야기 가져오기



주의하세요 `stories` 변수를 사용하기 위해 `this.stories` 대신에 `vm.stories`를 사용했습니다. 콜백안에서 `this`는 `Vue`를 가리키는 것이 아니기 때문입니다. 그렇기 때문에 `Vue` 인스턴스를 저장하기 위해 `vm` 변수를 만들어 코드 어디서든 사용할 수 있게 만들었습니다. `this`에 대해 더 자세히 알기 위해 문서<sup>57</sup>를 확인하세요.

## 리팩토링

텍스트 에디터에서 많은 양의 코드를 가지고 작업하면 혼동을 줄 수 있습니다. 브라우저도 마찬가지입니다. 이러한 이유로 예제 코드를 리팩토링 합니다. `<ul>` 대신에 `<table>` 엘리먼트를 사용하여 `stories`를 출력합니다.

<sup>57</sup> <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>

```
1 <div id= " app " >
2     <table class= " table table-striped " >
3         <tr>
4             <th>#</th>
5             <th>Plot</th>
6             <th>Writer</th>
7             <th>Upvotes</th>
8             <th>Actions</th>
9         </tr>
10        <tr v-for= " story in stories " is= " story " :story= " story " ></tr>
11    </table>
12    <template id= " template-story-raw " >
13        <tr>
14            <td>
15                {{story.id}}
16            </td>
17            <td>
18                <span>
19                    {{story.plot}}
20                </span>
21            </td>
22            <td>
23                <span>
24                    {{story.writer}}
25                </span>
26            </td>
27            <td>
28                {{story.upvotes}}
29            </td>
30        </tr>
31    </template>
32    <p class= " lead " >Here ' s a list of all your stories.
33    </p>
34    <pre>{{ $data }}</pre>
35 </div>
```

하지만 약간의 문제가 있습니다.

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	
7	Aren't you a little short for a storm trooper?	Princess Leia	387	
9	The force is strong with this one.	Darth Vader	836	
10	There is good in him. I've felt it.	Luke Skywalker	318	
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	
16	Now, young Skywalker... you will die.	The Emperor	899	

Here's a list of all your stories.

## 렌더링 이슈

테이블이 제대로 렌더링 되지 않았습니다. 왜 그럴까요<sup>58</sup>?

<table>과 같은 일부 HTML 엘리먼트는 내부에 어떤 엘리먼트를 나타낼 수 있는지에 대한 제한이 있습니다. 사용할 수 없는 사용자 정의 엘리먼트는 올바르게 렌더링되지 않습니다. 이러한 경우에 **is**를 사용하여 사용자 정의 엘리먼트임을 알려줄 수 있습니다.

그러므로 특수 속성 **is**를 사용하여 이 문제를 해결할 수 있습니다.

```
<table>
  <tr is="my-component"></tr>
</table>
```

아래와 같이 변경합니다.

```
<tr v-for="story in stories" is="story" :story="story"></tr>
```

<sup>58</sup><http://goo.gl/Xr9RoQ>

A screenshot of a web browser window titled "Stories". The address bar shows "localhost:3000/stories.html". The page content is a list of quotes, each with a character name and a quote text followed by a vote count. The quotes are:

- Padme said "Anakin, you're breaking my heart! And you're going down a path I cannot follow!" 583
- C-3PO said "Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations." 384
- Princess Leia said "Help me Obi-Wan Kenobi, you're my only hope." 896
- Darth Vader said "I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master." 680
- Senator Palpatine said "The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural." 787
- Princess Leia said "Aren't you a little short for a storm trooper?" 387
- Darth Vader said "The force is strong with this one." 836
- Luke Skywalker said "There is good in him. I've felt it." 318
- Chancellor Palpatine said "Good Anakin. Good. Kill him. Kill him now." 397
- Darth Vader said "I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master." 867
- The Emperor said "Now, young Skywalker... you will die." 899

잘 표시된 테이블

이제 더 좋아 보입니다!

## 데이터 수정

이전에 사용자가 좋아하는 이야기를 투표할 수 있는 기능을 만들었습니다. 이제 더 많은 것들을 해볼 차례입니다. 이야기가 표를 얻을 때마다 서버에 알림을 보내 데이터베이스도 수정합니다.

이미 존재하는 이야기를 수정하기 위해 `api/stories/{storyID}`에 HTTP PATCH 또는 PUT 요청을 합니다.

이전에 만든 `upvoteStory` 함수안에서 `story`의 `upvotes`가 증가하면 HTTP 요청을 해야합니다. 서버에 데이터 수정을 요청하기 위해 요청의 페이로드에 새로 수정한 `story` 변수를 전달합니다.

```
1 <td>
2   <div class= " btn-group " >
3     <button @click= " upvoteStory(story) " class= " btn btn-primary " >
4       Upvote
5     </button>
6   </div>
7 </td>
```

```
1 Vue.component( ' story ' ,{
2   template: '#template-story-raw' ,
3   props: [ ' story ' ],
4   methods: {
5     upvoteStory: function(story){
6       story.upvotes++;
7       $.ajax({
8         url: '/api/stories/' +story.id,
9         type: ' PATCH ' ,
10        data: story,
11      });
12    }
13  },
14 })
```

**upvote** 함수를 가져와 **story** 컴포넌트 안에 배치하였습니다. PATCH 요청을 만들고 새로운 데이터를 제공하면 서버는 **upvotes** 카운트를 수정합니다.

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button>
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	<button>Upvote</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button>
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	<button>Upvote</button>
16	Now, young Skywalker... you will die.	The Emperor	899	<button>Upvote</button>

### 이야기 투표

## 데이터 제거

stories 중 싫어하는 이야기를 삭제하는 기능을 만들겠습니다. **story**를 배열과 DOM에서 제거하려면 검색 후 **stories** 배열에서 제거해야합니다.

```

1 <td>
2   <div class= "btn-group" >
3     <button @click= "upvoteStory(story)" class= "btn btn-primary" >
4       Upvote
5     </button>
6     <button @click= "deleteStory(story)" class= "btn btn-danger" >
7       Delete
8     </button>
9   </div>
10 </td>

```

actions 칼럼에 삭제 버튼을 추가하고 삭제 메소드를 바인딩 했습니다. **deleteStory** 메소드는 다음의 작업을 할 것입니다

```

1 Vue.component( 'story' ,{
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // story 찾기
7       var index = vm.stories.indexOf(story);
8
9       // 삭제
10      vm.stories.splice(index, 1)
11    }
12  }
13  ...
14 })

```

물론 이번에는 실제로 지워지지 않습니다. 데이터베이스에서 이야기를 지우려면 AJAX DELETE 요청을 해야합니다.

```

1 Vue.component( 'story' ,{
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // story 찾기
7       var index = vm.stories.indexOf(story);
8
9       // 삭제
10      vm.stories.splice(index, 1)
11
12      // DELETE 요청
13      $.ajax({
14        url: '/api/stories/' +story.id,
15        type: 'DELETE'
16      });
17    },
18  }
19  ...
20 })

```

이전과 같이 URL에 **DELETE** 요청을 해야 합니다. 이제 모든 것이 준비 되었고 데이터베이스와 DOM 모두에서 이야기를 삭제할 수 있습니다.

A screenshot of a web browser window titled "Stories". The URL in the address bar is "localhost:3000/stories.html". The page displays a table of 11 stories. Each story has a row with columns for the ID (e.g., #2, #3, ..., #11), the plot (e.g., "Anakin, you're breaking my heart! And you're going down a path I cannot follow!"), the writer (e.g., Padme, C-3PO, Princess Leia, Darth Vader, etc.), the number of upvotes (e.g., 586, 384, 896, 680, etc.), and two buttons for "Upvote" and "Delete". Below the table, a message says "Here's a list of all your stories."

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button> <button>Delete</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button> <button>Delete</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button> <button>Delete</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button> <button>Delete</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button> <button>Delete</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button> <button>Delete</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button> <button>Delete</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button> <button>Delete</button>

### 투표와 이야기 삭제

지금까지 할 수 있는 것은 모두 했습니다. 다음 장에서 더욱 많은 것을 할 것입니다. 새 이야기를 만들고 현재 이야기를 수정할 것입니다. 우선은 jQuery를 vue-resource로 바꾸는 것부터 하겠습니다.

# HTTP 클라이언트

## 소개

이전의 Vue는 자체 HTTP클라이언트를 사용했었습니다. 하지만 Vue.js 팀은 공식 라이브러리에서 이를 포함하지 않기로 결정했습니다. Evan You의 글을 읽어보세요. [vue-resource 윤퇴<sup>59</sup>](#)는 이러한 결정을 하게된 이유를 설명합니다.

이번 챕터에서는 [axios<sup>60</sup>](#)를 사용하여 모든 웹 요청을 다시 구현합니다.

axios를 사용하기 전에 vue-resource를 사용하는 예를 보여주려 합니다. 어떤 것이 더 여러분에게 맞는지 살펴보고 결정하세요. jQuery는 물론 괜찮습니다. 하지만 AJAX 호출에만 사용한다면 사용하지 않는 것이 좋습니다.

## Vue-resource

Vue-resource는 웹 요청을 위한 서비스를 제공하고 XMLHttpRequest 또는 JSONP를 사용해 응답을 처리합니다.

[GitHub<sup>61</sup>](#)의 설치 방법과 설명서를 읽어보세요. 이전 챕터에서 해온 것처럼 [cdn<sup>62</sup>](#)에서 “가져오기(pull in)”할 것입니다.

서버에서 데이터를 가져오려면 vue-resource의 \$http 메소드를 아래 코드와 같이 사용해야 합니다.

```
// GET request
this.$http({url: '/someUrl', method: 'GET'})
  .then(function (response) {
    // 성공한 경우 콜백
  }, function (response) {
    // 오류의 경우 콜백
});
```

<sup>59</sup><https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4#.lew43e17f>

<sup>60</sup><https://github.com/mzabriskie/axios>

<sup>61</sup><https://github.com/vuejs/vue-resource>

<sup>62</sup><https://cdnjs.com/libraries/vue-resource>

## i 안내

vue-resource를 사용할 때, Vue 인스턴스는 `this.$http(options)` 함수를 제공합니다. 이 함수는 HTTP 요청을 만들기 위한 options 객체를 이용해 Promise를 반환합니다. 또한 Vue 인스턴스는 모든 함수 콜백에서 자동으로 `this`에 바인딩됩니다.

method 옵션을 전달하는 대신 요청된 모든 유형의 축약형 메소드가 있습니다.

### 요청 축약형 메소드

---

```

1 this.$http.get(url, [data], [options]).then(successCallback, errorCallback);
2 this.$http.post(url, [data], [options]).then(successCallback, errorCallback);
3 this.$http.put(url, [data], [options]).then(successCallback, errorCallback);
4 this.$http.patch(url, [data], [options]).then(successCallback, errorCallback);
5 this.$http.delete(url, [data], [options]).then(successCallback, errorCallback);
```

---

Evan의 글에 따르면, 사용하는데 문제가 없으면 계속 사용해도 괜찮습니다. 하지만 이제부터 글에서 추천한 axios를 사용할 것입니다.

## axios

axios는 브라우저와 Node.js에서 사용할 수 있는 Promise 기반의 HTTP 클라이언트입니다. 또한 vue-resource가 제공하는 거의 모든 API를 가지며 취소 또한 지원합니다. 그리고 TypeScript<sup>63</sup>도 지원합니다.

axios를 사용하여 GET 요청을 수행하는 방법은 다음과 같습니다.

```
// 주어진 ID를 가지고 사용자를 조회합니다.
axios({
  method: 'get',
  url: '/user/kostas'
})
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
});
```

편의를 위해 모든 요청 메소드에 대한 별칭을 제공합니다.

---

<sup>63</sup><http://www.typescriptlang.org/>

## Request method aliases

---

```

1 axios.request(config)
2 axios.get(url[, config])
3 axios.delete(url[, config])
4 axios.head(url[, config])
5 axios.post(url[, data[, config]])
6 axios.put(url[, data[, config]])
7 axios.patch(url[, data[, config]])

```

---



별칭 메소드를 사용할 때 config에서 url, method, data 속성을 지정할 필요가 없습니다.

**팁**

vue-resource와 같이 `this.$http`를 사용하려면 `Vue.prototype.$http = axios`와 같이 설정하면 됩니다. 사용하던 vue-resource를 axios로 바꾸려면 이 방법이 유용할 것입니다.

## axios 사용하기

axios를 예제에서 사용해봅니다. 우선 아래에 있는 코드를 HTML에 추가해야합니다.

`stories.html`

---

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

---

이야기를 가져오기 위해 GET 요청을 만듭니다.

`app.js`

---

```

mounted: function() {
  // GET 요청
  axios.get('/api/stories')
    .then(function (response) {
      Vue.set(vm, 'stories', response.data)
      // 또는 이전에 본 것처럼
      // vm.stories = response.data
    })
}

```

---

위의 코드를 사용하면 문제없이 이야기 목록을 가져옵니다.

축약형 메소드를 사용하여 DELETE 및 PATCH 요청도 계속 해보겠습니다.

## app.js

```
....  
upvoteStory: function(story){  
    story.upvotes++;  
    axios.patch( '/api/stories/' + story.id, story)  
}
```

## DELETE 요청

```
deleteStory: function(story){  
    var index = this.$parent.stories.indexOf(story);  
    this.$parent.stories.splice(index, 1)  
    axios.delete( '/api/stories/' + story.id)  
}
```

AJAX 메소드를 모두 다 변경했습니다!



## 정보

story 컴포넌트는 stories 배열에 접근할 수 없기 때문에, `this.$parent.stories`를 사용해야 합니다. 이 밖에 `vm.stories` 또는 이벤트 방출을 이용해서 부모 인스턴스에 접근할 수 있습니다.

## 기능 향상하기

이야기 목록을 더 멋지게 만들기 위하여 몇가지 기능을 더 추가해야 합니다. 이제 사용자에게 이야기의 줄거리, 작가 및 새로운 이야기를 만들 수 있는 방법을 추가합니다.

## 이야기 수정

사용자에게 이야기의 세부 사항을 수정할 수 있는 몇가지 입력을 제공합니다. 바인딩 된 입력 두개를 이용할 것입니다. 단, 오직 사용자가 이야기를 수정할 때만 화면에 출력합니다. 이전 장에서 작업했던 내용과 유사합니다.

`story`가 수정 상태에 있는지 정의하려면 Edit 버튼을 눌렀을 때 참이 될 `editing` 속성을 사용합니다.

```
1 <td>
2     <!--이야기를 수정하는 경우 plot을 위한 입력이 출력됨 -->
3     <input v-if= "story.editing" v-model= "story.plot" class= "form-control" >
4     </input>
5     <!--그 밖의 경우 이야기의 plot을 출력 -->
6     <span v-else>
7         {{story.plot}}
8     </span>
9 </td>
10 <td>
11     <!--이야기를 수정하는 경우 writer를 위한 입력이 출력됨 -->
12     <input v-if= "story.editing" v-model= "story.writer" class= "form-control" >
13     </input>
14     <!--그 밖의 경우 이야기의 writer를 출력 -->
15     <span v-else>
16         {{story.writer}}
17     </span>
18 </td>
19 <td>
20     {{story.upvotes}}
21 </td>
22 <td>
23     <div v-if= "!story.editing" class= "btn-group" >
24         <button @click= "upvoteStory(story)" class= "btn btn-primary" >
25             Upvote
26         </button>
27         <button @click= "editStory(story)" class= "btn btn-default" >
28             Edit
29         </button>
30         <button @click= "deleteStory(story)" class= "btn btn-danger" >
31             Delete
32         </button>
33     </div>
34 </td>
```

```

1 Vue.component( 'story' ,{
2   ...
3   methods: {
4     ...
5     editStory: function(story){
6       story.editing=true;
7     },
8   }
9   ...
10 })

```

수정된 테이블에는 두개의 새로운 입력과 버튼이 생겼습니다. `editStory` 함수를 사용하여 `story.editing`을 `true`로 설정합니다. 그리고 입력을 출력하며 Upvote와 Delete 버튼을 숨깁니다. 하지만 이 방법은 효과가 없을 것 입니다. `story.editing`을 `true`로 설정했음에도 DOM이 수정되지 않고 있습니다. 왜 이런일이 벌어질까요?

[Vue.js 블로그의 Common Gotchas<sup>64</sup>](#)에 따르면 데이터를 관찰할 때 없었던 속성을 새로 추가하면 DOM을 수정하지 않습니다. 가장 좋은 방법은 항상 반응성이 있는 속성으로 선언하는 것입니다. 런타임에 속성을 반드시 추가하거나 제거해야하는 경우가 생기면 전역 `Vue.set` 또는 `Vue.delete`을 사용하십시오.

이러한 이유로 서버에서 각 이야기를 전달받은 직후에 `story.editing` 속성을 `false`로 초기화해야합니다.

이를 하기 위해서 GET 요청이 성공한 직후에 자바스크립트의 `.map()`을 사용하여 처리해야 합니다.

```

mounted: function() {
  var vm = this;

  // GET 요청
  axios.get( '/api/stories' )
    .then(function (response) {
      // set data on vm
      var storiesReady = response.data.map(function (story) {
        story.editing = false;
        return story
      })
      vm.stories = storiesReady
      //Vue.set(vm, 'stories', storiesReady)
    });
}

```

---

<sup>64</sup><http://vuejs.org/2016/02/06/common-gotchas/>

## 정보

.map() 메소드는 각 엘리먼트에 대해 정의된 콜백 함수를 호출하고 그 결과를 담은 배열을 반환합니다. .map()에 대한 자세한 정보와 문법은 [여기<sup>65</sup>](#)에 있습니다.

이 메소드는 이야기 객체마다 **editing** 속성을 추가한 다음 수정된 이야기를 반환합니다. 새로운 변수인 **storiesReady**는 새로운 속성이 추가된 배열입니다.

이야기를 수정 중일 때 사용자에게 새 값으로 수정하거나 수정을 취소하는 두 가지 옵션을 제공해야합니다.

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
14	Fear is the path to the dark side.	Yoda	561	
15	I find your lack of faith disturbing.	Darth Vader	582	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
16	Laugh it up, Fuzz ball.	Han Solo	772	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

### 이야기 수정을 위한 입력 양식

사용자가 이야기를 수정할 때만 표시되는 버튼 두개를 추가하겠습니다. 그리고 updateStory라는 새로운 메소드를 만들겠습니다. Update Story 버튼을 누르면 현재 수정중인 이야기를 수정합니다.

```
<!-- 이야기 수정중이면 버튼을 출력 -->
<div class="btn-group" v-else>
  <button @click="updateStory(story)" class="btn btn-primary">
    Update Story
  </button>
  <button @click="story.editing=false" class="btn btn-default">
    Cancel
  </button>
</div>
```

---

<sup>65</sup> [https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)

```
Vue.component('story', {
  ...
  methods: {
    ...
    updateStory: function(story) {
      axios.patch('/api/stories/' + story.id, story)
        // editing을 false로 설정하여 입력을 숨기고
        // actions를 다시 표시합니다.
      story.editing = false;
    },
  }
})
...
})
```

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
14	Fear is the path to the dark side.	Yoda	561	<button>Update Story</button> <button>Cancel</button>
15	I find your lack of faith disturbing.	Darth Vader	582	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
16	Laugh it up, Fuzz ball.	Han Solo	772	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

### 이야기 수정

이제 잘 작동합니다. 성공적으로 PATCH 요청이 끝나면 입력을 숨기고 액션 버튼이 다시 나타나야 합니다. 그렇게 하려면 `story.editing` 속성을 `false`로 만들어야 합니다.

## 새로운 이야기 만들기

사용자가 새로운 이야기를 작성하고 서버에 저장할 수 있는 약간은 까다로운 작업을 시작하겠습니다. 첫째로 사용자가 새로운 이야기를 작성할 수 있는 입력 엘리먼트를 제공해야 합니다. 그리고 빈 이야기를 만들고 `push()` 자바스크립트 메소드를 이용하여 `stories` 배열에 추가해야 합니다. 이야기가 가지고 있는 `editing`을 제외한 모든 속성을 `null`로 초기화할 것입니다. 새 이야기를 바로 수정해야 하기 때문에 `editing` 속성을 `true`로 설정해야 합니다.

```

1 var vm = new Vue({
2   ...
3   methods: {
4     createStory: function(){
5       var newStory={
6         "plot": " ",
7         "upvotes": 0,
8         "editing": true
9       };
10      this.stories.push(newStory);
11    },
12  }
13 })

```

```

1 <p class="lead">Here's a list of all your stories.
2   <button @click="createStory()" class="btn btn-primary">
3     Add a new one?
4   </button>
5 </p>

```



## 정보

`push()` 메소드는 배열의 끝에 새로운 항목을 추가하고 새 길이를 반환합니다. `push()` 메소드와 문법에 대한 자세한 내용은 [여기<sup>66</sup>](#) 있습니다.

새 함수 `createStory`를 Vue 인스턴스에 추가합니다.

목록에 버튼을 추가했습니다. 버튼을 클릭하면 `createStory`가 호출됩니다. `newStory.editing`<sup>o</sup> 참이 되었으므로 Edit 액션 버튼과 함께 plot과 writer가 바인딩 된 입력을 렌더링 됩니다.

또한 새로운 story 객체는 데이터베이스에 저장하기 위해 서버로 전달해야 합니다. `storeStory` 메소드안에서 POST 요청을 해야 합니다.

---

<sup>66</sup> [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/push](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push)

```

1 Vue.component( 'story' ,{
2     ...
3     methods: {
4         ...
5         storeStory: function(story){
6             axios.post( '/api/stories/' , story).then(function () {
7                 story.editing = false;
8             });
9         },
10    }
11    ...
12 })

```

POST 요청의 축약형을 사용할 것 입니다. 성공한 경우의 콜백 함수에서 action 버튼을 다시 표시하고 폼의 입력 및 수정 버튼을 숨기기 위해 false로 설정해야 합니다. 아래에 새로운 방법에 따라 버튼 그룹을 수정합니다.

```

1 <td>
2 <div class= "btn-group " v-if= " !story.editing " >
3     <button @click= " upvoteStory(story) " class= " btn btn-primary " >
4         Upvote
5     </button>
6     <button @click= " editStory(story) " class= " btn btn-default " >
7         Edit
8     </button>
9     <button @click= " deleteStory(story) " class= " btn btn-danger " >
10        Delete
11    </button>
12 </div>
13 <div class= "btn-group " v-else>
14     <button class= " btn btn-primary " @click= " updateStory(story) " >
15         Update Story
16     </button>
17     <button class= " btn btn-success " @click= " storeStory(story) " >
18         Save New Story
19     </button>
20     <button @click= " story.editing=false " class= " btn btn-default " >
21         Cancel
22     </button>
23 </div>
24 </td>

```

코드 블럭에서 약간의 실수를 발견했을 것 입니다. 수정 모드(v-else 블록)에서 수정과 저장을 위한 버튼이 함께 보여집니다. 하지만 이야기는 저장 또는 수정 해야 합니다. 두가지를 한번에

할 수는 없습니다. 따라서 이야기가 오래된 내용이고 사용자가 내용을 편집하려는 경우 수정 버튼이 필요합니다. 반면에 이야기가 새로운 내용이면 저장 버튼이 필요합니다.

ID	Content	Author	Upvotes	Actions
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	Upvote Edit Delete
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	Upvote Edit Delete
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	Upvote Edit Delete
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	Upvote Edit Delete
7	Aren't you a little short for a storm trooper?	Princess Leia	387	Upvote Edit Delete
9	The force is strong with this one.	Darth Vader	836	Upvote Edit Delete
10	There is good in him. I've felt it.	Luke Skywalker	318	Upvote Edit Delete
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	Upvote Edit Delete
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	Upvote Edit Delete
16	Now, young Skywalker... you will die.	The Emperor	899	Upvote Edit Delete

A new story  Myself 0 Update Story Save New Story Cancel

Here's a list of all your stories. [Add a new one?](#)

### 작은 실수가 있습니다

이 문제를 해결하기 위해 버튼을 다시 만듭니다. Update 버튼은 기존 이야기일 때만 보여집니다. 따라서 Save new Story 버튼은 새 이야기일 때만 보여집니다.

서버에서 가져온 모든 이야기에 id 속성이 있는 것을 눈치챘을 것입니다. 이를 통해 이야기가 새 것인지 아닌지를 판단합니다.

```

1 <div class="btn-group" v-else>
2   <!-- 이야기가 기존에 있던 것이면 수정해야합니다
3   TIP: 데이터베이스에서 가져온 이야기는 id를 가지고 있습니다-->
4   <button v-if="story.id" class="btn btn-primary" @click="updateStory(story)">
5     Update Story
6   </button>
7   <!--새 이야기면 저장해야합니다-->
8   <button v-else class="btn btn-success" @click="storeStory(story)">
9     Save New Story
10  </button>
11  <!--취소 버튼은 항상 출력-->
12  <button @click="story.editing=false" class="btn btn-default">
13    Cancel

```

```
14    </button>
15  </div>
```



## 팁

이야기가 이미 데이터베이스에 있는 경우 id를 가지고 있어야 합니다.

#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

new plot new story by me 0 Save New Story Cancel

Here's a list of all your stories. [Add a new one?](#)

## 새 이야기 추가

잘 동작합니다. 어렵지 않습니다.

이 부분을 끝낸 후 앱을 테스트 할 때 또 다른 문제가 발생합니다. 새로운 이야기를 생성, 저장 및 편집한 후에 버튼에 “Update Story” 대신 “Save new Story”가 표시 됩니다. 서버에서 새로 생성된 이야기를 가져온 후 아직 보내지 않은 상태이므로 id가 없습니다. 이 문제를 해결하기 위해 데이터베이스에 저장한 직후 다시 가져올 수 있습니다.

코드를 반복하는 것을 좋아하지 않기 때문에 이야기를 가져오는 부분을 `fetchStories()` 메소드로 만들었습니다. 이제 언제든지 이 메소드를 가지고 이야기를 가져올 수 있습니다.

## The fetchStories method

---

```
1 var vm = new Vue({
2   el: '#v-app',
3   data : {
4     stories: [],
5   },
6   mounted: function(){
7     this.fetchStories()
8   },
9   methods: {
10     createStory: function(){
11       var newStory={
12         "plot": "",
13         "upvotes": 0,
14         "editing": true
15
16       };
17       this.stories.push(newStory);
18     },
19     fetchStories: function () {
20       var vm = this;
21       axios.get( '/api/stories' )
22         .then(function (response) {
23           var storiesReady = response.data.map(function (story) {
24             story.editing = false;
25             return story
26           })
27           // vm.stories = storiesReady
28           Vue.set(vm, 'stories', storiesReady)
29           // 또는: vm.stories = storiesReady
30           });
31     },
32   }
33});
```

---

이제, POST 요청이 성공하면 `fetchStories()`를 호출합니다.

```

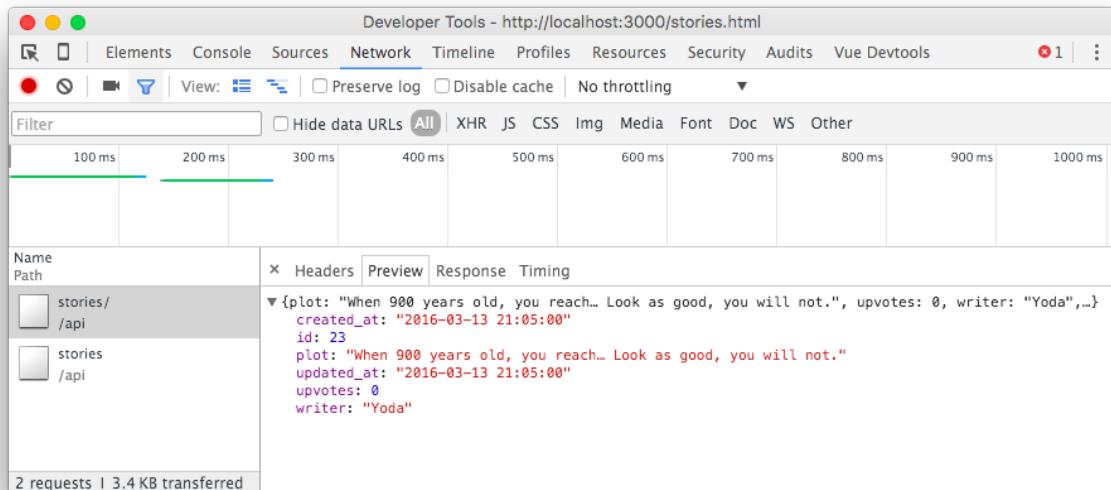
1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       axios.post('/api/stories/', story).then(function () {
7         story.editing = false;
8         vm.fetchStories();
9       });
10      },
11    }
12   ...
13 })

```

이게 전부입니다! 이제 원하는대로 이야기를 생성하고 수정할 수 있습니다.

## 단위별 저장과 수정

이전 문제를 수정하기 위한 더 나은 방법은 새로 만든 이야기만 데이터베이스에서 가져오는 것입니다. 모든 이야기를 가져올 필요는 없습니다. POST 요청에 대한 서버 응답을 확인하면 생성된 이야기와 함께 id를 반환합니다.



### 새 이야기를 만든 후 서버의 응답

서버의 내용과 일치하도록 이야기를 수정해야 합니다. 그러기 위해서는 id를 이야기의 속성으로 만듭니다. POST의 성공 콜백에서 이 작업을 해야 합니다.

```

1 Vue.component( 'story' ,{
2     ...
3     methods: {
4         ...
5         storeStory: function(story){
6             axios.post( '/api/stories/' , story).then(function (response) {
7                 Vue.set(story, 'id' , response.data.id);
8                 story.editing = false
9             });
10            },
11        }
12    ...
13 })

```

테이블에서 id를 보여주기 위해 `story.id = response.data.id` 대신에 `Vue.set(story, 'id' , response.data.id)`를 사용합니다. 새로운 스토리에는 id가 없기 때문에 stories 배열로 추가할 때 id가 변경되면 DOM이 수정되지 않으므로 추가된 id를 확인할 수 없습니다.



### 팁

데이터가 이미 관찰되고 있는 상태에서 새 속성을 추가하면 Vue.js는 이를 알아차리지 못합니다.

그래서 런타임에서 속성을 추가하거나 삭제할 때 전역 `Vue.set` 또는 `Vue.delete` 메소드를 사용해야 합니다

## 자바스크립트 파일

코드가 시작보다 많이 커졌습니다. 프로젝트가 커질수록 점점 더 관리하기 힘들어집니다. 이제 자바스크립트 코드를 HTML에서 분리해야 합니다.

`app.js` 파일을 만들고 `js` 디렉토리에 저장합니다.

모든 자바스크립트 코드는 지금부터 파일 안으로 옮깁니다. 새로 만든 스크립트를 HTML 페이지에 포함 하려면 이 태그를 추가하기만 하면 됩니다.

```
<script src= '/js/app.js' type="text/javascript"></script>
```

이제 모든 준비가 끝났습니다!

## 소스코드

아래는 이전 이야기 관리 예제의 전체 소스 코드입니다. 저장소에서 내려받은 경우에 코드가 매우 크기 때문에 즐겨 쓰는 텍스트 편집기에서 여는 것이 좋습니다.

파일은 `~/themajestyofvuejs2/apis/stories/public`에 있습니다. 저장소에서 내려받지 않은 경우에도 github의 [stories.html<sup>67</sup>](#)와 [app.js<sup>68</sup>](#)에서 확인할 수 있습니다.

`stories.html`

---

```

1 <html lang= " en " >
2 <head>
3     <title>Stories</title>
4     <link rel= " stylesheet " href= " https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/boots\ 
5 trap.min.css " >
6 </head>
7
8 <body>
9 <main>
10    <div class= " container " >
11        <h1>Stories</h1>
12        <div id= " v-app " >
13            <table class= " table table-striped " >
14                <tr>
15                    <th>#</th>
16                    <th>Plot</th>
17                    <th>Writer</th>
18                    <th>Upvotes</th>
19                    <th>Actions</th>
20                </tr>
21                <tr v-for= " story in stories " is= " story " :story= " story " ></tr>
22            </table>
23            <p class= " lead " >Here ' s a list of all your stories.
24                <button @click= " createStory() " class= " btn btn-primary " >
25                    Add a new one?
26                </button>
27            </p>
28            <pre>{{ $data }}</pre>
29        </div>
30    </div>
31 </main>
32 <template id= " template-story-raw " >
33     <tr>
```

<sup>67</sup> <https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html>

<sup>68</sup> <https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js>

```
34      <td>
35          {{story.id}}
36      </td>
37      <td class="col-md-6" >
38          <input v-if=" story.editing "
39              v-model=" story.plot "
40              class=" form-control " >
41          </input>
42          <!--그 밖의 경우 이야기의 plot을 출력 -->
43          <span v-else>
44              {{story.plot}}
45          </span>
46      </td>
47      <td>
48          <input v-if=" story.editing "
49              v-model=" story.writer " class=" form-control " >
50          </input>
51          <!--그 밖의 경우 이야기의 writer를 출력 -->
52          <span v-else>
53              {{story.writer}}
54          </span>
55      </td>
56      <td>
57          {{story.upvotes}}
58      </td>
59      <td>
60          <div class=" btn-group " v-if=" !story.editing " >
61              <button @click=" upvoteStory(story) "
62                  class=" btn btn-primary " >
63                  Upvote
64              </button>
65              <button @click=" editStory(story) " class=" btn btn-default " >
66                  Edit
67              </button>
68              <button @click=" deleteStory(story) "
69                  class=" btn btn-danger " >
70                  Delete
71              </button>
72          </div>
73          <div class=" btn-group " v-else>
74              <!--데이터베이스에서 가져온 이야기는 id를 가지고 있습니다-->
75              <button v-if=" story.id "
76                  class=" btn btn-primary "
77                  @click=" updateStory(story) " >
78                  Update Story
```

```

79          </button>
80
81          <!-- 새 이야기면 저장해야합니다-->
82          <button v-else class= "btn btn-success "
83             @click= "storeStory(story) " >
84              Save New Story
85          </button>
86
87          <!--취소 버튼은 항상 출력-->
88          <button @click= "story.editing=false "
89             class= "btn btn-default " >
90              Cancel
91          </button>
92      </div>
93  </td>
94 </tr>
95 </template>
96 <script src= " https://unpkg.com/vue@2.3.2/dist/vue.js " ></script>
97 <script src= " https://unpkg.com/axios/dist/axios.min.js " ></script>
98 <script src= '/js/app.js ' type= " text/javascript " ></script>
99 </body>
100 </html>

```

---

## app.js

```

1 Vue.component( ' story ' , {
2     template: '#template-story-raw ' ,
3     props: [ ' story ' ],
4     methods: {
5         deleteStory: function (story) {
6             var index = this.$parent.stories.indexOf(story);
7             this.$parent.stories.splice(index, 1)
8             axios.delete( '/api/stories/ ' + story.id)
9         },
10        upvoteStory: function (story) {
11            story.upvotes++;
12            axios.patch( '/api/stories/ ' + story.id, story)
13        },
14        editStory: function (story) {
15            story.editing = true;
16        },
17        updateStory: function (story) {
18            axios.patch( '/api/stories/ ' + story.id, story)
19            // editing을 false로 설정하여 입력을 숨기고 actions를 다시 표시합니다.
20            story.editing = false;

```

```
21     },
22     storeStory: function (story) {
23         axios.post( '/api/stories/ ', story).then(function (response) {
24             // 데이터베이스에 새 이야기를 저장한 후
25             // 다시 모든 이야기를 vm.fetchStories(); 로 가져옵니다.
26             // 더 좋은 방법은, 만든 이야기에 id를 추가하는 것 입니다.
27             // Vue.set(story, 'id', response.data.id);
28             // editing을 false로 설정하여 입력을 숨기고
29             // actions를 다시 표시합니다.
30             story.editing = false;
31         });
32     },
33 }
34 })
35
36 // Vue.prototype.$http = axios
37
38 new Vue({
39     el: '#v-app',
40     data: {
41         stories: [],
42         story: {}
43     },
44     mounted: function () {
45         this.fetchStories()
46     },
47     methods: {
48         createStory: function () {
49             var newStory = {
50                 plot: " ",
51                 upvotes: 0,
52                 editing: true
53             };
54             this.stories.push(newStory);
55         },
56         fetchStories: function () {
57             var vm = this;
58             axios.get( '/api/stories' )
59             .then(function (response) {
60                 // vm에 data를 설정
61                 var storiesReady = response.data.map(function (story) {
62                     story.editing = false;
63                     return story
64                 })
65                 // vm.stories = storiesReady
```

```
66             Vue.set(vm, 'stories', storiesReady)
67         });
68     },
69 }
70});
```

---



## Code Examples

이 장에서 사용된 예제 코드는 [GitHub<sup>69</sup>](#)에 있습니다.

## 혼자 해보기

웹 요청을 사용하고 응답을 처리하는데 익숙해지려면 이번 장에서 했던 것들을 다시 해보아야 합니다. API를 이용해 다음 작업을 해야 합니다.

- 새 테이블을 만들고 이미 존재하는 영화 목록을 출력하세요
- 이미 있는 영화를 수정할 수 있어야 합니다
- 데이터베이스에 새 영화 정보를 저장해야 합니다.
- 데이터베이스에서 영화를 삭제할 수 있어야 합니다.

데이터베이스와 API를 이미 준비해놓았습니다. HTML과 자바스크립트만 작성하면 됩니다.

## 사전 작업

[10장](#)의 지시사항에 따라 터미널을 열고 다음 명령어를 실행하세요.

```
>_ cd ~/themajestyofvuejs/apis/movies  
     sh setup.sh
```

할 수 없다면 아래 명령어를 실행해야 합니다.

```
>_ mkdir ~/themajestyofvuejs  
     cd ~/themajestyofvuejs  
     git clone https://github.com/hootlex/the-majesty-of-vuejs .  
     cd ~/themajestyofvuejs/apis/movies  
     sh setup.sh
```

<http://localhost:3000>에서 실행되는 서버와 멋진 영화를 가지고 있는 데이터베이스를 가지게 되었습니다.

<http://localhost:3000/api/movies>에서 JSON 형식의 영화 배열을 보면 모든 것이 잘 작동하는 것 입니다.

---

<sup>69</sup> <https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter12>

## API 엔드포인트

작업을 하기 위한 API 엔드포인트입니다.

HTTP Method	URI	Action
GET/HEAD	api/movies	모든 영화를 가져옵니다
GET/HEAD	api/movies/{id}	특정 영화를 가져옵니다
POST	api/movies	새 영화를 만듭니다
PUT/PATCH	api/movies/{id}	존재하는 영화를 수정합니다
DELETE	api/movies/{id}	특정 영화를 삭제합니다

## Your Code

Put your HTML code inside `~/themajestyofvuejs2/apis/movies/public/movies.html` file we have created. You can place your JavaScript code there too, or inside `js/app.js`.

To check your work visit `http://localhost:3000/movies.html` on your browser.

HTML 코드를 `~/themajestyofvuejs2/apis/movies/public/movies.html`에 넣으세요 자바스크립트도 `js/app.js`에 넣어야 합니다.

그리고 브라우저에서 `http://localhost:3000/movies.html` 접속하여 작동하는지 확인하세요!

이 작업을 즐기길 바랍니다. 행운을 빌어요!

#	Title	Director	Actions
2	Snatch.	Guy Ritchie	<button>Edit</button> <button>Delete</button>
3	The Lord of the Rings: The Fellowship of the Ring	Peter Jackson	<button>Edit</button> <button>Delete</button>
4	The Grand Budapest Hotel	Wes Anderson	<button>Edit</button> <button>Delete</button>
5	Fight Club	David Fincher	<button>Edit</button> <button>Delete</button>
6	Million Dollar Baby	Clint Eastwood	<button>Edit</button> <button>Delete</button>
7	Fight Club	David Fincher	<button>Edit</button> <button>Delete</button>
8	Aladdin	John Musker	<button>Edit</button> <button>Delete</button>
9	The Wolf of Wall Street	Martin Scorsese	<button>Edit</button> <button>Delete</button>
21	batman	ego	<button>Edit</button> <button>Delete</button>

Here's a list of all your movies. [Add a new one?](#)

### Example Output



## 해결방법의 예

이 문제의 잠재적인 해결 방법은 [예제](#)<sup>70</sup>를 참조하세요.

---

<sup>70</sup> <https://github.com/hoottex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12>

# 페이지네이션

이전 장에서 데이터베이스에서 모든 레코드를 가져와 테이블에 표시했습니다. 이 구현은 적은 양의 데이터에는 적합하지만 실제로 수천에서 수백만개의 데이터를 사용하는 경우, 단순히 데이터를 가져와 배열에 보관하기는 힘듭니다. 이 방법으로는 브라우저에서 많은 양의 데이터를 불러오는 것이 어렵습니다. 가능은 하겠지만 사용자의 관점에서 100,000개의 행을 가진 테이블을 다루는 것은 무리입니다.

## 정보

페이지네이션은 받은 데이터를 나누어 여러 페이지에 표시하는 거의 모든 웹 애플리케이션에서 사용하는 방법입니다. 또한 다양한 페이지에 대한 링크를 가지고 표시하는 로직을 포함하여 클라이언트와 서버측에서 처리할 수 있습니다. 서버측 페이지네이션이 더 일반적으로 사용됩니다.

이런 상황에서 API를 설계한 개발자는 반환될 데이터를 여러 페이지로 나누어야 합니다. HTTP 응답에는 전체 갯수, 페이지별 항목에 대해 알려주는 간단한 메타 데이터가 포함됩니다. 페이지를 순회하기 위해 **현재 페이지**, **다음 페이지** 그리고 **이전 페이지**와 같은 정보를 제공합니다.

페이지네이션 데이터에 대한 응답

```
{  
    "total": 10000,  
    "per_page": 50,  
    "current_page": 15,  
    "last_page": 200,  
    "next_page_url": "/api/stories?page=16",  
    "prev_page_url": "/api/stories?page=14",  
    "from": 751,  
    "to": 800,  
    "data": [...]  
}
```

페이지네이션의 메타 데이터는 **data** 또는 API 개발자가 선택한 객체 내부에 있어야 합니다.

## 페이지네이션 데이터에 대한 응답

---

```
{
  "data": [...],
  "pagination": {
    "total": 10000,
    "per_page": 50,
    "current_page": 15,
    "last_page": 200,
    "next_page_url": "/api/stories?page=16",
    "prev_page_url": "/api/stories?page=14",
    "from": 751,
    "to": 800,
  }
}
```

---

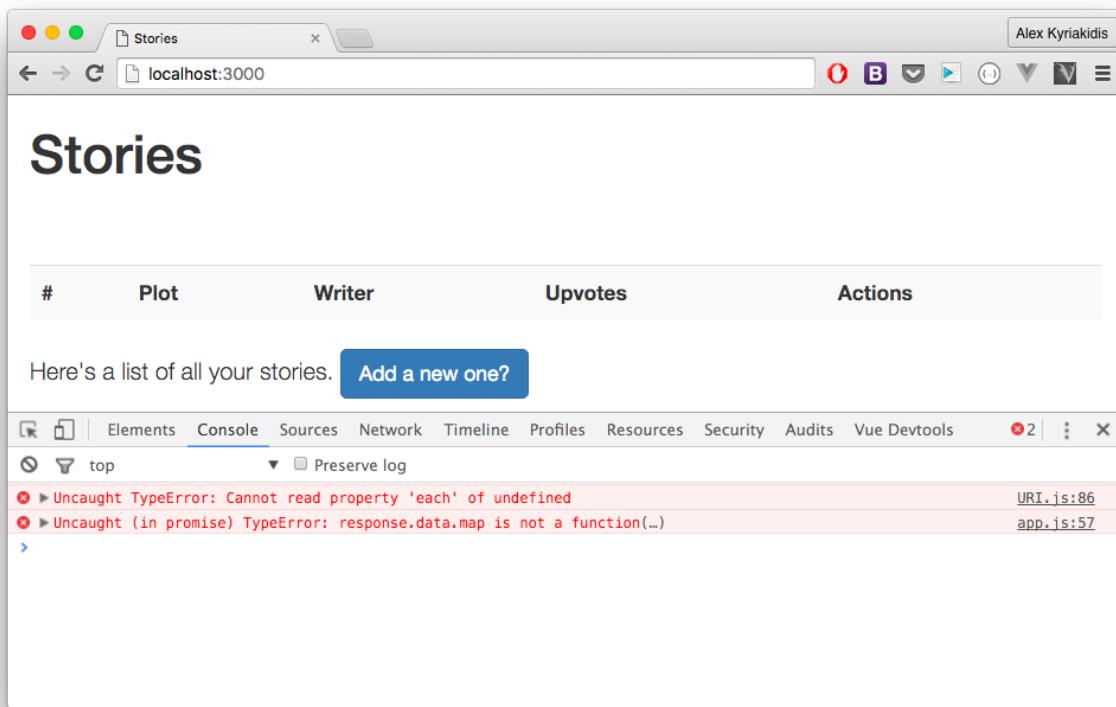
## 구현

이전 장의 story 예제를 사용하여 약간 개선된 페이지네이션 API를 추가합니다. 이 데이터를 사용할 수 있도록 코드를 수정해야 합니다.

이전 예제의 코드를 살펴보면 `fetchStories` 메소드가 다음과 같습니다.

```
new Vue({
  ...
  methods: {
    ...
    fetchStories: function () {
      var vm = this;
      this.$http.get( '/api/stories' )
        .then(function (response) {
          var storiesReady = response.data.map(function (story) {
            story.editing = false;
            return story
          })
          Vue.set(vm, 'stories', storiesReady)
        });
    },
    ...
  }
});
```

브라우저에서 HTML 파일을 열면 당연히 이 테이블이 제대로 렌더링 되지 않습니다.



이야기가 표시되지 않습니다

이유는 `stories`가 `data` 배열 안에 반환되기 때문입니다. 이 문제를 해결하려면 `response.data`에서 `response.data.data`로 변경해야 합니다(네. 다소 어색한 부분이 있습니다). `stories` 배열을 제외하고 페이지네이션 기능을 쉽게 구현하기 위해 객체 내에 페이지네이션에 대한 데이터를 저장해야 합니다.

이러한 데이터에 어떻게 액세스할 수 있는지 알기 위해 서버의 응답을 살펴봅니다.

```

{
  "total": 188,
  "per_page": 15,
  "current_page": 1,
  "last_page": 13,
  "next_page_url": "http://localhost:3000/api/stories?page=2",
  "prev_page_url": null,
  "from": 1,
  "to": 15,
  "data": [
    {
      "id": 1,
      "plot": "Mmm. Lost a planet, Master Obi-Wan has. How embarrassing.",
      "upvotes": 883,
      "writer": "Yoda",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    {
      "id": 2,
      "plot": "Don't call me a mindless philosopher, you overweight glob of grease.",
      "upvotes": 475,
      "writer": "C-3PO",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    {
      "id": 3,
      "plot": "Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!",
      "upvotes": 427,
      "writer": "Count Dooku",
      "created_at": "2016-04-10 17:47:03",
      "updated_at": "2016-04-10 17:47:03"
    },
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
    ▶ { ... }, // 6 items
  ]
}

```

### 서버 응답

우선, 모든 데이터가 필요하지는 않습니다. 그래서 `current_page`, `last_page`, `next_page_url` 그리고 `prev_page_url`를 계속 사용합니다.

페이지네이션 객체는 다음과 같습니다.

```

pagination: {
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14"
}

```

`fetchStories` 메소드를 수정하여 `pagination` 객체를 데이터베이스에서 가져올 때마다 수정해야합니다.

```

new Vue({
  ...
  methods: {
    ...
    fetchStories: function () {
      var vm = this;
      this.$http.get( '/api/stories' )
        .then(function (response) {
          var storiesReady = response.data.data.map(function (story) {
            story.editing = false;
            return story
          })
          //response.data를 사용합니다
          var pagination = {
            current_page: response.data.current_page,
            last_page: response.data.last_page,
            next_page_url: response.data.next_page_url,
            prev_page_url: response.data.prev_page_url
          }
          Vue.set(vm, 'stories', storiesReady)
          Vue.set(vm, 'pagination', pagination)
        });
    },
    ...
  }
});
  
```

## 페이지네이션 링크

**pagination** 객체를 가지고 있어도 api/stories에 HTTP GET 요청을 하기 때문에 항상 stories의 첫 페이지를 가져옵니다. 사용자의 상호작용(다음 페이지, 이전 페이지)에 따라 페이지를 변경해야 합니다.

먼저 **fetchStories** 메소드를 수정하여 원하는 페이지 전달인자를 받도록 합니다. 전달인자가 없는 경우 첫번째 페이지를 가져옵니다. 또한 새 메소드인 **makePagination**을 만들어 코드를 조금 더 깔끔하게 만듭니다.

```

new Vue({
  ...
  methods: {
    ...
    fetchStories: function (page_url) {
      var vm = this;
      page_url = page_url || '/api/stories'
      this.$http.get(page_url)
        .then(function (response) {
          var storiesReady = response.data.data.map(function (story) {
            story.editing = false;
            return story
          })
          vm.makePagination(response.data)
          Vue.set(vm, 'stories', storiesReady)
        });
    },
    makePagination: function (data){
      //response.data를 사용합니다
      var pagination = {
        current_page: data.current_page,
        last_page: data.last_page,
        next_page_url: data.next_page_url,
        prev_page_url: data.prev_page_url
      }
      Vue.set(vm, 'pagination', pagination)
    }
    ...
  }
})

```

메소드가 준비되었으므로 적절하게 호출할 방법이 필요합니다. #app div 상단에 다음 페이지와 이전 페이지 용 버튼 2개가 추가 됩니다. 각 버튼을 클릭하면 해당 page\_url을 전달하여 **fetchStories** 메소드를 호출합니다.

```

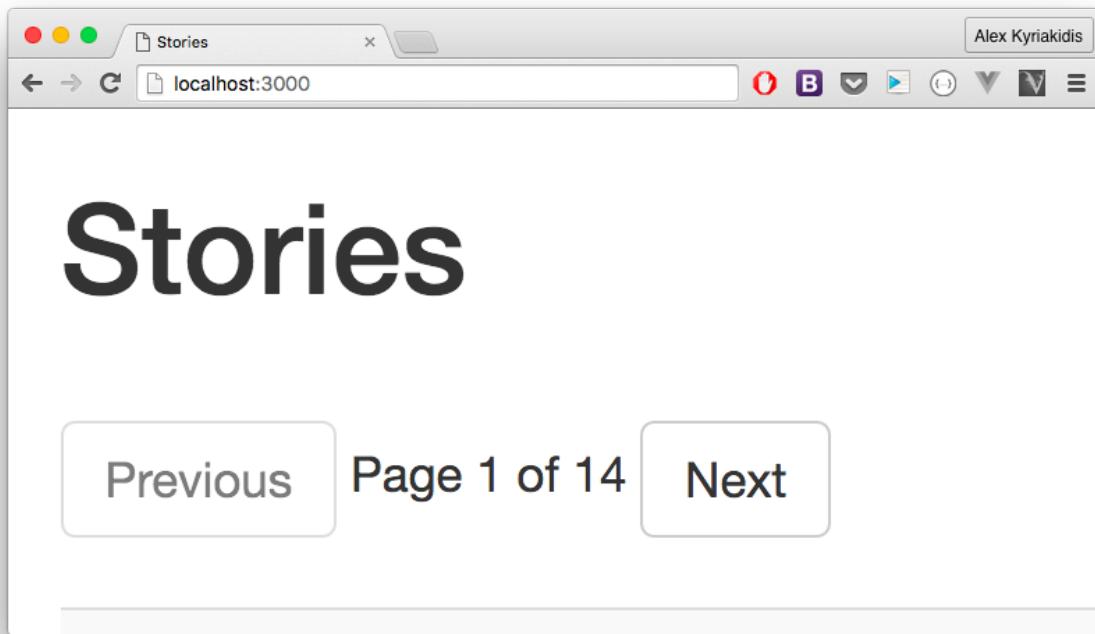
1 <div class= "pagination" >
2   <button @click= "fetchStories(pagination.prev_page_url)" >
3     Previous
4   </button>
5   <button @click= "fetchStories(pagination.next_page_url)" >
6     Next
7   </button>
8 </div>

```

대단해요! 버튼을 클릭하면 예상대로 잘 작동하는 것을 볼 수 있습니다. 눈 깜짝할 사이에 페이지네이션을 완성했습니다. 이제 사용자에게 현재 보고 있는 페이지와 총 페이지 수를

알려주어야 합니다. 또한 사용자가 첫번째 페이지에 있을 때 이전 버튼을 비활성화하고 마지막 페이지에서 다음 버튼을 비활성화해야합니다.

```
1 <div class= "pagination" >
2   <button @click= "fetchStories(pagination.prev_page_url)" 
3     :disabled= "!pagination.prev_page_url"
4   >
5     Previous
6   </button>
7   <span>Page {{pagination.current_page}} of {{pagination.last_page}}</span>
8   <button @click= "fetchStories(pagination.next_page_url)"
9     :disabled= "!pagination.next_page_url"
10  >
11    Next
12  </button>
13 </div>
```



비활성화 된 이전 버튼



## 예제 코드

이 장에서 사용된 예제 코드는 [GitHub](#)<sup>71</sup>에 있습니다.

<sup>71</sup><https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter13>

## 혼자 해보기

이 장에서 특별히 더 해야할 것은 없습니다. 실제로 이 예제를 사용하려면 페이지네이션 API와 함께 해야합니다.

이전 장의 문제를 풀고(코드를 내려받아 서버를 시작한 경우) 몇 번의 클릭만 하면 됩니다. 그렇지 않은 경우에 [다음 안내](#)를 따라가세요.

페이지네이션 API는 '/themajestyofvuejs2/apis/pagination/stories'에 있습니다.

최종 코드를 확인하려면 [Github](#)<sup>72</sup>을 살펴보세요.

---

<sup>72</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/apis/pagination/stories/public>

# 대규모 애플리케이션 구축

# ECMAScript 6

한 걸음 더 나아가 대규모 애플리케이션을 구축하는 방법으로 들어가기 전에 ECMAScript 6에 익숙해져야합니다.

## 정보

ECMAScript는 JavaScript, ActionScript 및 JScript를 비롯한 여러 프로그래밍 언어의 기초가 되는 클라이언트 측 스크립팅 언어의 사양입니다.

ES2015라고 알려진 ECMAScript 6(ES6)는 ECMAScript 표준의 최신 버전입니다. 2015년 6월에 ES6의 사양이 완성되었습니다. ES5는 2009년에 표준화된 이후로 언어에 대한 첫 주요 업데이트입니다. 주요 자바스크립트 엔진에서 ES6 기능을 [구현하는 중](#)<sup>73</sup>입니다.

## 소개

ES6에서 수많은 새로운 기능이 추가되었습니다. 이번에는 다음 장에서 사용할 부분을 다룹니다. ES6의 새로운 기능에 대한 자세한 내용은 [leanpub](#)<sup>74</sup>에 있는 Nicholas C. Zakas의 책 “Understanding ECMAScript 6”을 적극 권장합니다. 또한 [온라인 버전](#)<sup>75</sup>을 무료로 읽을 수 있습니다.

또한 [Babel](#)<sup>76</sup>, [tutsplus](#)의 글<sup>77</sup>과 같은 유용한 리소스와 자습서가 있습니다. Nicholas C. Zakas의 [블로그](#)<sup>78</sup> 글과 수많은 웹의 글을 읽어보세요!

---

<sup>73</sup><http://kangax.github.io/compat-table/es6/>

<sup>74</sup><https://leanpub.com/understandinges6/>

<sup>75</sup><https://leanpub.com/understandinges6/read>

<sup>76</sup><https://babeljs.io/docs/learn-es2015/>

<sup>77</sup><http://code.tutsplus.com/articles/use-ecmascript-6-today--net-31582>

<sup>78</sup><https://www.nczonline.net/blog/2013/09/10/understanding-ecmascript-6-arrow-functions/>

## 호환성

당연하게도 브라우저에서 지원해야 사용할 수 있습니다. 모질라는 가장 앞서서 지원해줍니다. ES6 호환성 표<sup>79</sup>는 브라우저가 지원하는 ECMAScript 6 기능을 살펴보는데 좋은 시작입니다.



### 노트

크롬 브라우저를 사용하는 경우 대부분의 ES6 기능은 켜고 끌 수 있습니다. chrome://flags에 접속하여, “실험용 자바스크립트” 섹션을 찾아 사용하도록 설정해야합니다.

이제부터 ES6 기능을 사용하여 예제를 만듭니다.

## 변수 선언

### Let 선언

**let**은 **var**의 새로운 버전입니다. 기본적으로 **var**를 **let**으로 바꾸면 변수를 선언할 수 있습니다. 변수의 범위는 현재 코드 블럭 안으로 제한됩니다. **let** 변수는 감싸진 블럭의 맨 위로 호이스팅되지 않기 때문에, **let** 변수 선언을 블록 안에서 먼저 하는 것이 좋습니다. 예제입니다.

if 안의 let

---

```

1 let age = 22
2 if (age >= 18) {
3   let adult = true;
4   console.log(adult); // true 출력
5 }
6 // adult는 여기서 접근할 수 없습니다
7 console.log(adult);
8 //ERROR: Uncaught ReferenceError: adult is not defined

```

---

<sup>79</sup> <https://kangax.github.io/compat-table/es6/>

### 맨 위의 let

---

```
1 let age = 22
2 let adult
3 if (age >= 18) {
4     adult = true;
5     console.log(adult); // true 출력
6 }
7 // 이제 adult에 접근할 수 있습니다
8 console.log(adult); // true 출력
```

---

## 상수 선언

`let` 선언과 마찬가지로 상수는 블럭 단위의 범위를 가집니다. `let`과 `const`는 큰 차이가 있습니다. `const`를 선언하면 상수가 됩니다. 상수는 값을 더이상 변경할 수 없습니다.

```
1 const name = "Alex"
2
3 name = "Kostas" //throws error
```



### 정보

많은 다른 언어의 상수와 마찬가지로 한번 설정하면 변경할 수 없습니다. 그러나 다른 언어들과 다른 점은 객체의 경우 값을 수정할 수 있습니다.

## 화살표 함수

ECMAScript 6의 가장 흥미로운 부분 중 하나는 화살표 함수입니다. 화살표 함수란 함수를 정의할 때 “화살표”( $\Rightarrow$ )를 사용하는 새 문법입니다. 표현식과 구문을 모두 제공합니다. 화살표 함수는 일반 함수와 달리 주변 코드와 동일한 `this`를 공유합니다.

예를 들어, 다음 화살표 함수는 하나의 전달인자를 가지고 1씩 증가된 값을 반환합니다.

```
var increment = value => value + 1;
increment(5) // 6 반환
```

// 동일합니다

```
var increment = function(value) {
    return value + 1;
};
```

또 다른 예제는 두개의 전달인자를 가지고 그 합을 반환하는 화살표 함수 입니다.

```
var sum = (a, b) => a + b;
sum(5, 10) // 15 반환
```

// 동일합니다

```
var sum = function(a, b) {
    return a + b;
};
```

이번에는 화살표 함수가 전달인자 없이 작동하는 예제 입니다.

```
var sayHiAndBye = () => {
    console.log('Hi!');
    console.log('Bye!');
};
```

sayHiAndBye()

// 동일합니다

```
var sayHiAndBye = function() {
    console.log('Hi!');
    console.log('Bye!');
};
```

## 모듈

모듈은 자바스크립트에서 가장 향상된 기능입니다. ES6는 이제 여러 파일에서 모듈을 내보내고 가져올 수 있는 기능을 지원합니다.

가장 단순한 예제로 .js 파일을 만들고 다른 파일에서 사용합니다.

---

module.js

---

```
1 export name = 'Alex'
```

---



---

main.js

---

```
1 import {name} from './module'
2 console.log('Hello', name)
3 // "Hello Alex" 출력
```

---

함수와 함께 변수를 하나씩 내보낼 수도 있습니다.

---

module.js

---

```
1 export var name = 'Alex'
2 export function getAge(){
3     return 22;
4 }
```

---



---

main.js

---

```
1 import {name, getAge} from './module'
2 console.log(name, 'is', getAge())
```

---

또한 객체 내부에서 내보낼 수 있습니다.

---

module.js

---

```
1 var name = 'Alex'
2 function getAge(){
3     return 22;
4 }
5 export default {name, age}
```

---



---

main.js

---

```
1 import person from './module'
2 console.log(person.name, 'is', person.getAge())
3 // "Alex is 22" 출력
```

---

## 클래스

자바스크립트의 클래스는 ECMAScript 6에서 도입되었으며 자바스크립트 프로토타입 상속에 대한 선택적 슈가입니다. 클래스 구문은 자바스크립트에 객체지향 상속 모델을 도입하지 않았습니다. 자바스크립트 클래스는 객체를 만들고 상속을 처리하는 훨씬 간단하고 명확한 문법을 제공합니다.

### 클래스 예제

---

```
// 부모 클래스
class Rectangle {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }

    calcArea() {
        return this.height * this.width;
    }

    // getter를 만듭니다 get 키워드를 사용합니다.
    get area() {
        return this.calcArea();
    }
    //setter를 만듭니다 set 키워드를 사용합니다
}

// 자식 클래스
class Square extends Rectangle{
    constructor(side) {
        // 부모 생성자 호출
        super(side, side)
    }
}

var square = new Square(5);

console.log(square.area); // 25 출력
```

---

## 전달인자 기본값

ES6를 사용하면 전달인자에 기본 값을 지정할 수 있습니다.

```
function divide(x, y = 2){
    return x/y;
}

// 동일합니다

function divide(x, y){
    y = y == undefined ? 2 : y;
    return x/y;
}
```

## 템플릿 리터럴

템플릿 리터럴은 포함된 표현식을 허용하는 문자열 리터럴입니다. 여러 줄 문자열과 보간 기능이 있습니다. 이전 버전의 ES2015/ES6 사양에는 “템플릿 문자열”로 불렸습니다.

템플릿 리터럴은 큰 따옴표나 작은 따옴표 대신 백틱(‘)을 사용합니다. 백틱 안에서 \${표현식}을 사용할 수 있습니다. 표현식은 함수 또는 변수입니다.

템플릿 리터럴 – 변수 사용

---

```
let name = 'Alex';
console.log('Hello ${name} ')

// 동일합니다
console.log('Hello ' + name)
```

---

템플릿 리터럴 – 함수 사용

---

```
add = (a, b) => a + b

let [a, b] = [10, 2]

console.log('If you have ${a} eggs and you buy ${b}
more you\'ll have ${add(a,b)} eggs! ')

// 동일합니다
console.log('If you have ' + a + ' eggs and you buy ' + b +
' more you\'ll have ' + add(a,b) + ' eggs! ')
```

---

### 템플릿 리터럴 – 표현식 사용

```
let [a, b] = [10, 2]
```

```
console.log('If you have ${a} eggs and you buy ${b}  
more you\'ll have ${a + b} eggs!')
```

// 동일합니다

```
console.log('If you have ' + a + ' eggs and you buy ' + b +  
'\nmore you\'ll have ' + (a + b)*1 + ' eggs!')
```

'\n' 을 사용하여 여러 줄로 메시지를 나눌 수 있습니다.

# 고급 워크플로우

살펴본 모든 ES6 기능(혹은 더 많은 것들)을 보고 기뻐할만 하지만 앞에서 언급한 것처럼 모든 브라우저에서 ES6/ES2015를 완벽하게 지원하지는 않습니다.

새로운 자바스크립트 문법으로 코드를 작성하려면 모든 브라우저에서 사용할 수 있는 [Vanilla JS<sup>80</sup>](#)로 변형해줄 도구가 필요합니다. 이를 생각하지 않았더라도 실제 프로덕션 환경에서는 매우 중요합니다.

저의 이야기를 하자면 몇년 전, 한 동료가 모든 브라우저에서 완벽하게 지원하지 않는 멋진 자바스크립트 기능을 사용하기 시작했습니다. 며칠이 지나 사용자들로부터 웹페이지의 일부가 제대로 표시되지 않는다는 항의가 들어왔지만 이유를 찾을 수 없었습니다. PC, 안드로이드, 아이폰 등에서 테스트했지만 모두 100% 동작했습니다. 나중에 구버전 모바일 사파리에서 지원하지 않았다는 사실을 알게 되었습니다. 이런일이 일어나면 안됩니다!

작성한 코드들이 페이스북의 모바일 브라우저를 포함한 모든 브라우저에서 잘 작동하는지 확인하는 것이 어려울 때가 많습니다. 이러한 상황이 가장 두려워집니다.

## Babel을 이용한 ES6 컴파일

Babel은 중개자입니다. 바벨은 최신 자바스크립트 소스를 사용할 수 있도록 도와주는 자바스크립트 컴파일러입니다.

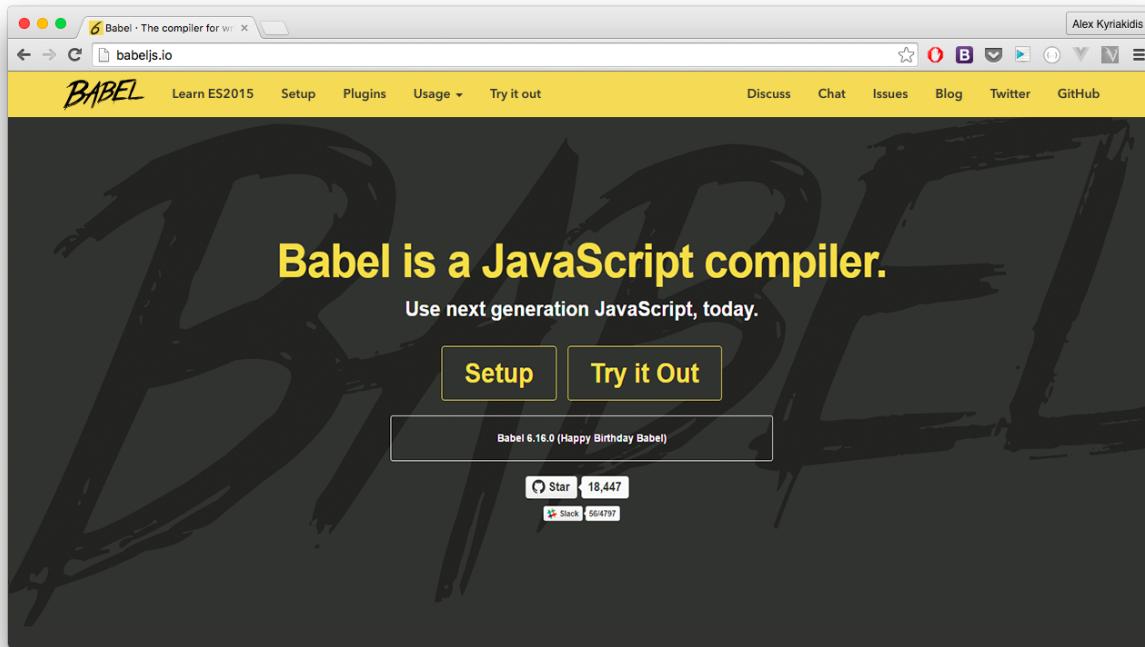


### 정보

소스에서 다른 소스로 컴파일해주는 컴파일러는 트랜스컴파일러 혹은 트랜스파일러라고 부릅니다. 이는 한 프로그래밍 언어로 작성된 코드를 입력받으면 다른 프로그램언어를 만들어주는 컴파일러입니다

---

<sup>80</sup> <http://vanilla-js.com/>

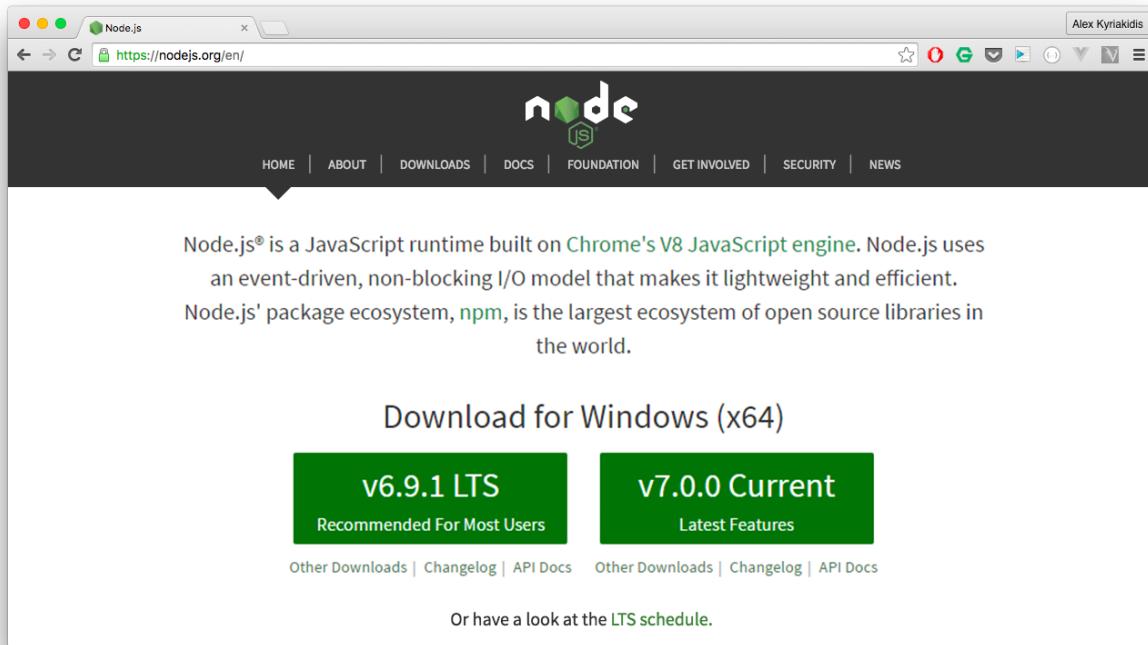


## Babel

Babel을 설치하기 전에 Node.js를 우선 설치해야 합니다. 이를 위해, [Node 공식 웹사이트<sup>81</sup>](https://nodejs.org/en/)에 들어가 최신 안정화 다운로드 버튼을 누르세요 맥의 경우 .pkg, 윈도우의 경우 \*.msi 파일을 내려받을 수 있습니다. 내려받기가 완료되면, 파일을 열어 안내에 따라 설치하세요. 재시작이 필요한 경우 재시작을 하면 완료됩니다!

---

<sup>81</sup><https://nodejs.org/en/>



Node.js

## 설치

새 디렉터리를 만들고 **package.json** 파일을 만드세요. 그리고 빈 JSON 객체({})를 추가하세요. 이 과정을 직접 하거나 아래의 지시에 따라 터미널에서 아래 명령어를 이용할 수 있습니다.

```
>_ mkdir babel-example  
     echo {} > package.json
```

Babel을 설치하기 위해 아래 명령어를 입력하세요.

```
>_ npm install babel-cli - save-dev
```

```

alex@192: ~
└─ path-is-absolute@1.0.0
  └─ convert-source-map@1.2.0
  └─ commander@2.9.0 (graceful-readlink@1.0.1)
  └─ v8flags@2.0.11 (user-home@1.1.1)
  └─ source-map@0.5.6
  └─ chalk@1.1.1 (escape-string-regexp@1.0.5, supports-color@2.0.0, ansi-styles@2.2.1, strip-ansi@3.0.1, has-ansi@2.0.0)
  └─ glob@5.0.15 (inherits@2.0.1, once@1.3.3, inflight@1.0.4, minimatch@3.0.0)
  └─ output-file-sync@1.1.1 (xtend@4.0.1, mkdirp@0.5.1)
  └─ request@2.72.0 (tunnel-agent@0.4.3, aws-sign2@0.6.0, oauth-sign@0.8.2, forever-agent@0.6.1, is-typedarray@1.0.0, caseless@0.11.0, stringstream@0.0.5, aws4@1.4.1, isstream@0.1.2, json-stringify-safe@5.0.1, extend@3.0.0, tough-cookie@2.2.2, node-uuid@1.4.7, qs@6.1.0, combined-stream@1.0.5, mime-types@2.1.11, form-data@1.0.0-rc4, bl@1.1.2, hawk@3.1.3, http-signature@1.1.1, har-validator@2.0.6)
  └─ babel-core@6.8.0 (babel-messages@6.8.0, shebang-regex@1.0.0, babel-template@6.8.0, babel-helpers@6.8.0, private@0.1.6, babel-code-frame@6.8.0, debug@2.2.0, babylon@6.8.0, minimatch@2.0.10, babel-types@6.8.1, babel-generator@6.8.0, babel-traverse@6.8.0, json5@0.4.0)
  └─ bin-version-check@2.1.0 (minimist@1.2.0, semver@4.3.6, semver-truncate@1.1.0, bin-version@1.0.4)
  └─ lodash@3.10.1
  └─ babel-register@6.8.0 (home-or-tmp@1.0.0, mkdirp@0.5.1, source-map-support@0.2.10, core-js@2.4.0)
  └─ babel-polyfill@6.8.0 (babel-regenerator-runtime@6.5.0, core-js@2.4.0)
  └─ babel-runtime@6.6.1 (core-js@2.4.0)
  └─ chokidar@1.5.0 (inherits@2.0.1, glob-parent@2.0.0, async-each@1.0.0, is-glob@2.0.1, is-binary-path@1.0.1, readdirp@2.0.0, anymatch@1.3.0, fsevents@1.0.12)

~ »

```

터미널 출력

완료되면 package.json 파일의 내용은 다음과 같아집니다.

package.js

---

```
{
  "devDependencies": {
    "babel-cli": "^6.18.0"
  }
}
```

---



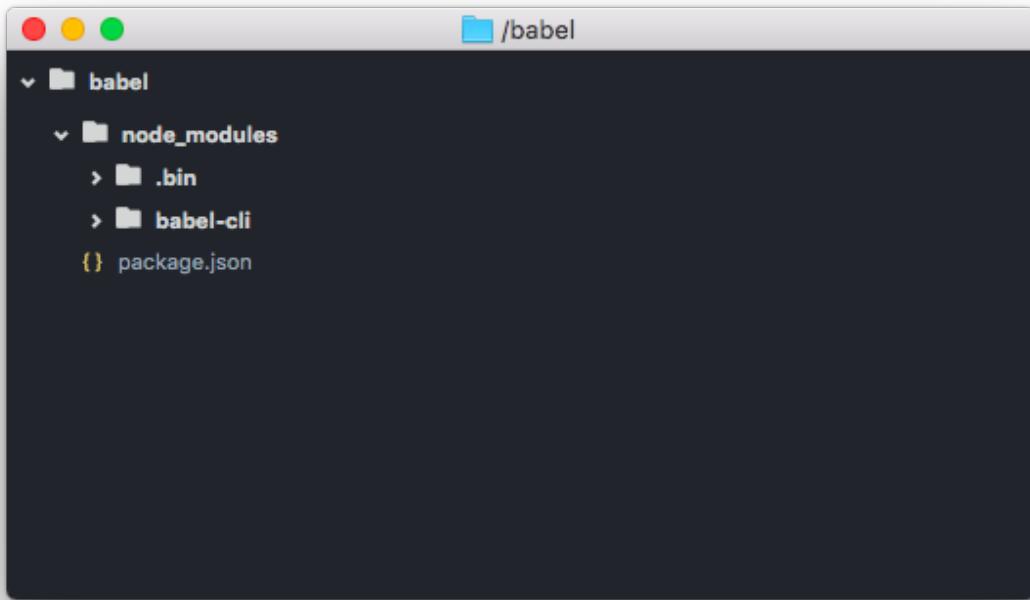
## package.json이 무엇입니까?

package.json 파일에는 앱 또는 모듈에 대한 메타데이터가 포함됩니다. 가장 중요한 점은 **npm install**을 실행할 때 npm에서 설치할 의존성 목록을 포함하는 것입니다. 컴포저(Composer)에 익숙하면 composer.json과 유사하다고 생각하면 됩니다.

package.json에 대한 더 많은 정보를 얻기 위해 [npm 문서](#)<sup>82</sup>를 읽어보세요

프로젝트의 디렉터리 구조는 다음과 같아야 합니다.

<sup>82</sup><https://docs.npmjs.com/files/package.json>



프로젝트 디렉터리

## 설정

이제 Babel을 설치했으므로 빌드에서 실행할 소스코드 변환 설정을 해야 합니다. ES2015 코드를 변환해야 하므로 [ES2015-Preset<sup>83</sup>](#)를 설치합니다.

추가로 프리셋을 위한 설정파일(.babelrc)을 생성합니다.

```
>_ npm install babel-preset-es2015 - save-dev
echo { "presets": [ [ "es2015" ] ] } > .babelrc
```



### 팁

두 번째 명령어가 실패하면 다음과 같이 파일 내용을 따옴표로 둑어야 합니다.

```
echo '{ "presets": [ [ "es2015" ] ] }' > .babelrc
```

<sup>83</sup><https://babeljs.io/docs/plugins/preset-es2015/>

## 빌드에 대한 별칭

Babel을 직접 사용하지 않고 npm의 scripts를 이용합니다.

**scripts** 필드를 **package.json** 파일에 추가하고 **babel** 명령어를 **build**처럼 등록하면 됩니다. 지금까지 작성한 **package.json**의 내용입니다.

package.json

---

```
{
  "scripts": {
    "build": "babel src -d assets/js"
  },
  "devDependencies": {
    "babel-cli": "^6.8.0",
    "babel-preset-es2015": "^6.18.0"
  }
}
```

---

별칭처럼 사용할 수 있습니다. **npm run build**를 실행하면 **babel src -d assets/js** 명령어가 실행됩니다. 이 명령어는 **src** 디렉터리에서 **assets/js** 디렉터리로 코드를 추출하도록 지시합니다.

**build** 명령을 실행하기 전에 몇가지 더 해야합니다. 우선, 위에서 다룬 디렉터리들(src와 assets/js)을 만드세요.

## 사용법

앞으로 src안에서 파일을 만들어 작업합니다. 간단한 **sum** 함수를 가지는 **sum.js** 파일을 만듭니다.

src/sum.js

---

```
const sum = (a, b) => a + b;
console.log(sum(5,3));
```

---

이게 전부입니다. 이제 실행해봅니다.

>\_ npm run build

이를 실행하면 터미널에서 **src\sum.js** 파일이 **assets\js\sum.js**로 컴파일 되었고 아래처럼 보이게 됩니다.

assets/js/sum.js

---

```
"use strict";

var sum = function sum(a, b) {
    return a + b;
};
console.log(sum(5, 3));
```

---

이제부터 ES6코드를 컴파일할 때마다 **build**를 실행하면 됩니다. 멋지지 않습니까?

**sum.js** 파일의 결과를 브라우저에서 볼 차례입니다. **sum.html** 파일을 만들고 js파일을 추가합니다.

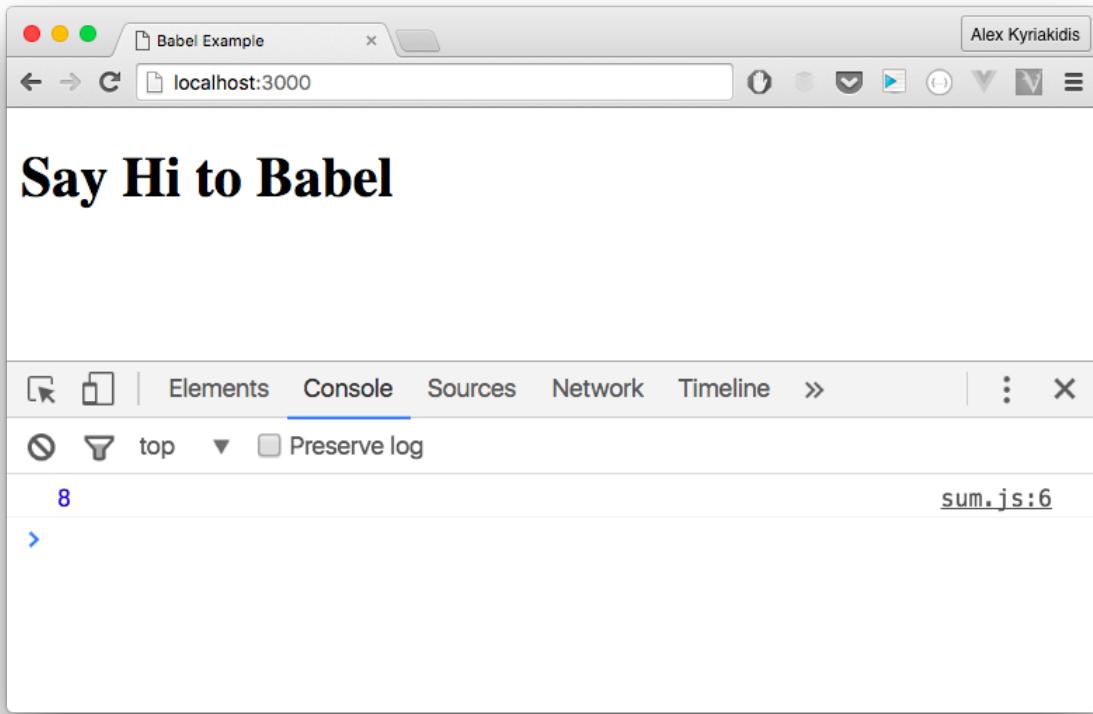
sum.html

---

```
<!DOCTYPE html>
<html>
<head>
    <title>Babel Example</title>
</head>
<body>
    <h1>Babel Example</h1>

    <script src= "assets/js/sum.js" ></script>
</body>
</html>
```

---



### 브라우저 출력

보시다시피, `sum` 함수의 결과는 콘솔에 잘 출력됩니다.



### 정보

js 파일을 테스트하고 싶지만 브라우저에서 열고 싶지 않으면 Node.js를 이용할 수 있습니다.

`sum.js` 예제에서 `console.log(sum(5,3))` 라인이 있으므로 터미널에 `node sum.js`를 입력하면 결과 8이 출력됩니다.

## 혼자 해보기

이번 과제는 방금 만들었던 예제를 재현하여 배운 것을 기억하는 것을 목표로 합니다. `sum.js` 대신에 ES6 클래스를 사용하여 `Ninja` 클래스를 포함하는 `Ninja.js` 파일을 만듭니다.

`Ninja`는 속성 `name`과 메소드 `announce`를 가지고 있어 닌자가 있음을 경고합니다.

**예제**

---

```
new Ninja( 'Leonardo' ).announce()  
//alerts "Ninja Leonardo is here!"
```

---

자바스크립트를 HTML에 포함하기 전에 Babel을 사용하여 컴파일하는 것을 잊지 마세요.

**힌트**

이전 장에서 클래스를 작성하는 예제를 찾을 수 있습니다.

**힌트 2**

자바스크립트 파일을 변경할 때마다 `npm run build`를 실행하는 것을 잊지 마세요.  
그렇지 않으면 업데이트 되지 않습니다.

**해결방법의 예**

이 문제의 잠재적인 해결 방법은 [예제<sup>84</sup>](#)를 참조하세요.

---

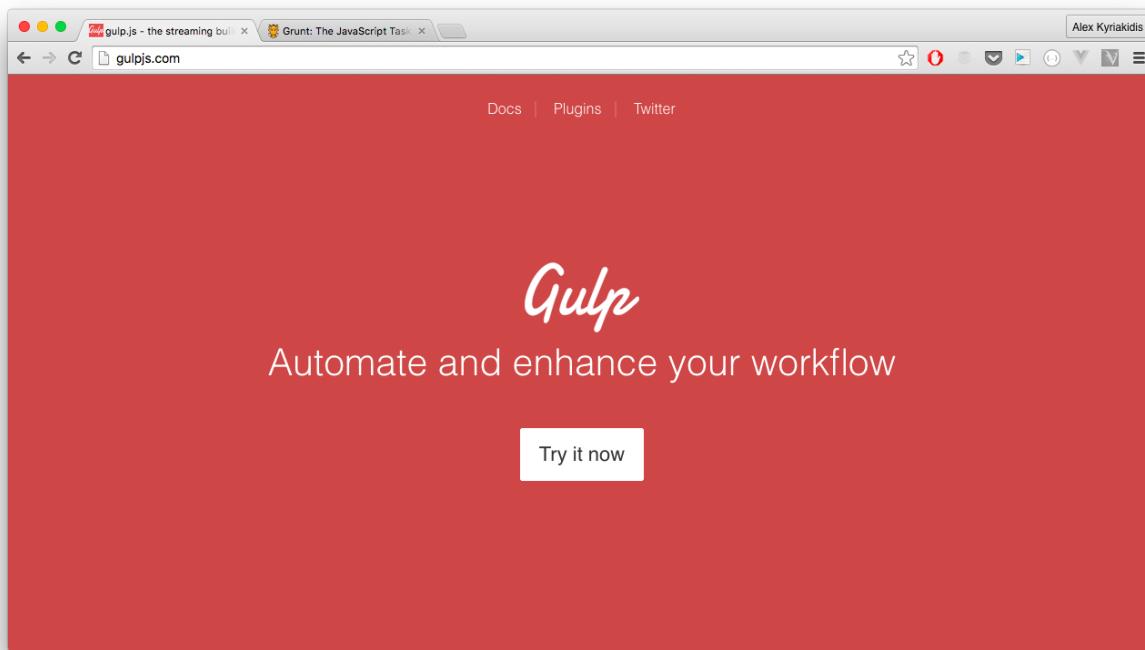
<sup>84</sup> <https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.1>

## Gulp를 이용한 워크플로우 자동화

### 태스크러너

이전 혼자해보기 섹션을 잘 수행하셨다면 코드를 변경할 때마다 `npm run build`를 실행 했을 것입니다.

[Gulp<sup>85</sup>](#) 또는 [Grunt<sup>86</sup>](#)와 같은 태스크러너를 이용하면 아주 편해집니다. 태스크러너는 워크플로우를 자동화 할 수 있습니다.



Gulp



### 왜 태스크러너를 사용해야 하나요?

자동화. 이 한 단어로 설명할 수 있습니다. 프로젝트 코드 최소화, 컴파일, 단위테스트, 린팅 등과 같은 반복되는 작업을 해야할 때 조금 더 적은 노력만 하면 됩니다.

<sup>85</sup><http://gulpjs.com/>

<sup>86</sup><http://gruntjs.com/>



## Gulp vs Grunt

Grun는 Gulp와 유사한 도구입니다. 태스크를 정의하고 실행할 수 있습니다.

가장 큰 차이는 Grunt는 설정 객체를 이용해서 태스크를 수행하고 Gulp는 자바스크립트 함수를 사용하는 점입니다. Gulp는 자바스크립트 기반으로 작동하기 때문에 태스크 작업시 매우 유연하게 사용할 수 있습니다.

Gulp와 Grunt 둘다 수많은 플러그인 라이브러리를 가지고 있습니다. 필요한 태스크가 있을 때 추가해서 사용할 수 있습니다.

### 설치

Gulp를 사용하여 js 파일의 변경을 감시하고 build 명령을 자동으로 실행하는 예제입니다. 먼저 Gulp를 전역으로 설치해야합니다.

>\_ npm install gulp-cli - global

그 다음 프로젝트의 devDependencies에 Gulp를 설치합니다.

>\_ npm install gulp - save-dev

이제 Gulp를 설치했으므로 태스크를 관리하는 **gulpfile.js**를 프로젝트의 루트에 만듭니다.

gulpfile.js

---

```
const gulp = require( 'gulp' );

gulp.task( 'default' , function() {
    // 기본 태스크를 여기에 작성합니다
});
```

---

### 사용법

콘솔에서 **gulp**를 실행하면 무언가 시작하지만 아무일도 일어나지 않습니다. default 태스크가 비어있기 때문입니다.

**babel**을 직접 실행하지 않고 **gulp-babel<sup>87</sup>**를 이용해서 하는 과정을 살펴보겠습니다.

<sup>87</sup> <https://www.npmjs.com/package/gulp-babel>

```
>_ npm install gulp-babel - save-dev
```

babel이라는 이름을 가지는 gulp task를 추가하고 기본 작업으로 설정합니다. gulpfile은 아래와 같습니다.

gulpfile.js

---

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', [ 'babel' ]);

// 기본 babel 태스크
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: [ 'es2015' ]
    }))
    .pipe(gulp.dest('assets/js/'))
})
```

---

이 태스크는 기본적으로 es2015 프리셋을 사용하여 src 디렉터리 아래의 모든 js 파일을 변환하여 assets/js 디렉터리에 넣도록 Babel에 요청합니다.

## 감시

현재 콘솔에서 실행하는 gulp는 npm run build와 같은 효과가 있습니다. 이번 섹션의 목표는 js 파일이 변경될 때마다 이 작업을 실행하는 것입니다. 이를 위해 gulpfile안에 감시자를 설정합니다.

gulpfile.js

---

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', [ 'watch' ]);

// 기본 babel 태스크
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: [ 'es2015' ]
    }))
})
```

---

```
.pipe(gulp.dest('assets/js/'))  
})  
  
// 기본 watch 태스크  
gulp.task('watch', function() {  
  gulp.watch('src/*.js', ['babel']);  
})
```

---

assets/js안의 파일에 변화가 생길때마다 gulp는 babel 태스크를 수행합니다. 정말 편해지겠죠?

## 혼자 해보기

이번 과제는 이전 내용과 이어집니다. 이전 장의 과제를 하지 않았으면 다시 돌아가셔야 합니다! 이 장은 태스크러너에 대한 내용을 다룹습니다. Gulp를 사용하여 감시자를 설정하고 변경이 감지되면 Babel을 사용하여 코드를 컴파일 해야합니다.



### 노트

이미 Gulp를 실행할 때 터미널에 메시지("Starting" - "Finished")를 출력합니다. 변경이 끝날때까지 약간의 시간이 걸릴 수 있습니다. 너무 서두르지 마세요



### 해결방법의 예

이 문제의 잠재적인 해결 방법은 예제<sup>88</sup>를 참조하세요.

<sup>88</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.2>

```
/babel-example » gulp watch
[19:59:55] Using gulpfile /babel-example/gulpfile.js
[19:59:55] Starting 'watch'...
[19:59:55] Finished 'watch' after 12 ms
[19:59:58] Starting 'babel'...
[20:00:04] Finished 'babel' after 5.35 s
```

Gulp가 지켜보고있어요!

## Webpack을 이용한 모듈 번들링

### 모듈 번들러

이제까지 작성한 워크플로우 `sum.js`는 잘 작동하고 있습니다. 이번에는 피자와 맥주 값을 계산하고 고객에게 알려주는 기능을 추가합니다.

src/sum.js

---

```
const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`Alex, you have to pay ${sum(pizza, beer)}`)
```

---

이 코드는 그럴듯 해 보이지만 모든 사람이 Alex는 아니므로 고객의 이름을 전달할 수 있는 `client.js`라는 새 파일을 생성합니다.

src/client.js

```
export const name = 'Alex'
```

이 파일에서 이름을 가져올 것입니다.

src/sum.js

```
import { name } from './client'

const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(` ${name} you have to pay ${sum(pizza, beer)} `)
```

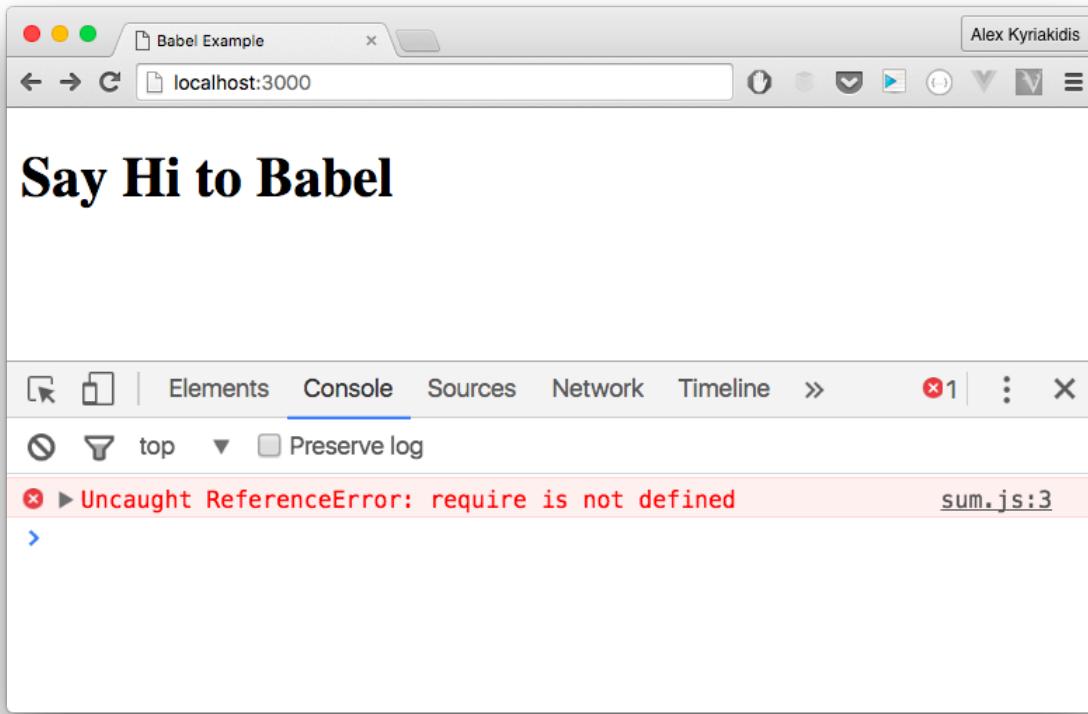
훌륭합니다! node assets/js/sum.js를 실행하면 이미 예상하고 있는 결과가 표시됩니다.



A screenshot of a terminal window titled "alex@192: /babel-example". The command "/babel-example » node assets/js/sum.js" is entered, followed by the output "Alex you have to pay 15\$". The terminal has a dark background with light-colored text and a light gray border.

sum.js 출력

브라우저에서 html 파일을 열 때 동일한 동작을 할 것으로 예상하겠지만 그렇지 않습니다! 여러분을 보셨을겁니다.



Require 가 정의되지 않았습니다

node assets/js/sum.js를 확인하고 맨 위에 있는 var \_client = require( './client' );를 살펴보세요. 브라우저에서 오류가 나는 이유는 require()가 브라우저/클라이언트측 자바스크립트에 없기 때문입니다. 이를 수정하려면 모듈을 한 파일에 묶어서 <script> 태그 안에 포함해야 합니다.

```

sum.js — assets/js — /babel-example
JS sum.js

'use strict';

var _client = require('./client');

var pizza = 10;
var beer = 5;

var sum = function sum(a, b) {
  return a + b + '$';
};

console.log(_client.name, ' have to pay', sum(pizza, beer));

```

The screenshot shows a file tree on the left with a file 'sum.js' selected. The file content is displayed on the right, showing a simple script that requires 'client' and defines a 'sum' function that adds two numbers and appends a dollar sign. It then logs the client's name and the result to the console.

assets/js/sum.js

이를 위해 Webpack<sup>89</sup> 또는 Browserify<sup>90</sup>와 같은 모듈 번들러가 필요합니다.

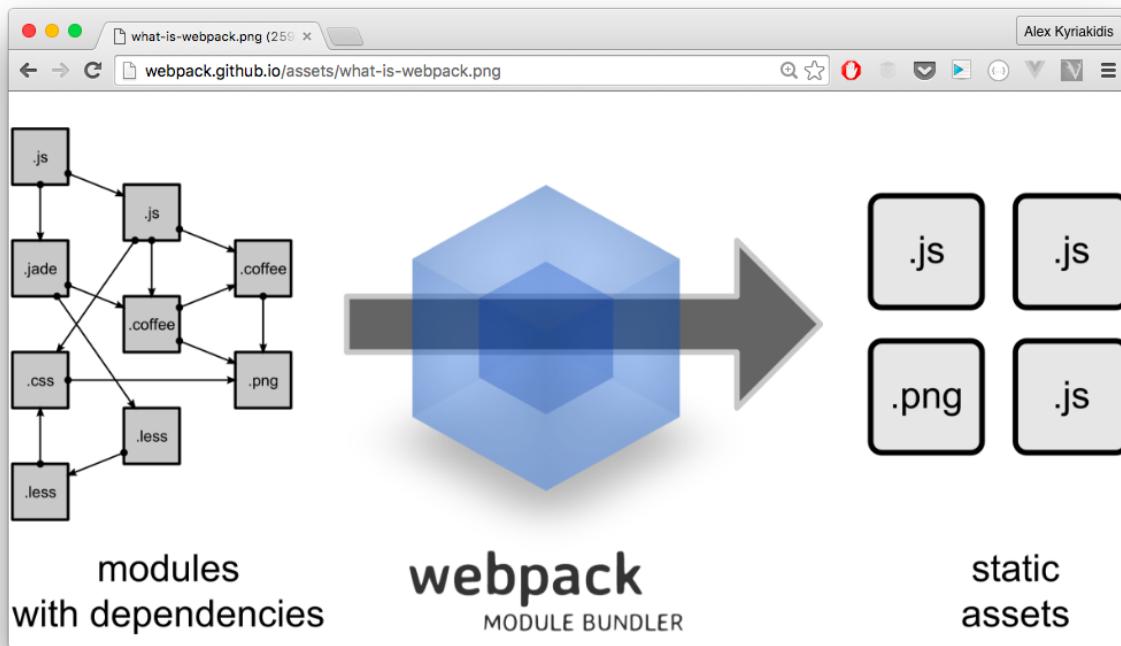
## Webpack

Webpack은 모듈 번들러입니다. 자바스크립트 모듈을 이용해 종속성을 판단한 후 연결해주고 필요한 모듈로 정적인 에셋을 만듭니다.

다음 예제에 Webpack을 사용할 것입니다. 왜냐하면 모듈을 번들하는 것 이외에도 쓸모가 있습니다. 로더를 사용하여 원하는 방법으로 모든 유형의 파일을 변환하도록 Webpack에 지시하여 최종 결과물을 얻을 수 있습니다.

<sup>89</sup> <https://webpack.github.io/>

<sup>90</sup> <http://browserify.org/>



Webpack의 동작

## 설치

먼저 Webpack을 전역으로 설치한 후 프로젝트에 종속성으로 추가해야합니다.

```
>_ npm install webpack -g  
      npm install webpack - save-dev
```



### 팁

이 글을 쓰는 시점에 Windows의 Vagrant<sup>91</sup>를 사용하는 환경에서 `npm install`을 사용하면 버그가 발생할 수 있습니다. 이 문제를 해결하려면 Vagrant를 종료하고 Windows의 터미널에서 프로젝트로 이동한 후 `npm install`을 실행하세요.

<sup>91</sup><https://www.vagrantup.com/>

## 사용법

자바스크립트를 컴파일 하기 위해 Webpack에 소스 위치와 출력 위치를 알려주어야 합니다. 이번 장에서는 원본 소스는 assets/js/sum.js이고, Babel을 통해 컴파일 된 코드의 위치는 assets/webpacked/app.js로 지정합니다.

```
>_ webpack assets/js/sum.js assets/webpacked/app.js
```

### Webpack 출력

빌드가 완료된 이후에 출력된 js인 assets/webpacked/app.js를 sum.html에서 사용할 수 있습니다. 물론 node assets/webpacked/app.js 명령어를 이용할 수 있습니다.

## 자동화

저처럼 게으른 사람이라면 변경할 때마다 webpack을 실행하지 않도록 자동화할 방법을 찾을 것 입니다. 원본 파일을 변경하면 다시 빌드하도록 Webpack을 구성할 수 있습니다. 여기서 이것을 다루지 않습니다. 대신에 여러 도구를 사용하는 방법을 보여주기 위해 Gulp를 사용합니다.

## 깊이 읽기

Webpack에 대해 더 깊게 공부하고 싶으면 Nader Dabit<sup>92</sup>의 “Beginner’s guide to Webpack”<sup>93</sup>를 읽어보세요.

Webpack을 Gulp을 통합하기 위해 [webpack-stream](#)<sup>94</sup> 플러그인을 사용합니다.

```
>_ npm install webpack-stream - save-dev
```

설치가 끝나면 `webpack`이라는 이름을 가지는 새 태스크를 만들고 `babel` 태스크를 실행한 후 변경 사항을 감지할 때마다 Gulp가 이를 실행하도록 합니다.

gulpfile.js

---

```
const gulp = require( 'gulp' );
const babel = require( 'gulp-babel' );
const webpack = require( 'webpack-stream' );

gulp.task( 'default' , [ 'watch' ]);

// 기본 babel 태스크
gulp.task( 'babel' , function() {
    return gulp.src( 'src/*.js' )
        .pipe(babel({
            presets: [ 'es2015' ]
        }))
        .pipe(gulp.dest( 'assets/js/'))
})

// 기본 webpack 태스크
gulp.task( 'webpack' , [ 'babel' ], function() {
    return gulp.src( 'assets/js/sum.js' )
        .pipe(webpack({
            output: {
                path: "/assets/webpacked",
                filename: "app.js"
            }
        }))
})
```

---

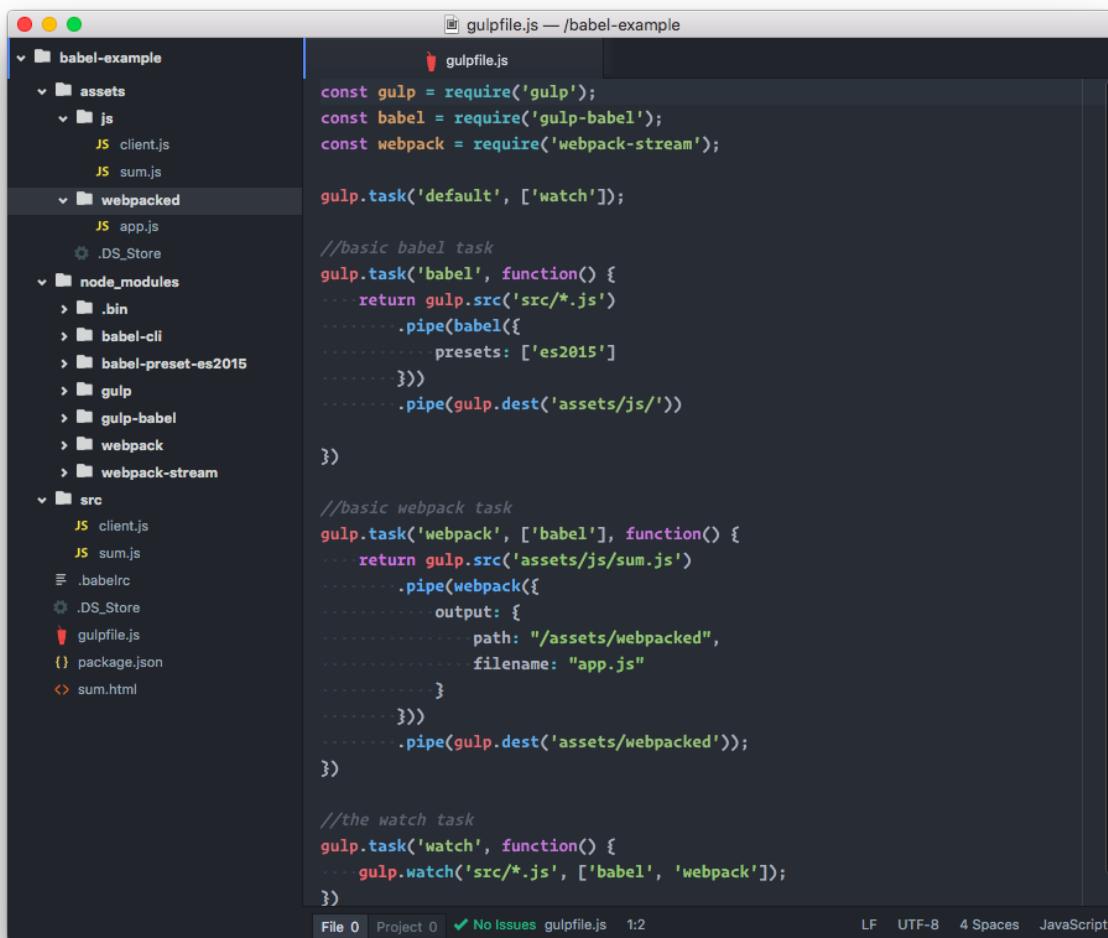
<sup>92</sup><https://twitter.com/dabit3>

<sup>93</sup><https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460>

<sup>94</sup><https://www.npmjs.com/package/webpack-stream>

```
.pipe(gulp.dest('assets/webpacked'));  
})  
  
// 기본 watch 태스크  
gulp.task('watch', function() {  
  gulp.watch('src/*.js', ['babel', 'webpack']);  
})
```

이 방법은 좋아보이지 않습니다. 단지 익힌 것들을 함께 사용하는 것을 보여주는 데 모일 뿐입니다. 프로덕션 환경에서는 Webpack 작업을 자동화하는 훨씬 좋은 방법이 있습니다.



The screenshot shows a code editor with a file tree on the left and a code editor window on the right. The file tree shows a project structure with folders like 'babel-example', 'assets' (containing 'js' with files 'client.js' and 'sum.js'), 'webpacked' (containing 'app.js' and '.DS\_Store'), 'node\_modules' (with various packages), and 'src' (with 'client.js', 'sum.js', '.babelrc', '.DS\_Store', 'gulpfile.js', 'package.json', and 'sum.html'). The code editor window displays 'gulpfile.js — /babel-example'. The code in the editor is:

```
const gulp = require('gulp');  
const babel = require('gulp-babel');  
const webpack = require('webpack-stream');  
  
gulp.task('default', ['watch']);  
  
//basic babel task  
gulp.task('babel', function() {  
  return gulp.src('src/*.js')  
    .pipe(babel({  
      presets: ['es2015']  
    }))  
    .pipe(gulp.dest('assets/js/'))  
});  
  
//basic webpack task  
gulp.task('webpack', ['babel'], function() {  
  return gulp.src('assets/js/sum.js')  
    .pipe(webpack({  
      output: {  
        path: "/assets/webpacked",  
        filename: "app.js"  
      }  
    }))  
    .pipe(gulp.dest('assets/webpacked'))  
});  
  
//the watch task  
gulp.task('watch', function() {  
  gulp.watch('src/*.js', ['babel', 'webpack']);  
})
```

At the bottom of the code editor, there are tabs for 'File 0', 'Project 0', 'No Issues', 'gulpfile.js', '1:2', 'LF', 'UTF-8', '4 Spaces', and 'JavaScript'.

Gulp에서 Webpack 사용

## 요약

ES6를 컴파일하기 위해 [Babel<sup>95</sup>](#)을 사용했습니다.

이 같은 작업(최소화, SASS/LESS 컴파일 등)을 자동화하기 위해 태스크러너인 [Gulp<sup>96</sup>](#) 또는 [Grunt<sup>97</sup>](#)를 사용합니다.

모듈을 번들링하기 위해 [Webpack<sup>98</sup>](#) 또는 [Browserify<sup>99</sup>](#)을 사용합니다.

다음 장에서는 Vue의 단일 파일 컴포넌트에 대해 살펴보고 Vue가 제공하는 여러 도구와 함께 지금까지 배운 도구들을 사용해볼 것입니다.



### 노트

이 장이 이해하기 어려웠다면 걱정하지 마세요. 모든 내용을 기억할 필요는 없습니다.

이번 장에서 보여준 것들이 어떻게 작동하는지만 이해해도 괜찮습니다. 다음 장에서는 프로젝트 템플릿을 사용할 것입니다. 모듈 번들링, 자동화, 변경이 생기면 빌드등의 기능이 이미 구현되어 있으므로 이를 활용할 것입니다.

---

<sup>95</sup> <http://babeljs.io/>

<sup>96</sup> <http://gulpjs.com/>

<sup>97</sup> <http://gruntjs.com/>

<sup>98</sup> <https://webpack.github.io/>

<sup>99</sup> <http://browserify.org/>

# 단일 파일 컴포넌트

이전 장의 마지막에서 언급했던 것처럼 이번 장에서는 단일 파일 컴포넌트를 살펴봅니다. Vue의 단일 파일 컴포넌트를 사용하기 위해 Webpack과 vue-loader 또는 Browserify와 vueify가 필요합니다. 이번 예제에서는 Webpack을 사용합니다. 이전 장에서 Webpack을 어떻게 사용하는지 이미 살펴보았습니다. Browserify를 선호한다면 사용하더라도 문제될 것이 없습니다.

단일 파일 컴포넌트 혹은 뷰 컴포넌트는 템플릿과 자바스크립트, CSS 스타일을 가지고 있는 .vue로 구성됩니다. Webpack이 처리하는 단계에서 다른 파일과 함께 번들링 합니다.

Webpack은 vue-loader<sup>100</sup>를 사용하여 Vue 컴포넌트를 일반 자바스크립트 모듈로 변환합니다. vue-loader는 기본적으로 활성화된 ES2015와 범위를 가지는 CSS 등의 기능도 제공합니다.

## vue-cli

Webpack을 이용하는 새로운 워크플로우를 작성하지 않기 위해 vue-cli를 사용합니다.



### 정보

vue-cli<sup>101</sup>는 Vue.js 프로젝트를 구성하기 위한 간단한 커맨드라인 인터페이스입니다.

이 훌륭한 도구는 미리 구성된 빌드를 만들 수 있는 가장 빠른 방법입니다. 핫-리로드, 저장 시 린트, 유닛 테스팅 이외에도 많은 기능을 가지고 있습니다. 현재 Webpack 및 Browserify 용 템플릿을 제공하고 있지만, 필요한 경우 [스스로 만들어서 사용](#)<sup>102</sup> 할 수도 있습니다.

## vue-cli 템플릿

Vue.js 공식 템플릿 저장소<sup>103</sup>는 CLI에서 사용할 수 있는 다섯가지 종류의 템플릿을 제공합니다. 이 숫자는 더 늘어날 것입니다. Github에서 어떤 것들이 있는지 확인하세요.

모든 템플릿은 package.json 파일을 가지고 있습니다. 이 파일은 프로젝트의 의존성을 처리하고 미리 설정된 NPM 스크립트를 포함합니다.

<sup>100</sup><https://github.com/vuejs/vue-loader>

<sup>101</sup><https://github.com/vuejs/vue-cli>

<sup>102</sup><https://github.com/vuejs/vue-cli#custom-templates>

<sup>103</sup><https://github.com/vuejs-templates>

Vue의 프로젝트 템플릿을 사용하면 많은 기능을 함께 사용할 수 있습니다. 예를 들어, “Webpack” 템플릿의 설명에 따르면 “핫-리로드, 린팅, 테스트 및 CSS 추출 기능을 갖춘 완벽한 기능의 Webpack + vue-loader를 포함합니다.”

## 설치

예제에서는 Webpack 설치 방법을 따릅니다. 다음 명령을 사용하여 **vue-cli**를 전역으로 설치하세요.

```
>_ npm install vue-cli -g
```

## 사용법

CLI를 사용하면 터미널에서 **vue init <template-name> <project-name>** 명령어를 사용할 수 있습니다. <**template-name**> 이름은 Vue 템플릿(공식 또는 사용자 정의)입니다. 그리고 <**project-name**>은 프로젝트가 저장될 디렉터리/프로젝트의 이름입니다.

이제 아래와 명령어를 보세요.

```
>_ vue init webpack-simple simple-project
```

`simple-project` 프로젝트의 구조를 살펴보겠습니다.

```

{
  "name": "simple-project",
  "description": "A Vue.js project",
  "author": "tmvuejs",
  "private": true,
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --inline --hot",
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules"
  },
  "dependencies": {
    "vue": "^2.0.1"
  },
  "devDependencies": {
    "babel-core": "^6.0.0",
    "babel-loader": "^6.0.0",
    "babel-preset-es2015": "^6.0.0",
    "cross-env": "^3.0.0",
    "css-loader": "^0.25.0",
    "file-loader": "^0.9.0",
    "vue-loader": "^9.7.0",
    "webpack": "^2.1.0-beta.25",
    "webpack-dev-server": "^2.1.0-beta.0"
  }
}

```

File 0 Project 0 No Issues package.json 1:1 LF UTF-8 2 Spaces JSON 7 updates

### webpack-simple 구조

예를 들어 완전한 기능을 가지는 Webpack 템플릿을 사용하려면 아래 명령어를 사용하세요.

```
>_ vue init webpack stories-classic-project
```



### 정보

`vue list`를 입력하면 공식 템플릿 목록을 볼 수 있습니다.

새 프로젝트를 초기화할 때 이름, 버전, 작성자 등 세부 정보를 입력하라는 메시지가 표시됩니다.

ESlint 프리셋을 선택하는 질문이 나올 것입니다. [feross/standard](https://github.com/feross/standard)<sup>104</sup>와 [airbnb/javascript](https://github.com/airbnb/javascript)<sup>105</sup> 중 하나를 선택하세요.

<sup>104</sup> <https://github.com/feross/standard>

<sup>105</sup> <https://github.com/airbnb/javascript>

스타일이 적용되는 규칙을 잘 이해할 수 있는 비교표를 작성하였습니다.

규칙	feross/standard	airbnb/javascript
들여쓰기	공백 2개	공백 2개
세미콜론	아니오!	예
사용하지 않은 변수	허용 안함	허용 안함
문자열 따옴표	작은 따옴표	작은 따옴표
==== 강제 여부	예	예
빈 줄 허용 갯수	1	2
함수 이름 뒤 공백	예	아니오
[ 로 시작 가능 여부	아니오	예
파일 마지막 줄의 글자 w/a	예	예
뒤따르는 콤마 사용 여부	아니오	아니오



## Standard와 Airbnb

표의 내용은 스타일에 적용되는 것을 중심으로 작성했습니다. 가장 적합한 것을 살펴보고 결정하려면 Github 저장소를 확인하세요.

스타일을 선택한 후 Karma-Mocha<sup>106</sup> 및 Nightwatch<sup>107</sup>와 같은 여러 도구를 설치하라는 메시지를 표시합니다. 지금 당장은 이러한 도구들이 필요하지 않으므로 아니오를 선택하세요.

<sup>106</sup><https://github.com/karma-runner/karma-mocha>

<sup>107</sup><http://nightwatchjs.org/>

```
1. blackpr@blackbs-mbp: ~/vuejs (zsh)
→ vuejs vue init stories-project
? Generate project in current directory? (Y/n)

→ vuejs vue init webpack stories-project
? Project name stories-project
? Project description A Vue.js project
? Author Alex
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

vue-cli · Generated "stories-project".

To get started:

cd stories-project
npm install
npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack
→ vuejs █
```

## Vue 템플릿 설치



### 정보

Karma는 테스트 프레임워크 Mocha<sup>108</sup>의 플러그인 어댑터입니다.

Nightwatch는 Selenium<sup>109</sup> 서버를 실행하는 브라우저 자동 테스트를 작성하는데 사용합니다.

## Webpack 템플릿

프로젝트 설정을 마무리하려면 의존성을 설치해야합니다. 계속 따라가 보겠습니다.

<sup>108</sup><https://mochajs.org/>

<sup>109</sup><http://www.seleniumhq.org/>

```
>_ cd stories-classic-project  
    npm install  
    npm run dev
```

터미널에 Listening at http://localhost:8080 이 표시됩니다.

**webpack: bundle is now VALID** 메시지가 출력될 때까지 기다려야 합니다.

The screenshot shows a terminal window titled "1. npm run dev (node)". It displays the following output:

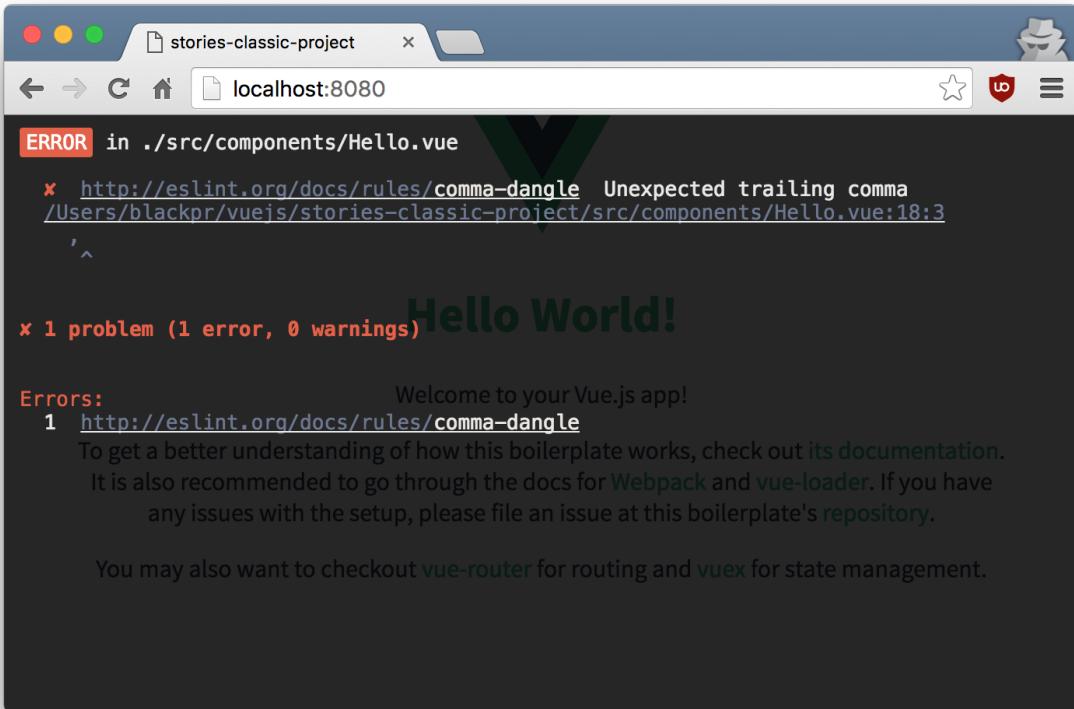
```
1. npm run dev (node)  
  └─ lodash._arraycopy@3.0.0  
    └─ lodash._arrayeach@3.0.0  
      └─ lodash._createassigner@3.1.1  
        └─ lodash._isiteratee@3.0.9  
          └─ lodash._restparam@3.6.1  
            └─ lodash._getnative@3.9.1  
              └─ lodash._istypedarray@3.0.6  
                └─ lodash._toplainobject@3.0.0  
                  └─ lodash._basecopy@3.0.1  
  
  ➔ stories-classic-project npm install  
  ➔ stories-classic-project npm run dev  
  
> y@1.0.0 dev /Users/blackpr/vuejs/stories-classic-project  
> node build/dev-server.js  
  
Listening at http://localhost:8080  
  
webpack built 459f183651c77b8f6e8d in 1815ms  
Hash: 459f183651c77b8f6e8d  
Version: webpack 1.13.1  
Time: 1815ms  
  Asset      Size  Chunks      Chunk Names  
  app.js    1.06 MB    0  [emitted]  app  
index.html 234 bytes    [emitted]  
Child html-webpack-plugin for "index.html":  
  Asset      Size  Chunks      Chunk Names  
  index.html 21.4 kB    0  
webpack: bundle is now VALID.
```

서버 실행중...



## 주의사항

주의하세요. 코드에 명시적으로 처리할 부분들이 많습니다. 블럭 사이의 빈 줄, 라인 끝의 공백, 2개의 공백을 사용하는 등의 [선택한 스타일 규칙](#)을 따르지 않으면 오류가 발생합니다.



에러 출력



## 노트

`webpack-simple`을 사용하는 경우 브라우저 화면을 가려 출력되는 오류 대신 터미널에서 출력됩니다.

## 프로젝트 구조

위 단계를 마친 후 프로젝트 디렉터리에 필요한 모든 파일들이 있어야 합니다.

The screenshot shows a code editor interface with a sidebar containing a project tree and three tabs at the top: index.html, main.js, and Hello.vue. The main.js tab is active, displaying the following code:

```
import Vue from 'vue';
import App from './App';

/* eslint-disable no-new */
new Vue({
  el: '#app',
  template: '<App/>',
  components: { App },
});
```

The project tree on the left includes build, config, node\_modules, src (containing assets, components, Hello.vue, App.vue), static, .babelrc, .editorconfig, .eslintrc.js, .gitignore, index.html, package.json, and README.md.

## Webpack 구조

앞으로 다룰 파일의 목록입니다.

1. **index.html**
2. **main.js**
3. **src**와 **src/components** 디렉터리 아래의 파일들

## index.html

**index.html**부터 살펴보겠습니다. 아래처럼 보일 것 입니다.

index.html

---

```
<html>
  <head>
    <meta charset=" utf-8 " >
    <title>stories-classic-project</title>
  </head>
  <body>
    <app></app>
    <!-- built files will be auto injected -->
  </body>
</html>
```

---

보시다시피 이미 컴포넌트가 포함된 기본적인 설정입니다. 주석은 Webpack의 결과물인 **app.js** 스크립트를 참조합니다. 기본적으로 Webpack이 스크립트를 번들한 다음 자동으로 만들어진 스크립트를 추가합니다. 수동으로 추가할 필요가 없습니다.

## Hello.vue

src/components 디렉터리를 찾아 Hello.vue 파일을 열어보세요. .vue의 구조를 볼 수 있습니다.

src/components/Hello.vue

---

```
<template>
  <div class=" hello " >
    <h1>{{ msg }}</h1>
    <h2>Essential Links</h2>
    ...
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>
```

<!-- "scoped" 속성은 CSS를 이 컴포넌트에서만 사용할 수 있도록 합니다 -->

```
<style scoped>
h1, h2 {
  font-weight: normal;
}
...
</style>
```

**<script>** 태그 안에는 컴포넌트의 data만 들어있습니다. 템플릿을 정의할 필요는 없습니다. **<template>**이 존재하면 자동으로 바인딩 됩니다. **<template>** 블럭은 당연히 컴포넌트의 템플릿을 정의합니다. HTML의 **<template-hello>** 태그처럼 **<template>**를 생각해보세요

**<script>** 블럭만 사용할 수 있지만 이 방법은 단일 파일 컴포넌트의 장점을 잊게 됩니다.

ES6의 기능인 **export**는 이미 설치한 멋진 도구들(트랜스파일러 + 모듈 번들러)에 의해 처리되는 것을 기억하세요.



## 주의사항

각 .vue 파일은 하나의 **<script>** 블럭만 가집니다. 모든 템플릿은 **<div id=hello>...</div>**와 같은 다른 엘리먼트를 캡슐화하는 단 하나의 루트 엘리먼트만 가집니다.

스크립트는 Vue.js 컴포넌트 옵션 객체를 내보내야 합니다. **Vue.extend()**에 의해 생성된 확장 생성자를 내보내는 것도 지원되나, 평범한 객체 사용을 권장합니다.

ES6는 여러개의 내보내기를 허용하지만 단일 파일 컴포넌트는 그렇지 않습니다.

추가로 내보내기를 하려면 아래와 같은 경고를 표시합니다.

**[vue-loader] src/components/Hello.vue: named exports in /\*.vue files are ignored.**  
예상대로 **<style>** 블럭은 CSS 스타일을 정의합니다.

## App.vue

애플리케이션의 메인 템플릿을 가지는 App.vue 파일은 src에 있습니다. 이는 컴포넌트이며 일반적으로 다른 컴포넌트를 포함합니다.

App.vue에는 텍스트와 스타일을 포함합니다. 아래 파일 예제는 구조를 잡는데 중점을 두고 있어 간단하게 구성하였습니다.

src/App.vue

---

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

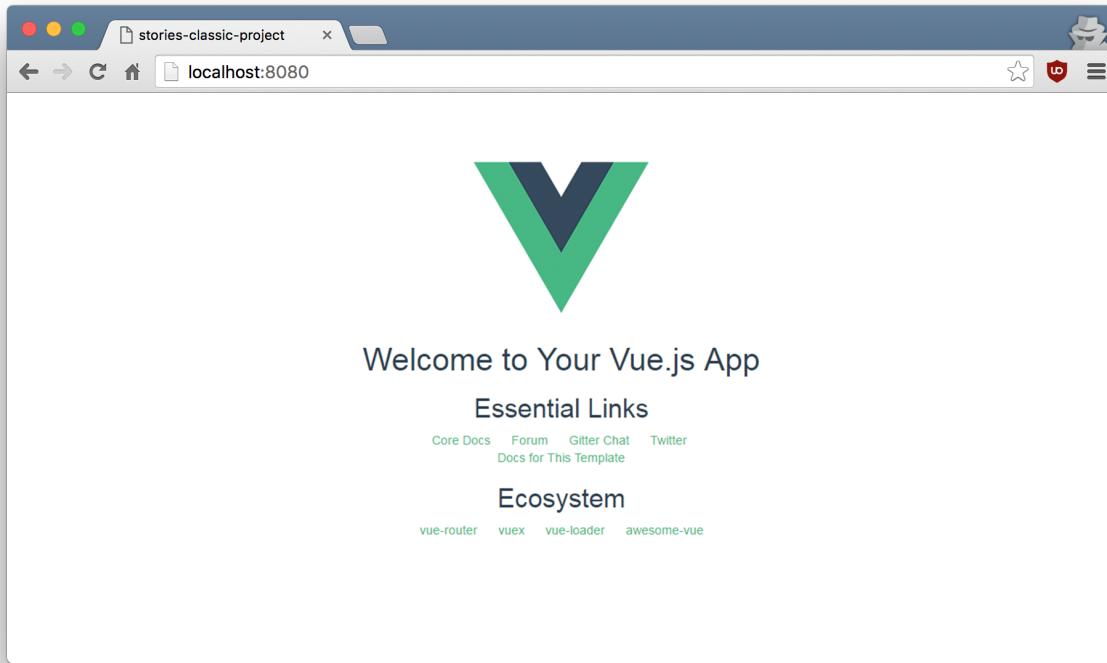
<script>
import Hello from './components/Hello'

export default {
  name: 'app',
  components: {
    Hello
  }
}
</script>

<style>
...
</style>
```

---

위에서 본 Hello.vue 파일과 유사한 구조를 가지고 있습니다. 기본적으로, Hello 컴포넌트를 포함하는 **components** 객체를 가집니다. 이 객체 안에서 새 컴포넌트들을 가져올 수 있습니다. 템플릿을 보면, **<hello></hello>**를 볼 수 있습니다. 당연히 Hello 컴포넌트의 템플릿 내용이 화면에 표시 됩니다.



프로젝트 홈페이지

## main.js

main.js 파일은 src 디렉토리에 있습니다. 당연히 이 파일이 메인 스크립트입니다.

src/main.js

```
import Vue from 'vue'
import App from './App'

/* eslint-disable no-new */
new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

Vue를 node\_modules 모듈과 src 디렉토리의 App 컴포넌트에서 모듈로 가져옵니다. 아래는 Vue 인스턴스가 있고, 컴포넌트 객체에는 App 인스턴스가 있습니다.

스크립트나 컴포넌트를 전역으로 가져올 필요가 생기면 main.js 안에 넣으면 됩니다.



## 노트

`template: <App />` 옵션은 앞에서 언급한 `index.html`에 넣을 컴포넌트의 템플릿 출력입니다.

`<App/>`는 `<App></App>` 또는 `<app></app>`와 동일합니다.



## 정보

Webpack 템플릿 프로젝트 구조에 대한 자세한 내용은 해당 [설명서<sup>110</sup>](#)를 참조하십시오.

## .vue 파일 만들기

단일 파일 컴포넌트의 내용과 프로젝트에서 사용하는 방법을 살펴 보았습니다. 실제 시나리오에 적용해볼 차례입니다. 사용자가 자신의 이야기와 경험을 올리는 일종의 소셜 네트워크 혹은 포럼을 만든다고 가정합니다. 시작하려면 가입과 로그인을 위한 두가지 폼과 사용자의 이야기를 표시하는 한개의 페이지가 필요합니다.

시작하기 전에 모든 컴포넌트에서 스타일을 사용할 수 있도록 부트스트랩을 추가합니다. 이를 위해 `index.html` 파일을 수정합니다.

`index.html`

---

```
<html>
  <head>
    <meta charset=" utf-8 " >
    <title>stories-classic-project</title>
    <link rel=" stylesheet " href=" https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css " >
  </head>
  <body>
    <div id=" app " ></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

---

로그인 폼을 만듭니다. 파일 이름은 `Login.vue` 입니다.

---

<sup>110</sup> <http://vuejs-templates.github.io/webpack/structure.html>

src/components/Login.vue

```
<template>
  <div id= " login " >
    <h2>Sign in</h2>
    <input type= " email " placeholder= " Email address " >
    <input type= " password " placeholder= " Password " >
    <button class= " btn " >Sign in</button>
  </div>
</template>

<script>
export default {
  created () {
    console.log( ' login ' )
  }
}
</script>
```

브라우저에서 위 파일을 보려면 Login 컴포넌트를 어딘가에 포함해야 합니다. 이를 위해 App 컴포넌트를 가져와 컴포넌트 객체에 추가합니다.

src/App.vue

```
<template>
  <div id= " app " >
    <img src= " ./assets/logo.png " >
    <hello></hello>
  </div>
</template>

<script>
import Login from './components/Login.vue'
import Hello from './components/Hello'

export default {
  name: ' app ',
  components: {
    Hello,
    Login
  }
}
</script>

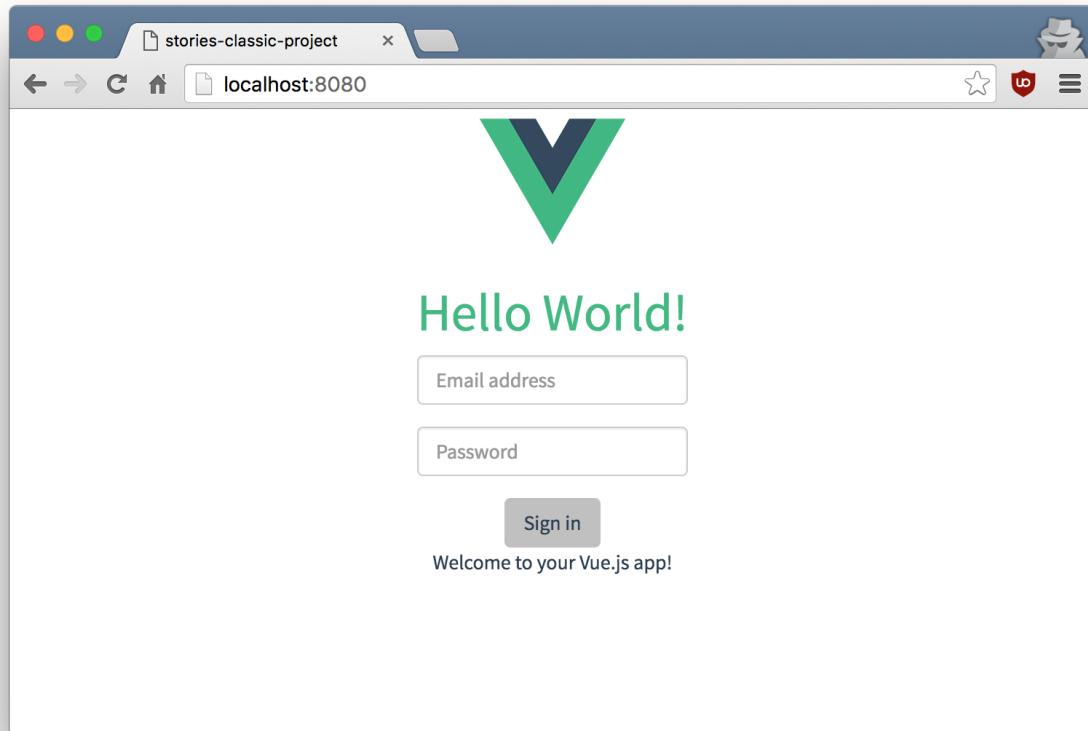
<style>
```

```
...  
  </style>
```

브라우저를 새로고침하여도 Login 컴포넌트는 표시되지 않습니다. 아직 참조하지 않았기 때문입니다. <hello></hello> 아래에 추가하면 로그인 폼을 볼 수 있습니다!

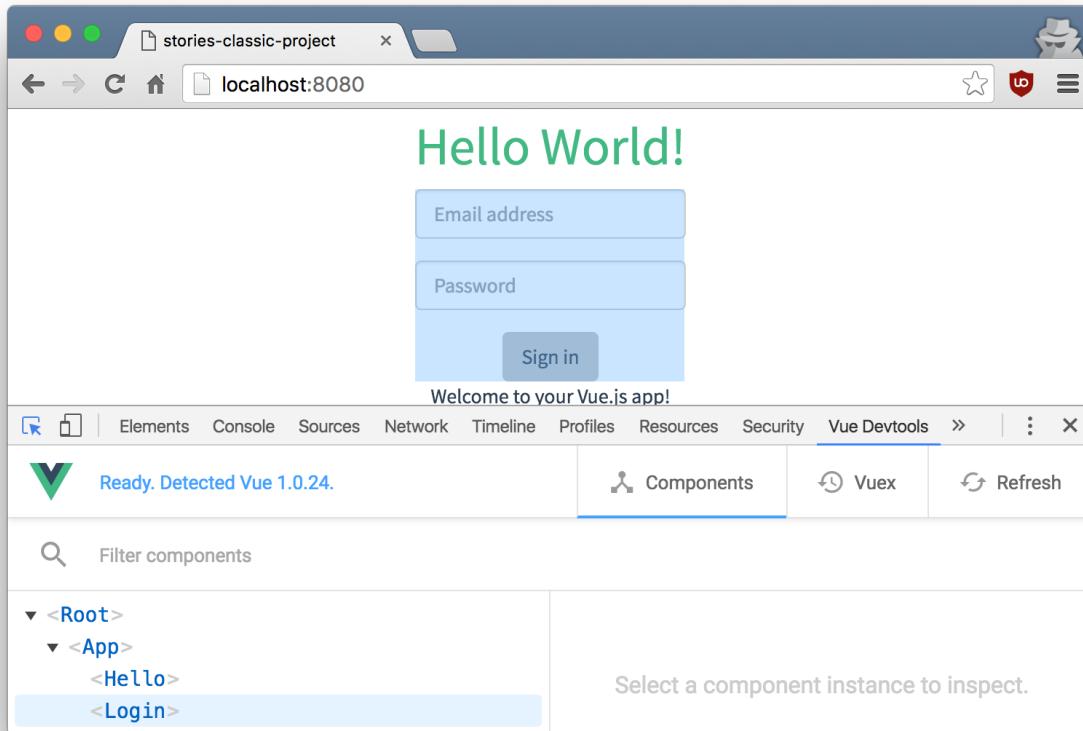
src/App.vue

```
<template>  
  <div id= " app " >  
    <img src= "./assets/logo.png " >  
    <hello></hello>  
    <login></login>  
  </div>  
</template>  
...
```



로그인 컴포넌트

브라우저 콘솔을 열면 컴포넌트가 생성될 때 **login** 메시지를 볼 수 있습니다. vue-devtools를 사용하는 경우 컴포넌트 트리에서도 표시 됩니다.



### 트리 뷰

이번에는 가입을 위한 컴포넌트를 만듭니다.

src/components/Register.vue

```
<template>
<div id="register">
  <h2>Register Form</h2>
  <input placeholder="First Name" class="form-control" />
  <input placeholder="Last Name" class="form-control" />
  <input placeholder="Email address" class="form-control" />
  <input placeholder="Pick a password" class="form-control" />
  <input placeholder="Confirm password" class="form-control" />
  <button class="btn">Sign up</button>
</div>
</template>
```

```
<script>
export default {
  created () {
    console.log( 'register' )
  }
}
</script>
```

---

이제 App.vue 파일에서 가져옵니다.

src/App.vue

```
<template>
<div id= " app " >
  ...
  <!-- <hello></hello> -->
  <!-- <login></login> -->
  <register></register>

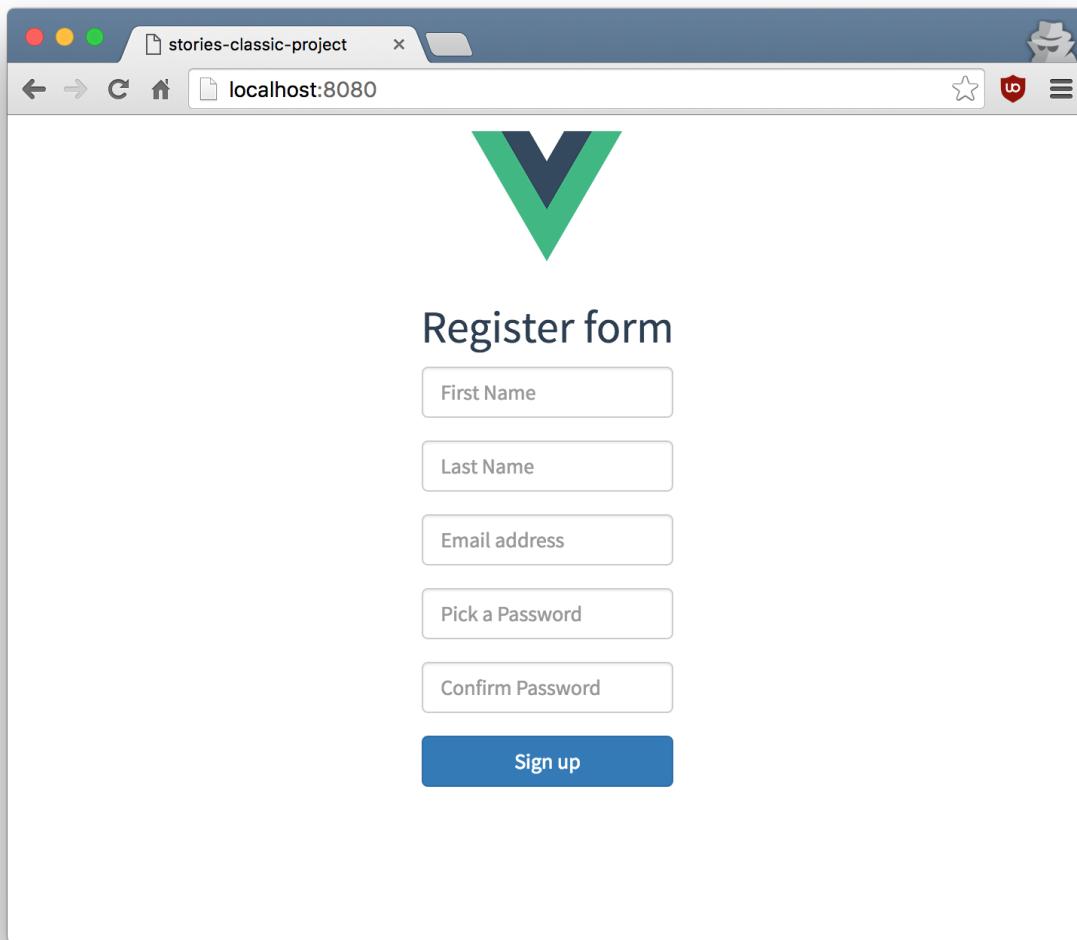
  ...
</div>
</template>

<script>
// import Hello from './components/Hello'
// import Login from './components/Login'
import Register from './components/Register'

export default {
  name: 'app',
  components: {
    // Hello,
    // Login,
    Register,
  }
}
</script>
...
```

---

브라우저를 확인하면 Register 컴포넌트의 템플릿을 보실 수 있습니다.



## 컴포넌트 등록



### 노트

다른 컴포넌트는 보여줄 필요가 없기 때문에 주석처리했습니다. 최초에 있었던 **Hello** 컴포넌트는 더이상 예제에서 필요하지 않으므로 제거합니다.

앞서 소셜 네트워크(또는 유사한 것)를 만들기로 했습니다. 이번에는 사용자의 이야기를 보여줄 차례입니다. 따라서 렌더링 할 때 모든 사용자의 이야기를 가져올 수 있는 **Stories** 컴포넌트를 만들어야 합니다.

src/components/Stories.vue

---

```
<template>
  <ul class= "list-group" >
    <li v-for= "story in stories" class= "list-group-item" >
      {{ story.writer }} said "{{ story.plot }}"
      Story upvotes {{ story.upvotes }}.
    </li>
  </ul>
</template>

<script>
export default {
  data () {
    return {
      stories: [
        {
          plot: 'My horse is amazing.',
          writer: 'Mr. Weebl',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Narwhals invented Shish Kebab.',
          writer: 'Mr. Weebl',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'The dark side of the Force is stronger.',
          writer: 'Darth Vader',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'One does not simply walk into Mordor',
          writer: 'Boromir',
          upvotes: 74,
          voted: false
        }
      ]
    }
  }
}
</script>
```

---

Stories.vue 파일의 내용입니다. 메인 App.vue 파일에서 사용할 수 있습니다. 이 시점에서 이야기는 단순하게 작업하기 위해 하드코딩 했습니다. 다른 컴포넌트와 마찬가지로 가져와서 사용해야 합니다.

src/App.vue

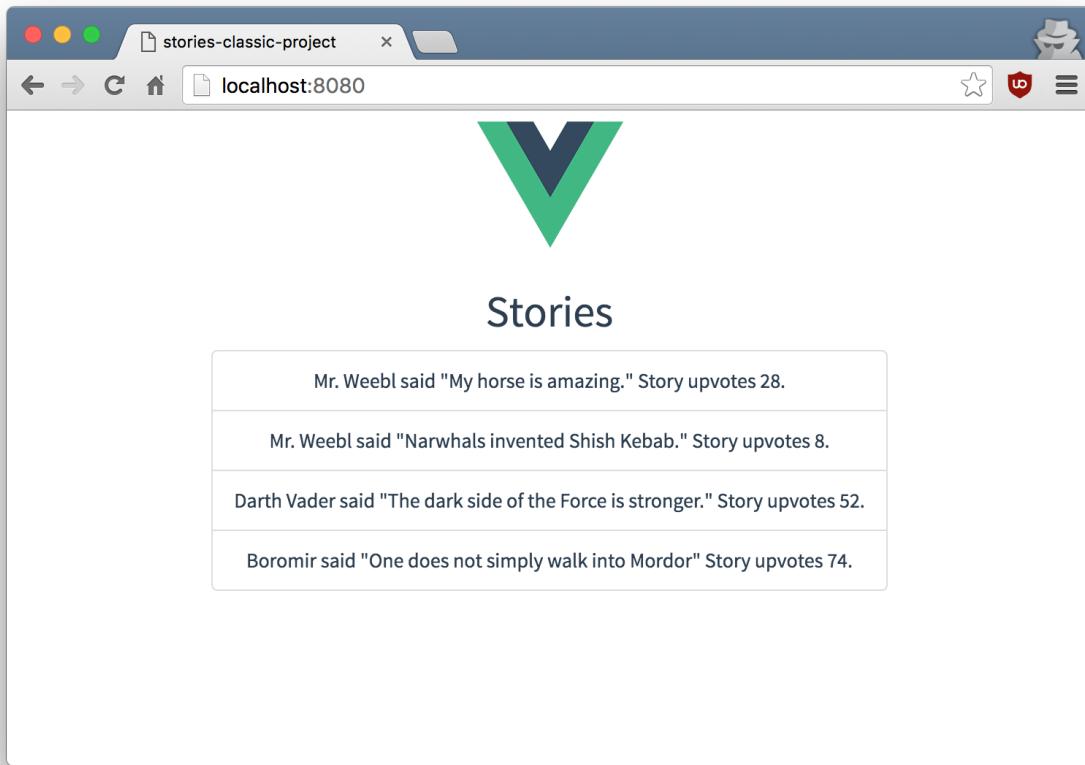
---

```
<template>
  <div id= "app ">
    <img class= "logo " src= "./assets/logo.png " >
    <!-- <login></login> -->
    <!-- <register></register> -->
    <stories></stories>
  </div>
</template>

<script>
// import Login from './components/Login.vue'
// import Register from './components/Register.vue'
import Stories from './components/Stories.vue'
export default {
  components: {
    Login,
    Register,
    Stories
  }
}
</script>

<style>
...
</style>
```

---



### 이야기 컴포넌트

좋습니다! 이제 모든 목록을 표시할 페이지가 생겼습니다.

### 컴포넌트 중첩

“인기 있는” 이야기를 원하는 장소 어디든 보여줄 필요가 있습니다. 따라서 앞으로 만들 Famous 컴포넌트는 필요한 곳 어디서나 사용할 수 있어야 합니다.

src/components/Famous.vue

```
<template>
  <div id= "famous " >
    <h2>Trending stories<strong>({{famous.length}})</strong></h2>
    <ul class= "list-group " >
      <li v-for= "story in famous " class= "list-group-item " >
        {{ story.writer }} said "{{ story.plot }}".
        Story upvotes {{ story.upvotes }}.
      </li>
    </ul>
  </div>
```

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

<script>
export default {
  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
      })
    }
  },
  data () {
    return {
      stories: [
        {
          plot: 'My horse is amazing.',
          writer: 'Mr. Weebl',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Narwhals invented Shish Kebab.',
          writer: 'Mr. Weebl',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'The dark side of the Force is stronger.',
          writer: 'Darth Vader',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'One does not simply walk into Mordor',
          writer: 'Boromir',
          upvotes: 74,
          voted: false
        }
      ]
    }
  }
}
</script>
```

---

위 내용은 Famous.vue 파일의 전체입니다. 이전 장에서 했던 것 처럼 계산된 속성을 사용해 stories 배열을 필터링하고 그 결과를 표시하는 template을 만들었습니다.

## 노트

stories 배열은 이번에도 하드코딩 합니다. 데이터는 이전과 같습니다. 좋은 방법이 아니지만 나중에 stories 배열을 정의하고 모든 컴포넌트에서 공유할 수 있는 방법을 알아보겠습니다.

그렇다면 어디에 이 컴포넌트를 사용할 수 있을까요? 가입 페이지에서 보여주고 사용자에게 인기있는 이야기에 대한 관심을 끌 수 있도록 하겠습니다. 이 말은 – 현재 프로젝트에서 – Register 컴포넌트 안에 Famous 컴포넌트를 넣는다는 것을 의미합니다. 이것은 App.vue 내부에서 했던 방식과 동일하게 할 수 있습니다.

그래서 Register.vue를 열고 가져온 다음 템플릿안에 추가하면 됩니다.

src/components/Register.vue

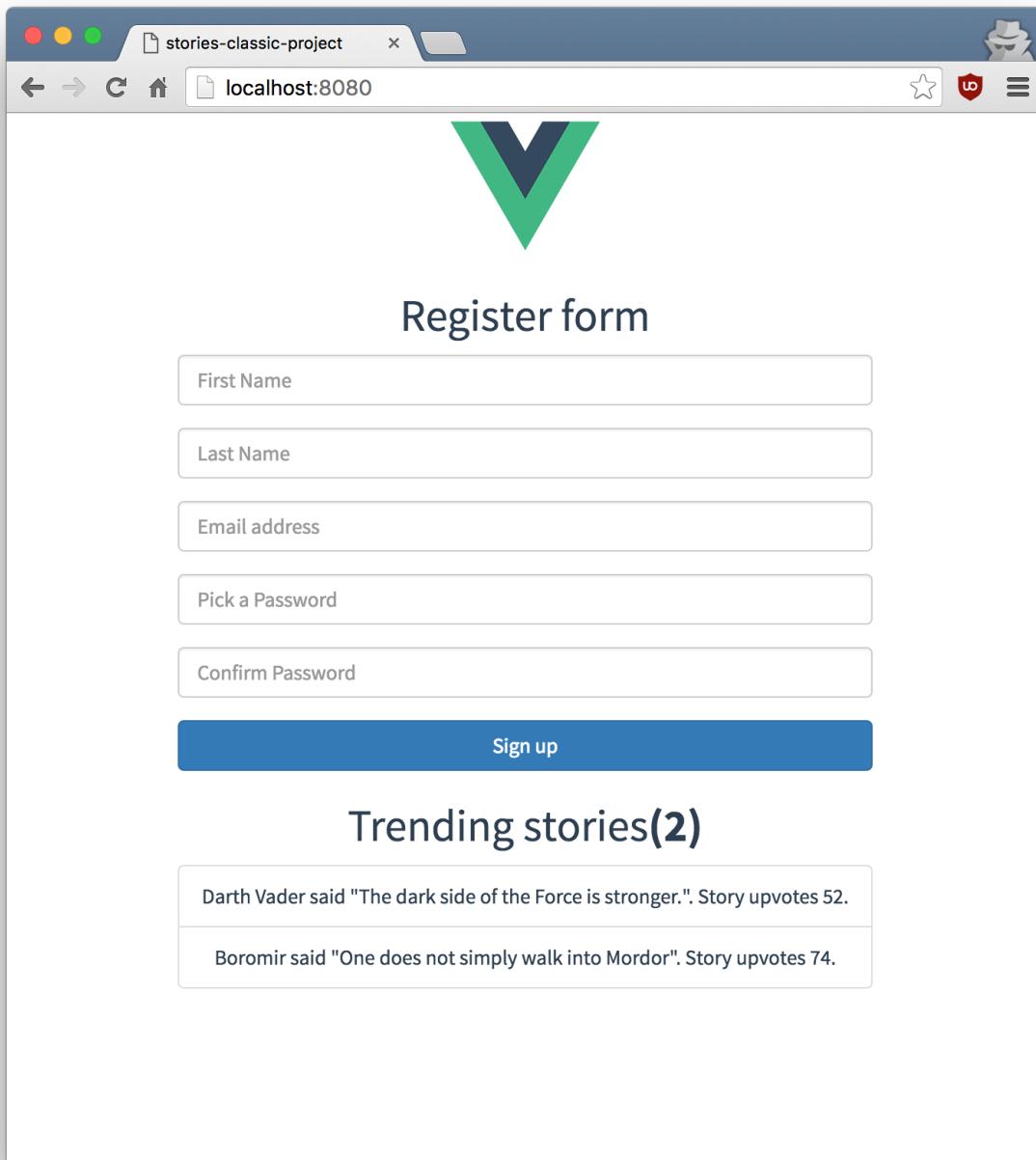
---

```
<template>
  <div id= "register" >
    <h2>Register Form</h2>
    ...
    <famous></famous>
  </div>
</template>

<script>
import Famous from './Famous.vue'

export default {
  components: {
    Famous
  },
  created () {
    console.log( 'register' )
  }
}
</script>
```

---



인기 있는 이야기를 보여주는 가입 페이지

가져올 때 파일 경로를 주의해야 합니다. 이제 두 파일이 같은 경로에 있으므로 전체 경로 대신 상대 경로 **./Famous**를 사용합니다. 익숙하지 않은 경우 실수하기 쉬우므로 주의해야 합니다!



## 예제 코드

이 장의 예제 코드는 [GitHub<sup>111</sup>](#)에 있습니다.

---

<sup>111</sup><https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter16/16.3>

# 중복 상태 제거

앞의 예제에서는 컴포넌트 내부의 데이터의 배열 –이야기 배열–을 하드코딩 했습니다. 이는 데이터를 처리하는 좋은 방법이 아닙니다.

둘 이상의 컴포넌트에서 동일한 데이터를 사용하는 경우 배열을 한번만 만들어 애플리케이션의 컴포넌트들이 공유하는 방법을 사용하는 것이 좋습니다.

`Stories.vue`와 `Famous.vue`는 같은 `stories` 배열을 사용합니다. 이번에 데이터를 공유하는 두가지 방법을 살펴봅니다.

1. 컴포넌트 속성을 사용합니다.
2. 전역 저장소를 사용합니다.

## 속성을 이용한 공유

첫번째 할 일은 `stories` 배열을 App 컴포넌트로 옮기는 것입니다.

src/App.vue

```
1 <script>
2 ...
3
4 export default {
5   components: {
6     ...
7   },
8   data () {
9     // 이야기 배열입니다
10    return {
11      stories: [
12        {
13          plot: 'My horse is amazing.',
14          writer: 'Mr. Weebl',
15          upvotes: 28,
16          voted: false
17        },
18        {
19          plot: 'Narwhals invented Shish Kebab.',
20          writer: 'Mr. Weebl',
```

```
21      upvotes: 8,
22      voted: false
23    },
24    {
25      plot: 'The dark side of the Force is stronger.',
26      writer: 'Darth Vader',
27      upvotes: 52,
28      voted: false
29    },
30    {
31      plot: 'One does not simply walk into Mordor',
32      writer: 'Boromir',
33      upvotes: 74,
34      voted: false
35    }
36  ]
37 }
38 }
39 }
40 </script>
```

---

다음 단계는 Stories와 Famous 컴포넌트에서 data()를 제거하고 stories 속성을 선언하는 것입니다.

첫번째 컴포넌트를 살펴봅니다.

src/components/Stories.vue

```
1 <script>
2   export default {
3     props: [ 'stories' ]
4   }
5 </script>
```

---

App.vue 안에서 컴포넌트를 참조하는 방법을 수정해야 합니다.

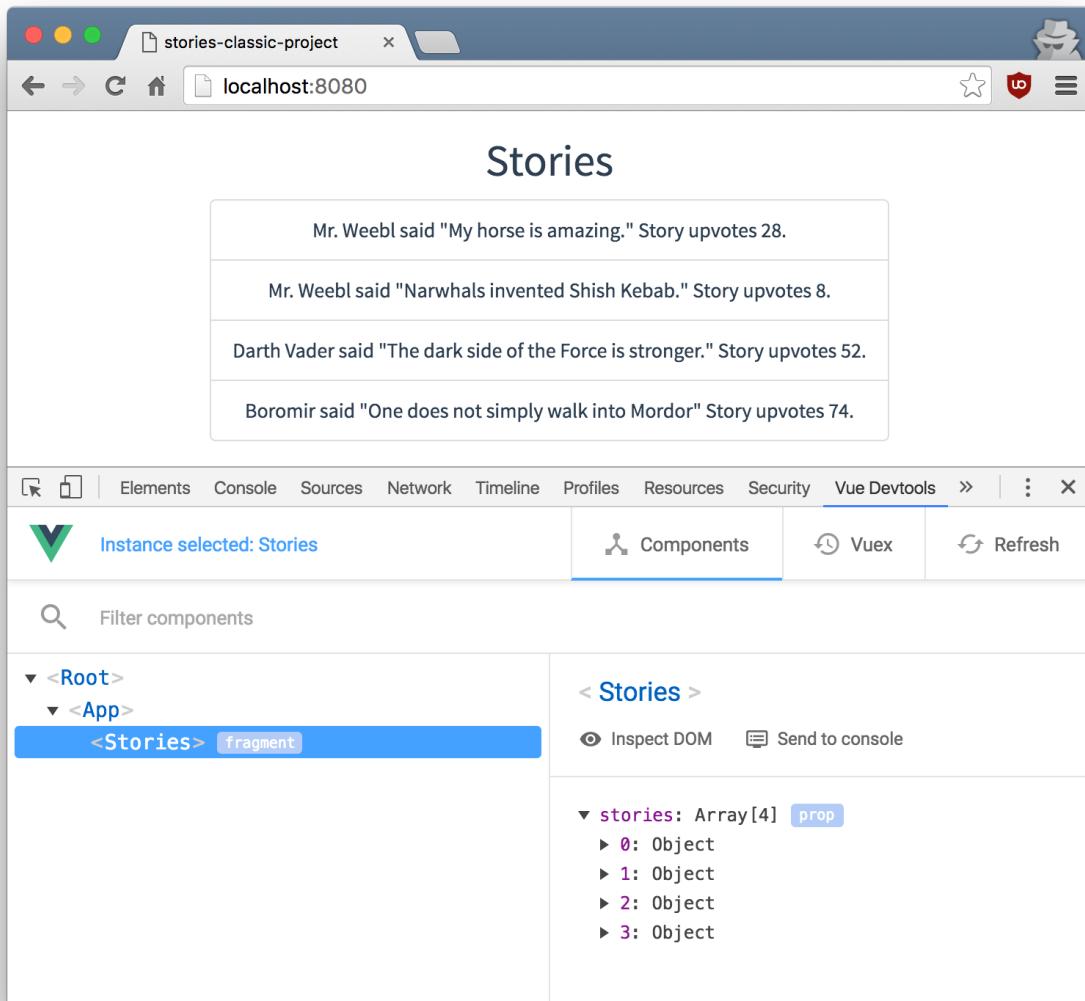
src/App.vue

---

```
1 <template>
2   <div id= " app " >
3     ...
4     <stories :stories= " stories " ></stories>
5     ...
6     <p>
7       Welcome to your Vue.js app!
8     </p>
9   </div>
10 </template>
```

---

여기서 stories 속성을 stories 배열에 바인딩합니다.



### 속성을 이용한 동일한 결과

잘 작동할 것입니다. 부모 컴포넌트에서 stories를 가져왔습니다!

App.vue 안에서 참조되지 않기 때문에 Famous 컴포넌트에서는 동일하게 할 수 없습니다. 배열을 Register 컴포넌트에 전달한 후 Famous로 전달해야 합니다.

src/App.vue

```
1 <template>
2   <div id= " app " >
3     ...
4     <register :stories= " stories " ></register>
5     ...
6   </div>
7 </template>
```

src/components/Register.vue

```
1 <template>
2   <h2>Register Form</h2>
3   ...
4   <famous :stories= " stories " ></famous>
5 </template>
6
7 <script>
8   import Famous from './Famous'
9
10  export default {
11    components: {
12      Famous
13    },
14    props: [ ' stories ' ]
15  }
16 </script>
```

src/components/Famous.vue

```
1 <script>
2   export default {
3     props: [ ' stories ' ],
4
5     computed: {
6       famous () {
7         return this.stories.filter(function (item) {
8           return item.upvotes > 50
9         })
10      }
11    }
12  }
13 </script>
```

이러한 구현은 동작하는 예제이나 Famous 컴포넌트는 독립적이지 않아 효과적이지 않습니다. 즉, 루트 컴포넌트 App.vue에서 데이터를 전달하지 않는 한 원할 때마다 사용할 수 있는 방법은 없습니다.

독립적이지 않은 컴포넌트가 깊이 중첩 되어 있는 시나리오에서는 컴포넌트 간 불필요한 속성을 전달해야만 사용할 수 있습니다. 이 경우 Register의 사이드바 위젯에서 Famous를 사용하려면 stories 배열을 전달 받기 위해 긴 과정이 필요합니다.

App → Register → Sidebar → WidgetX → Famous

## 전역 저장소

처음에는 “속성을 이용한 방식”이 좋지만 Famous 컴포넌트에서 볼 수 있듯 프로젝트가 커지고 컴포넌트가 중첩됨에 따라 컴포넌트 간 데이터 관리가 어려워집니다.

이제 이전 예제의 데이터를 다루기 쉽게 바꾸어야 합니다. stories 데이터를 .js 파일로 추출하여 상수에 저장한 다음 나중에 원하는 위치에서 가져올 수 있습니다.

방금 만든 js 파일 이름을 store.js로 정하고 /src에 옮깁니다.

src/store.js

---

```

1  export const store = {
2    stories: [
3      {
4        plot: 'My horse is amazing.',
5        writer: 'Mr. Weebly',
6        upvotes: 28,
7        voted: false
8      },
9      {
10        plot: 'Narwhals invented Shish Kebab.',
11        writer: 'Mr. Weebly',
12        upvotes: 8,
13        voted: false
14      },
15      {
16        plot: 'The dark side of the Force is stronger.',
17        writer: 'Darth Vader',
18        upvotes: 52,
19        voted: false
20      },
21      {
22        plot: 'One does not simply walk into Mordor',
23        writer: 'Boromir',
24        upvotes: 74,

```

```

25     voted: false
26   }
27 ]
28 }
```

---



## 주의사항

이제 `stories` 속성은 모든 파일에서 제거되어야 합니다. 데이터 저장 방법을 변경했기 때문에 충돌이 발생하고 빌드가 실패할 수 있습니다.

`store.js`안에 모든 데이터를 저장한 후 ES6 모듈 문법을 사용하여 `Stories.vue`안에서 가져올 수 있습니다.

src/components/Stories.vue

```

1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         //store.stories를 사용할 수 있습니다
8         store
9       }
10    },
11    created () {
12      console.log( 'stories' )
13    }
14  }
15 </script>
```

---

`store` 객체를 가져와야 하기 때문에 컴포넌트의 템플릿을 수정해야 합니다.

src/components/Stories.vue

```

1 <template>
2   <ul class= "list-group" >
3     <li v-for= "story in store.stories" class= "list-group-item" >
4       {{ story.writer }} said {{ story.plot }}
5       Story upvotes {{ story.upvotes }}.
6     </li>
7   </ul>
8 </template>
```

---

배열의 항목(store.stories)을 렌더링하기 위해 **v-for**를 사용합니다. 이야기 목록은 이전과 동일하게 표시 됩니다.

컴포넌트의 stories 속성을 store.stories에 직접 바인딩함으로써 템플릿을 변경하지 않고도 동일한 작업을 할 수 있습니다.

src/components/Stories.vue

---

```

1 <script>
2   data () {
3     return {
4       // stories를 직접 바인딩 합니다
5       stories: store.stories,
6     }
7   }
8 </script>

```

---

Famous.vue에도 똑같이 적용합니다

src/components/Famous.vue

---

```

1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         stories: store.stories
8       }
9     },
10    computed: {
11      famous () {
12        return this.stories.filter(function (item) {
13          return item.upvotes > 50
14        })
15      }
16    }
17  }
18 </script>

```

---

이야기에 바인딩 되지 않았다면 계산된 속성 famous()는 this.store.stories를 필터링하도록 변경해야합니다.

단언하건대, 일단 사용법에 익숙해지면 전역 객체로 작업하는 것을 좋아할 것입니다.



## 예제 코드

이 장의 예제 코드는 [GitHub<sup>112</sup>](#)에 있습니다.

---

<sup>112</sup><https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter17>

# 컴포넌트 교체

단일 파일 컴포넌트를 사용하는 것은 싱글 페이지 애플리케이션을 Vue로 구축하는 가장 간단한 방법입니다. 지금까지 새로운 프로젝트를 설정하고 .vue 파일을 만든 후 중복 상태를 관리하는 방법을 익혔습니다. 이제 특정 컴포넌트를 교체하는 방법을 알아보아야 합니다.

참고로 앞의 예제에서 App.vue 안에 3개의 컴포넌트가 있고 그 안에 중첩된 일부 컴포넌트가 있습니다. 컴포넌트를 동적으로 교체하기 때문에 동시에 렌더링 되는 경우는 없습니다.

## 동적 컴포넌트

### 특수 속성 `is`

예약어 `<component>`를 통해 동일한 마운트 지점을 사용하여 여러 컴포넌트를 동적으로 전환할 수 있습니다. 이 때 특수 속성 `is`를 사용합니다.

src/App.vue

---

```
<template>
  <div id= " app " >
    <component is= " hello " >/</component>
    <p>
      This is very useful...
    </p>
  </div>
</template>

<script>
import Hello from './components/Hello'
// Hello 컴포넌트는 데이터의 "msg" 속성을 반환합니다
export default {
  components: {
    Hello
  }
}
</script>
```

---

새 프로젝트를 만들고 Hello.vue 파일을 수정합니다. 앞과 똑같은 결과이지만 `<component is= " hello " >` 엘리먼트를 사용하고 있습니다. Hello 컴포넌트는 `is` 속성에 바인딩 되어

있습니다. 어떻게 동작하는지 확인하려면 링크를 클릭하여 두 다른 컴포넌트를 변경하는 다음 예제를 확인하세요.

먼저, Greet.vue라는 다른 메시지를 사용하여 유사한 컴포넌트를 작성합니다.

src/Greet.vue

---

```
<template>
  <div class= "greet" >
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  data () {
    return {
      msg: 'No! I want to use the <component> element!'
    }
  }
}
</script>
```

---

Greet가 나타나는 메시지를 만들었습니다! App에 가져온 후 사용자에게 2개의 컴포넌트를 교체할 수 있는 기능을 제공합니다.

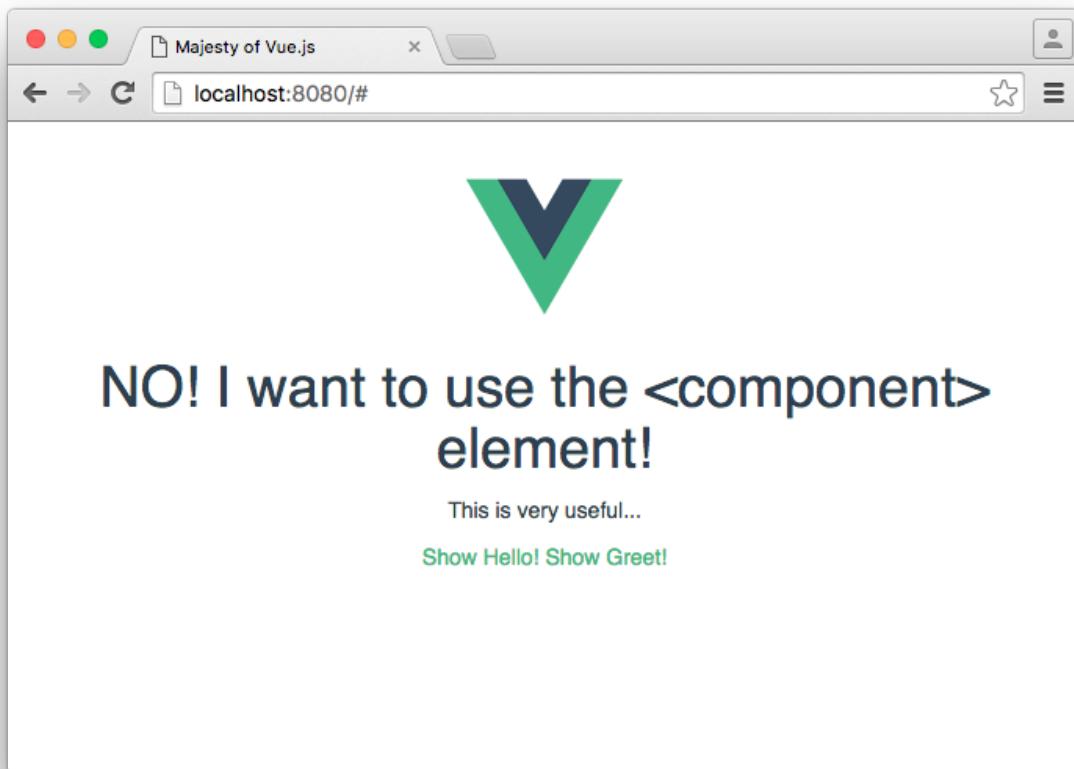
src/App.vue

---

```
<template>
  <div id= "app" >
    <img class= "logo" src= "./assets/logo.png" >
    <component :is= "currentComponent" >
      <!-- this.currentComponent가 변경되면 컴포넌트가 바뀝니다 -->
    </component>
    <p>
      This is very useful...
    </p>
    <a href= "#" @click= "currentComponent = 'hello'" >Show Hello</a>
    <a href= "#" @click= "currentComponent = 'greet'" >Show Greet</a>
  </div>
</template>

<script>
import Hello from './components/Hello'
import Greet from './components/Greet'
```

```
export default {
  components: {
    Hello,
    Greet
  },
  data () {
    return {
      currentComponent: 'hello'
    }
  }
}
</script>
```



Greet.vue

특별한 속성 `is`를 `currentComponent`에 바인딩 하였습니다. 값이 변경될 때 표시되는 컴포넌트 또한 바뀝니다. 보여지는 화면을 바꾸려면 두 링크 중 하나를 클릭하여 `currentComponent`의 값을 변경하면 됩니다.

여러 컴포넌트간 동적인 전환이 필요한 경우 이 방법이 유용할 수 있습니다.

## 네비게이션

이전 예제에서 .vue 파일을 사용하여 소셜 네트워크를 만들어 보았습니다. 여기에는 Login, Registration 등의 컴포넌트가 있습니다. 이제 탭 시스템을 사용하여 컴포넌트를 탐색할 수 있습니다.

Register에는 가장 인기있는 트랜드를 나타내는 Famous 컴포넌트가 있습니다.

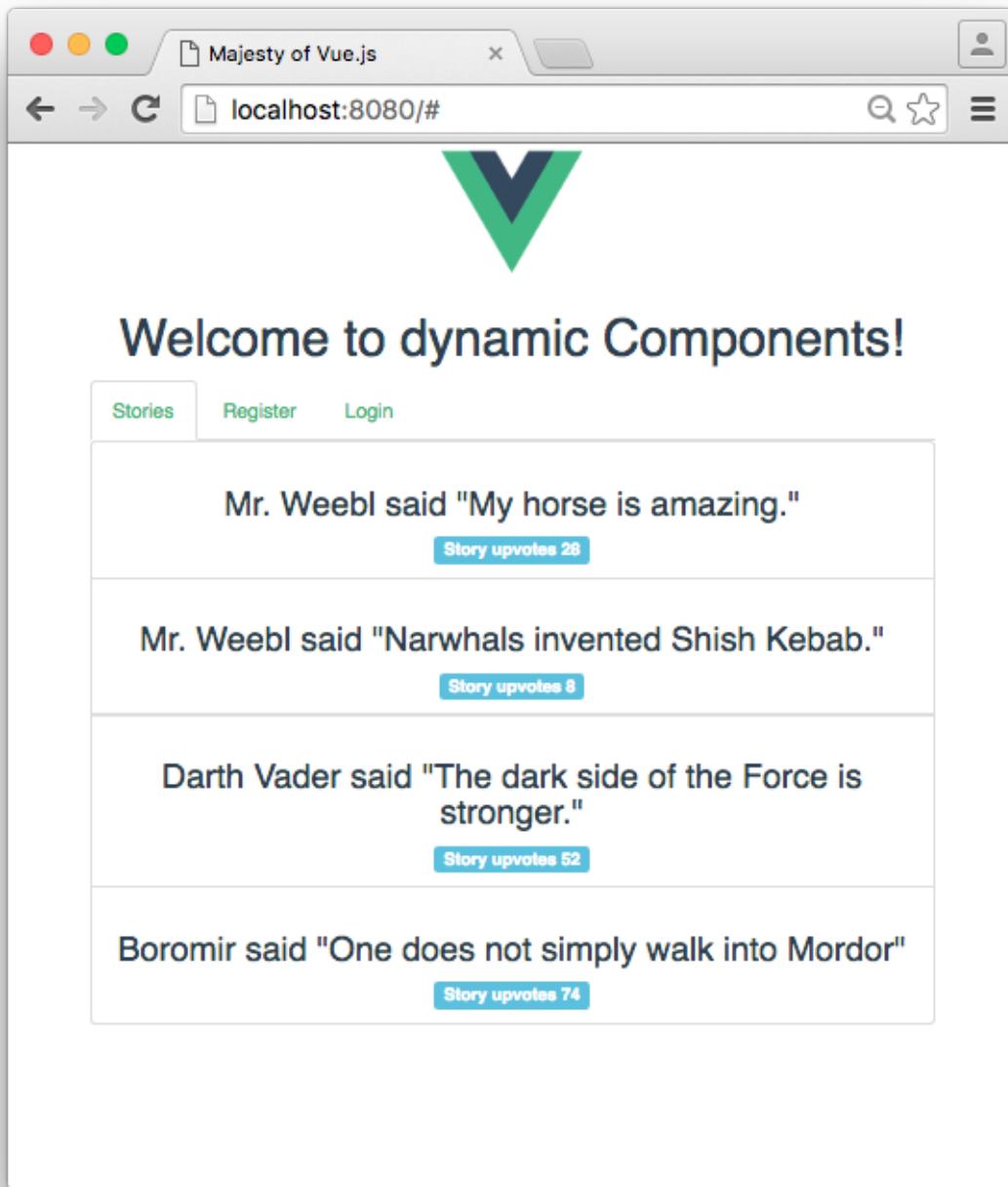
다음 예제를 자세히 읽어보세요.

src/App.vue

```
1 <template>
2   <div id="app">
3     
4     <h1>Welcome to dynamic Components!</h1>
5     <ul class="nav nav-tabs">
6       <!-- 조건부 'active' 클래스 설정 -->
7       <li v-for="page in pages" :class="isActivePage(page) ? 'active' : ''">
8         <!-- 링크를 이용해 탭 사이를 이동합니다 -->
9         <a @click="setPage(page)">{{page | capitalize}}</a>
10        </li>
11      </ul>
12      <component :is="activePage"></component>
13    </div>
14  </template>
15
16 <script>
17   import Vue from 'vue'
18   import Login from './components/Login.vue'
19   import Register from './components/Register.vue'
20   import Stories from './components/Stories.vue'
21
22   Vue.filter('capitalize', function (value) {
23     return value.charAt(0).toUpperCase() + value.substr(1)
24   })
25
26   export default {
27     components: {
28       Login,
29       Register,
30       Stories
31     },
32     data () {
```

```
33     return {
34         // 매번 렌더링할 페이지
35         pages: [
36             'stories',
37             'register',
38             'login'
39         ],
40         activePage: 'stories'
41     }
42 },
43 methods: {
44     setPage (newPage) {
45         this.activePage = newPage
46     },
47     isActivePage (page) {
48         return this.activePage === page
49     }
50 }
51 }
52
53 </script>
```

---



각 컴포넌트를 위한 페이지

한번 자세히 살펴보겠습니다.

pages 배열은 렌더링하려는 컴포넌트를 가지고 있습니다. **v-for** 디렉티브를 이용하여 각각의 템을 만듭니다.

탭 사이를 탐색하기 위해 `setPage` 메소드를 만들었습니다.

`activePage` 속성은 처음에 '`stories`'로 설정됩니다. 탭을 클릭하면 `activePage`가 변경되어 표시하려는 컴포넌트의 이름으로 바꿉니다.

어떤 탭이 활성화 되어야 하는지 결정하기 위해 현재 `activePage` 속성이 현재 컴포넌트의 이름과 일치하여 `active` 클래스를 추가하는 인라인 `if`를 적용합니다.

각 탭의 첫글자를 대문자로 만드려면 `capitalize` 이름을 갖는 `Vue.filter()`를 사용합니다. 이는 텍스트 보간 내에서 사용합니다.

이러한 몇가지 간단한 코드를 사용하여 간단한 컴포넌트 교체를 이용한 네비게이션 시스템을 완성하였습니다.



## 예제 코드

이 장의 예제 코드는 [GitHub](#)<sup>113</sup>에 있습니다.

---

<sup>113</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/codes/chapter18>

# Vue 라우터

라우팅은 일반적으로 애플리케이션이 클라이언트 요청에 응답하는 방법을 결정하는 것을 말합니다. 웹 브라우저의 요청은 라우팅 없이 애플리케이션으로 전달되지 않습니다. 라우터는 웹 서버가 정확하고 적절한 사용자의 정보를 가져올 수 있도록 도와줍니다. 라우터는 선로를 변경하는 운영자에게 선로를 변경할 시기를 알려주는 기차역의 역장과 같은 역할을 합니다

이전 장에서 익힌 컴포넌트 교체는 라우팅을 위한 것입니다. Vue.js의 공식 라우터는 vue-router라고 부릅니다. Vue.js 코어와 깊이 통합되어 싱글 페이지 애플리케이션을 쉽게 만들 수 있게 도와줍니다. vue-router는 이해하기 쉬우며 설치와 사용도 쉽습니다.

주요 기능으로는

- 중첩 라우트/뷰 매핑
- 모듈식 컴포넌트 기반 라우터 구성
- 라우트 파라미터, 쿼리, 와일드카드
- Vue.js 트랜지션 시스템에 기반한 트랜지션 효과
- 세분화된 네비게이션 컨트롤
- 자동으로 active 클래스를 추가하는 링크
- HTML5 히스토리 모드 또는 해시 모드(IE9에서 자동으로 지정됩니다)
- 히스토리 모드로 돌아갈 때 이전 스크롤 위치 복원



## 정보

위 내용은 제공하는 기능의 일부입니다. 자세한 내용은 [Github<sup>114</sup>](#)에서 확인하세요. 또한 [공식 한글 문서<sup>115</sup>](#)를 확인하세요.

## 설치하기

플러그인 설치 방법은 여러가지입니다. cdn, NPM과 Bower를 사용할 수 있습니다. 이 책에서는 NPM을 이용합니다.

```
>_ npm install vue-router
```

터미널에서 명령어를 입력하면 프로젝트 디렉터리의 node\_modules 디렉터리에 설치됩니다. 완료되면, main.js 파일을 열어 아래 내용을 추가합니다.

<sup>114</sup><https://github.com/vuejs/vue-router>

<sup>115</sup><http://router.vuejs.org/kr/index.html>

src/main.js

---

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)
```

---

Vue.js 플러그인을 사용하는 방법은 위 코드처럼 `Vue.use()`를 이용합니다. `Vue.use`의 더 자세한 내용은 [가이드<sup>116</sup>](#)를 확인하세요.

## 사용방법

첫번째 단계는 나중에 추가적인 옵션을 전달할 라우터를 인스턴스를 만드는 것입니다.

src/main.js

---

```
...
const router = new VueRouter({
  routes // routes: routes의 축약
})
```

---

이제 라우트를 정의합니다. 각 라우트는 컴포넌트와 매핑합니다. 즉, 예제에서 사용했던 `*.vue` 파일 경로를 추가합니다.

라우터의 경로 매핑을 정의하려면 라우트 객체를 전달할 수 있는 `routes` 배열을 정의합니다.

src/main.js

---

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Hello from '../src/components/Hello.vue'
import Login from '../src/components/Login.vue'
```

```
Vue.use(VueRouter)
```

---

```
const routes = [
  { path: '/', component: Hello },
  { path: '/login', component: Login }
]
```

---

2개의 라우트를 배열에 정의하였습니다.

---

<sup>116</sup><http://kr.vuejs.org/api/#Vue-use>

1. `http://localhost:3000/`(포트는 지정한 설정에 따라 다릅니다)에 접속하면 `Hello.vue`를 보여줍니다.
2. `http://localhost:3000/login`에 접속하면 `Login.vue` 컴포넌트를 보여줍니다.

## 정보

`{ path: '/login', component: Login }`처럼 작성할 수 있는 이유는 `Login` 컴포넌트가 코드의 맨 위에 정의되었기 때문입니다. `require()`를 이용할 수도 있습니다.

```
{ path: '/login', component: require( 'Login' ) }
```

다음 단계로 라우터의 아울렛을 만듭니다. 가이드에는 라우터가 렌더링할 루트 컴포넌트가 필요하다고 안내합니다. 이 경우 `App.vue` 컴포넌트를 루트 컴포넌트로 사용합니다. 루트 인스턴스를 만들고 마운트합니다. 전체 `app`의 `router`를 인식하도록 만들기 위해 라우터에 `router` 옵션을 만드시 주입해야합니다.

src/main.js

---

```
...
/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<app></app>',
  components: {
    App
  }
})
```

---

템플릿을 `<app></app>`로 설정하면 Vue는 `app`을 id로 가지는 `div`를 `App.vue` 컴포넌트의 템플릿으로 대체합니다. `main.js`파일에 `App`을 이미 가져오고 있기 때문에 설정할 수 있습니다.

`/* eslint-disable no-new */` 주석을 적지 않으면 eslint는 오류를 발생합니다.

<http://eslint.org/docs/rules/no-new> Do not use 'new' for side effects

여기서는 이 규칙을 사용하지 않습니다.

## 팁

eslint 규칙으로 인하여 문제가 발생할 때는 위와 같은 주석을 추가하여 비활성화할 수 있습니다. 예: `/* eslint-disable eqeqeq */` 전체 규칙은 [여기<sup>117</sup>](#)에 있습니다.

이제 탐색을 위해 몇개의 링크를 정의해야합니다. `App.vue`를 열어 수정합니다.

---

<sup>117</sup> <http://eslint.org/docs/rules/>

src/App.vue

```
<template>
  <div id="app">
    
    <h1>Welcome to Routing!</h1>
    <router-link to="/" >Home</router-link>
    <router-link to="/login" >Login</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

<router-view></router-view>는 컴포넌트가 렌더링되는 동안 모든 일이 일어나는 곳입니다.

**router-link**은 사용자가 라우터가 활성화된 앱에서 탐색할 수 있도록 만드는 컴포넌트입니다. **to** 속성은 `main.js`에 정의된 경로와 일치하는 대상 위치를 정의합니다. 기본적으로 **router-link**는 `href` 속성을 원하는 URI로 설정하여 `<a>`태그를 렌더링합니다. 더 복잡한 탐색에는 더 많은 전달인자가 필요할 수 있습니다. 뒤에서 살펴보겠습니다.



## 노트

라우터가 포함된 `vue-cli` 템플릿을 사용하는 경우 `router/index.js`에 라우트 배열을 정의해야 합니다.

## 이름을 가지는 라우트

지금 진행하는 예제처럼 작은 프로젝트에서는 `vue-router`가 요구사항을 잘 충족하고 있지만 프로젝트가 커질수록 더 많은 옵션이 필요할 것입니다. 예를 들어 `/login`을 `/signin`으로 변경하기로 했다면 로그인 페이지로 향하는 모든 링크를 수정해야합니다. 이러한 일이 발생하지 않도록 각 라우트에 이름을 지정할 수 있습니다.

라우트의 이름을 지정하려면 라우트 설정을 변경해야합니다.

src/main.js

---

```
...
const routes = [
  {
    path: '/',
    name: 'home',
    component: Hello
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]
```

---

`name` 속성을 추가하여 라우트에 이름을 부여할 수 있으며, 나중에 탐색을 위한 식별자로 사용할 수 있습니다.

src/App.vue

---

```
<template>
  <div id="app">
    ...
    <router-link :to=" { name: 'home' } " >Home</router-link>
    <router-link :to=" { name: 'login' } " >Login</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

---

링크의 목적지를 정의하는데 문자열(`:to= " /home "`)을 사용하는 대신에 객체(`:to= " { name: 'home' } "`)를 사용합니다. 나중에 자세히 설명합니다.

## 히스토리 모드

계속 진행하기 전에, vue-router와 브라우저 URL간의 연관성을 살펴봅니다. 앞에서 보았듯이 라우트가 변경되면 ‘#’ 기호가 URL에 추가됩니다. 예를 들어, 로그인 페이지로 이동하면 URL은 `/#/login`입니다.

이는 vue-router의 기본 설정인 해시모드이기 때문에 발생합니다. 해시모드는 URL 해시를 사용하여 전체 URL을 시뮬레이션하므로 URL이 변경될 때 페이지가 다시 로드되지 않습니다. 해시를 제거하려면 **히스토리 모드**로 변경해야합니다. 또한 모든 라우터 탐색을 위한 루트

경로를 정의하는 또 다른 옵션인 **base**를 설정합니다. 기본값인 `default`에서 변경하면 실제 브라우저 URL에 항상 새로운 값을 포함하는 라우트가 생깁니다.

**base**를 `/vuejs`로 변경하면 로그인 페이지 주소는 `/vuejs/login`입니다. **base**는 어떤 것으로든 설정할 수 있습니다 여기서는 `/`로 설정합니다.

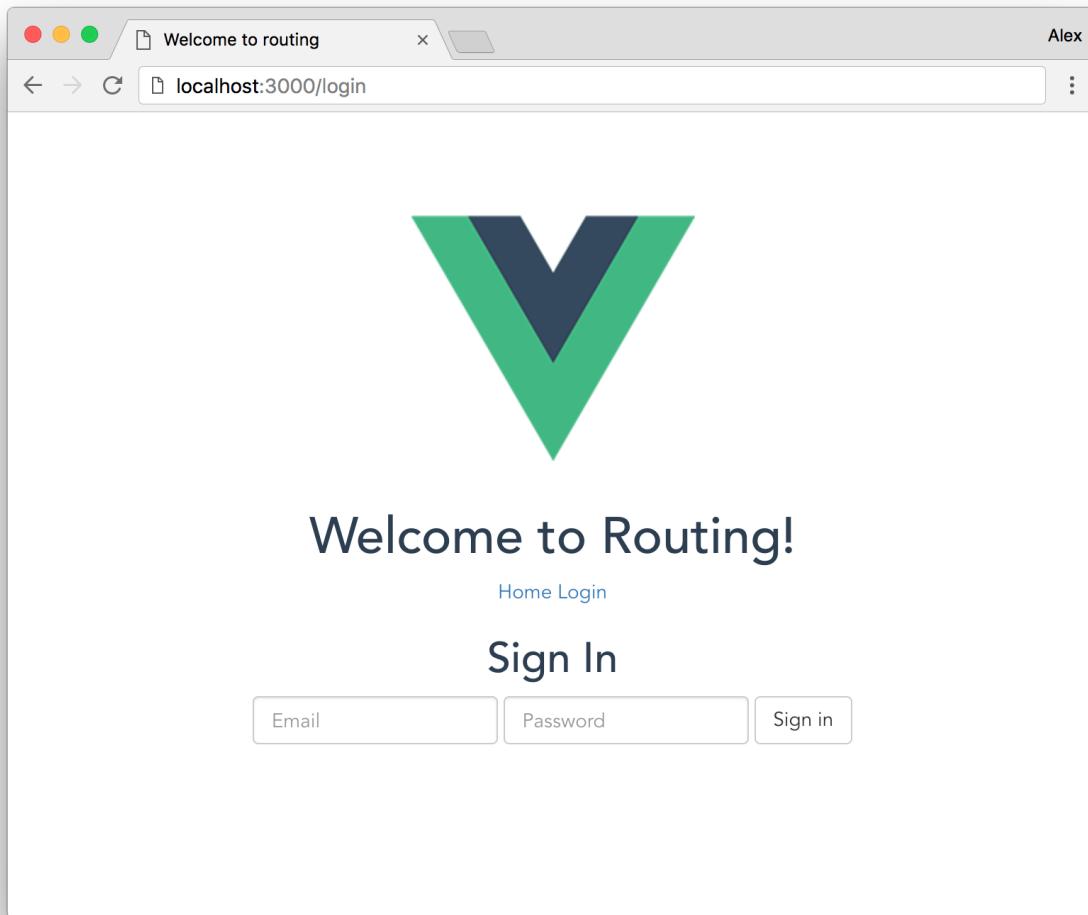
src/main.js

---

```
const router = new VueRouter({
  mode: 'history',
  base: '/',
  routes
})
```

---

이렇게하면 '#'를 URL에서 없앨 수 있습니다.



Neat URLs



## 정보

vue-router 옵션의 상세 내역은 [공식 문서](#)<sup>118</sup>를 확인하세요.

## 중첩 라우트

중첩 라우트는 다른 라우트 안에 있는 라우트입니다. vue-router를 사용하면 매우 간단하게 사용할 수 있습니다.

2개의 하위 페이지로 이야기를 표시하는 페이지를 추가합니다.

<sup>118</sup> <https://router.vuejs.org/kr/api/options.html>

- 한 페이지는 모든 이야기를 보여줍니다. (`StoriesAll.vue`)
- 다른 페이지는 좋아하는 페이지만 보여줍니다. (`StoriesFamous.vue`)

위에서 언급한 컴포넌트와 함께 `App.vue`와 비슷한 래퍼 컴포넌트를 만듭니다.

새 라우트를 추가하는 것으로 시작합니다. `routes` 배열에 `children` 옵션을 사용하고 래퍼 컴포넌트인 `StoriesPage.vue`에 중첩된 `<router-view>`를 추가합니다.

src/main.js

```
import Vue from 'vue'
import App from './App'

import Hello from './components/Hello.vue'
import Login from './components/Login.vue'
import StoriesPage from './components/StoriesPage.vue'
import StoriesAll from './components/StoriesAll.vue'
import StoriesFamous from './components/StoriesFamous.vue'

import VueRouter from 'vue-router'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    component: Hello
  },
  {
    path: '/login',
    component: Login
  },
  {
    path: '/stories',
    component: StoriesPage,
    children: [
      {
        path: '',
        name: 'stories.all',
        component: StoriesAll
      },
      {
        path: 'famous',
        name: 'stories.famous',
        component: StoriesFamous
      }
    ]
  }
]
```

```
        ]
    }
]
```

---

StoriesAll의 path 속성은 ” 입니다. 이는 하위 라우트의 기본 라우트라는 의미이며 /stories와 일치한 URL일 때 렌더링하는 것을 의미합니다. ’ / ’ 를 사용하여 기본 라우트를 정의할수도 있습니다.

StoriesFamous의 내용은 /stories/famous가 매치될 때 출력됩니다.

이 시점에서 컴포넌트의 내용을 표시할 필요는 없습니다. 둘 다 이야기 배열을 표시합니다. 래퍼 컴포넌트인 StoriesPage는 자식 컴포넌트의 내용을 렌더링하기 위해 링크 2개와 <router-view>태그를 추가합니다.

src/StoriesPage.vue

---

```
<template>
  <div>
    <h2>Stories</h2>
    <!-- navigation -->
    <router-link :to= " {name: ' stories.all ' } " >All</router-link>
    <router-link :to= " {name: ' stories.famous ' } " >Trending</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

---

## 자동 active 클래스 추가

현재 보고 있는 페이지를 강조하는 링크를 표시하면 좋지 않을까요? vue-router는 css 클래스를 활성화된 링크에 추가하는 멋진 기능이 있습니다. 이 클래스는 vue-router-active입니다. CSS에 스타일을 적용할 규칙만 추가하면 됩니다. App.vue 컴포넌트에 추가합니다.

src/App.vue

---

```
...
<style type= " text/css " >
  .router-link-active {
    color: green;
  }
</style>
```

---

이제 페이지를 방문할 때마다 링크가 녹색으로 바뀝니다. 이 기능을 사용하면 Home 링크가 항상 녹색인 것을 볼 수 있습니다. 이는 Home 경로가 /이기 때문에 발생합니다. 예를 들어

/login로 이동해도 Home이 활성 상태를 유지합니다. 이러한 상황을 방지하기 위해 특정 링크에 exact 속성을 추가해야 합니다.

네비게이션 링크는 아래와 같습니다.

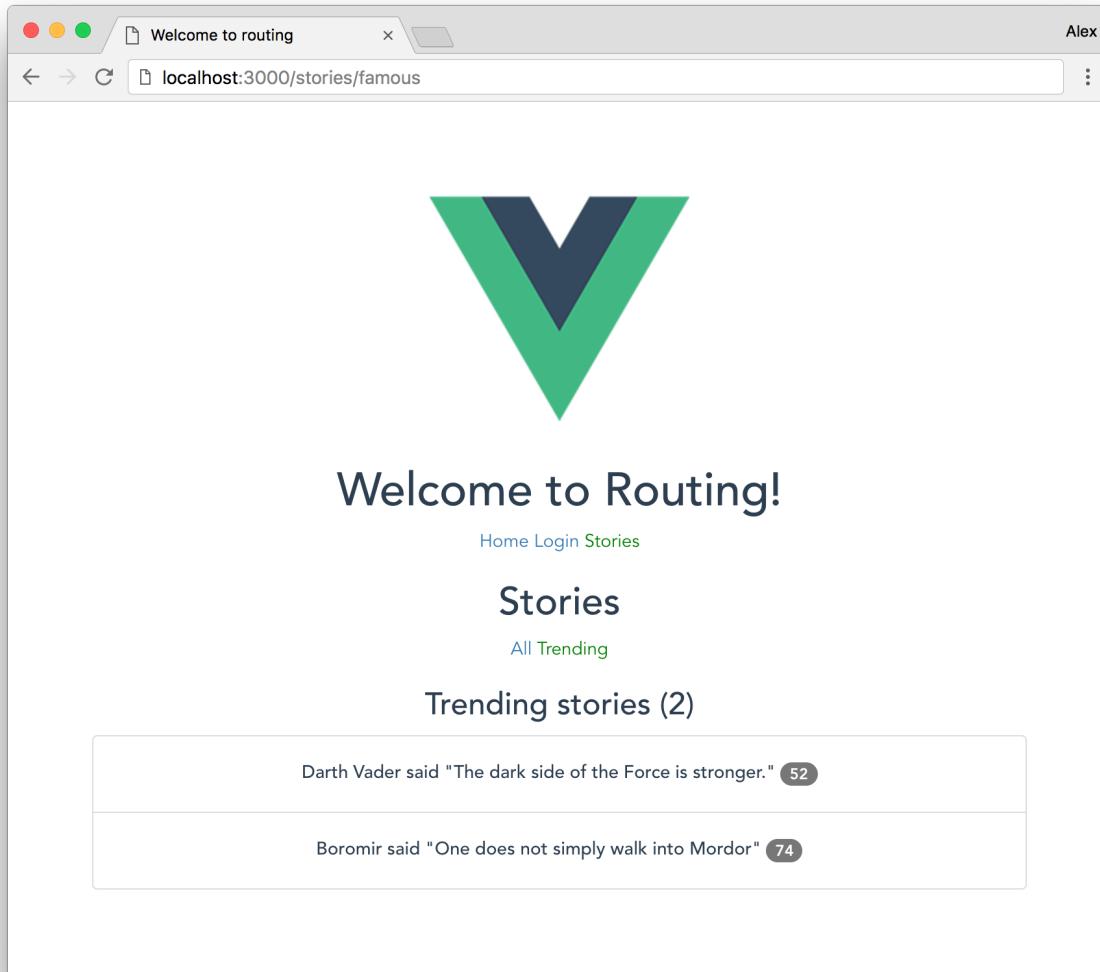
src/App.vue

---

```
<template>
  <div>
    ...
    <router-link :to= "{ name: 'hello' }" exact>Home</router-link>
    <router-link :to= "{ name: 'login' }" >Login</router-link>
    <router-link :to= "{ name: 'stories.all' }" >Stories</router-link>
    <router-view></router-view>
  </div>
</template>
```

---

StoriesPage.vue의 첫번째 링크에 exact를 추가했습니다. 좋은 점은 내부 탐색에서도 활성 링크가 강조된다는 것입니다.



Active Class

## 사용자 정의 활성 클래스

생성자 옵션 `linkActiveClass`을 사용하여 `active-class` 속성을 사용하거나 전역으로 특정 링크에 대한 활성 클래스(`router-link-active`) 이름을 변경할 수 있습니다.

지역범위의 사용자 정의 active 클래스

---

```
<router-link :to=" { name: 'hello' } " active-class=" my-active-class " exact>
  Home
</router-link>
```

---

전역범위 사용자 정의 active 클래스

---

```
const router = new VueRouter({
  mode: 'history',
  base: '/',
  linkActiveClass: 'my-active-class',
  routes
})
```

---

## 라우트 객체

구문분석된 경로의 모든 정보는 노출된 라우트 컨텍스트 객체(라우트 객체라고도 부릅니다)에서 액세스할 수 있습니다.

라우트 객체는 router-enabled 앱의 모든 컴포넌트에 주입될 것이고 `this.$route`처럼 접근 가능합니다. 라우트 변경이 일어날 때마다 업데이트됩니다.

아래에 `$route` 객체의 속성 목록입니다.

속성	상세
path	현재 경로와 동일한 문자열로 항상 절대 경로로 해석됩니다 예를 들어 <code>/foo/bar</code> .
params	동적 세그먼트와 스타 세그먼트의 키/값을 포함하는 객체입니다. 자세한 내용은 아래를 참조하세요.
query	쿼리 문자열의 키/값 쌍을 포함하는 객체입니다. 예를 들어, <code>/foo?user=1</code> 라우트의 경우, <code>\$route.query.user == 1</code> 을 사용할 수 있습니다.
hash	해시가 존재하는 경우 <code>#</code> , 존재하지 않는 경우 빈 문자열이 됩니다.
fullPath	쿼리 및 해시를 포함한 전체 URL 입니다.
matched	현재 경로의 모든 중첩 경로 세그먼트에 대한 라우트가 포함된 배열입니다. 라우트 레코드는 라우트 배열(및 하위 배열)에 있는 객체의 복사본입니다.
name	현재 라우트의 이름입니다. (존재하는 경우)

## 동적 세그먼트

vue-router는 동적 세그먼트를 사용하여 경로를 구성하는 기능을 제공합니다. 동적 세그먼트는 선행 콜론이 있는 세그먼트입니다. 값을 변경할 수 있기 때문에 동적이라고 부릅니다.



### 정보

URL 세그먼트는 슬래시로 구분된 URL 또는 경로의 일부입니다. `/user/:id/posts` 경로라면 `user`, `:id` 및 `posts`는 각각 하나의 세그먼트가 됩니다. 이 경로에서 `:id`는 동적 세그먼트이며 전달받은 값과 일치합니다. 예를 들어 `/user/11/posts`, `/user/37/posts` 등입니다.

동적 세그먼트를 포함하는 경로가 일치하면 동적 세그먼트를 `$route.params`에서 사용할 수 있습니다. 이 예에서 동적 세그먼트를 사용하여 `id`로 특정 이야기에 액세스하여 편집할 수 있는 뷰를 만듭니다.

src/main.js

```
1 const routes = [
2   // 다른 라우트
3   {
4     path: ':id/edit',
5     name: 'stories.edit',
6     component: StoriesEdit
7   }
8   ...
9 ]
```

이제 `StoriesAll.vue`와 `StoriesEdit.vue`를 연결하는 방법이 필요합니다. 파일을 수정합니다.



### 노트

`StoriesEdit.vue` 파일을 만들었습니다. 라우팅이 완료되면 곧 볼 수 있습니다.

src/component/StoriesAll.vue

```
1 <template>
2   <div class= " " >
3     <h3>All Stories ({{stories.length}})</h3>
4     <ul class= " list-group " >
5       <li v-for= " story in stories " class= " list-group-item " >
6         <div class= " row " >
7           <h4>{{ story.writer }} said {{ story.plot }}</h4>
8           <span class= " badge " >{{ story.upvotes }}</span>
9         </div>
10        <router-link
11          :to= " { name: 'stories.edit' } " tag= " button "
12            class= " btn btn-default " exact
13          >
14            Edit
15          </router-link>
16        </div>
17      </li>
18    </ul>
19  </div>
20 </template>
21
22 <script>
23 import {store} from '../store.js'
24
25 export default {
26   data () {
27     return {
28       stories: store.stories
29     }
30   },
31   mounted () {
32     console.log('stories')
33   }
34 }
35 </script>
```



## 노트

예제 파일은 이전 버전과 동일하지만 기능에 영향을 주지 않는 대부분의 스타일을 약간 변경하였습니다. 주목할만한 변화는 `store.js`내의 각 스토리에 `id`를 추가한 것입니다.

`stories.edit` 라우트를 위한 버튼을 추가하였습니다. 또한 이야기의 `id`를 경로에 전달해야 하나 충분하지 않습니다.

이를 위해 `to` 속성을 수정하고, `:id`를 각 이야기의 해당 `id`로 바꿉니다.

src/component/StoriesAll.vue

```

1 <template>
2   <div>
3     <h3>All Stories ({{stories.length}})</h3>
4     <ul class= "list-group" >
5       <li v-for= "story in stories" class= "list-group-item" >
6         <h4>{{ story.writer }} said {{ story.plot }}</h4>
7         <span class= "badge" >{{ story.upvotes }}</span>
8       </li>
9       <router-link
10        :to= "{ name: 'stories.edit' , params: { id: story.id } } "
11        tag= "button" class= "btn btn-default" exact>
12          Edit
13        </router-link>
14      </li>
15    </ul>
16  </div>
17 </template>
```

`<router-link>`를 출력할 때 `<a>` 태그 대신 `<button>` 태그를 사용하기를 원합니다. `tag` 속성을 사용하여 렌더링할 태그를 지정할 수 있습니다. 렌더링된 태그는 클릭 이벤트를 수신합니다.

`$route.params` 내에서 선택한 이야기의 `id`는 목적지에 도착하면 사용할 수 있습니다. 이를 통해 사용자가 편집하고자 하는 이야기를 골라내고 가져올 수 있습니다.

편집이 이루어지는 `StoriesEdit.vue`을 살펴볼 차례입니다.

src/components/StoriesEdit.vue

```

1 <template>
2   <div class= "row" >
3     <h3>Editing</h3>
4     <form>
5       <div class= "form-group col-md-offset-2 col-md-8" >
6         <input class= "form-control" v-model= "story.plot" >
7       </div>
8       <div class= "form-group col-md-12" >
9         <button @click= "saveChanges(story)" class= "btn btn-success" >
10           Save changes
11         </button>
12       </div>
```

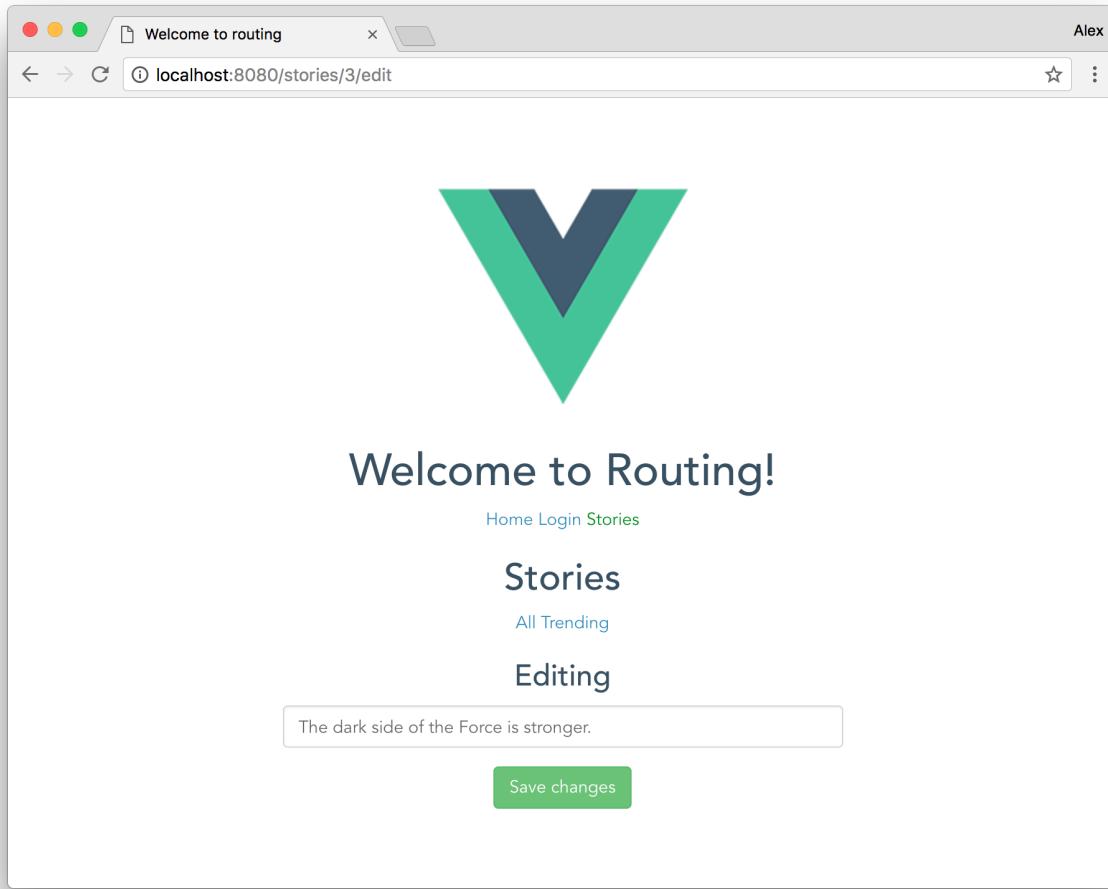
```
13      </form>
14  </div>
15 </template>
16
17 <script>
18 import {store} from '../store.js'
19
20 export default {
21   data () {
22     return {
23       story: {}
24     }
25   },
26   methods: {
27     isTheOne (story) {
28       return story.id === this.id
29     },
30     saveChanges (story) {
31       // 나중에 사용합니다
32     }
33   },
34   mounted () {
35     this.story = store.stories.find(this.isTheOne)
36   }
37 }
38 </script>
```

---

story의 id를 사용하여 JavaScript의 `find` 메소드<sup>119</sup>를 사용하여 stories 배열에서 원하는 story를 선택합니다.

선택한 이야기는 편집할 준비가 되어있습니다. 이야기의 id가 URL에 표시됩니다.

<sup>119</sup>[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/find](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find)



### 선택된 이야기 수정

버튼을 사용하여 StoriesEdit 페이지를 방문할 수 있지만 브라우저의 주소창에 URL을 /stories/2/edit을 직접 입력하려고 시도하면 오류가 발생하고 컴포넌트가 렌더링 되지 않습니다.

이유는 활성화된 이야기를 찾기 위해 엄격한 비교<sup>120</sup>를 사용했기 때문입니다. 페이지를 직접 접속하면 id가 문자열(숫자가 아님)로 전달됩니다. 그러므로 isTheOne는 항상 거짓입니다.

<sup>120</sup> [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison\\_Operators#Identity](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity)

src/components/StoriesEdit.vue

---

```
isTheOne (story) {
  // 숫자가 아니므로 거짓입니다.
  return story.id === this.id
}
```

---

쉽게 수정하려면 **Number** 객체 래퍼를 사용하여 **id**를 숫자로 변환하는 것 입니다.

src/components/StoriesEdit.vue

---

```
export default {
  ...
  mounted () {
    this.id = Number(this.$route.params.id)
  }
}
```

---

브라우저에서 확인하면 버튼을 클릭했을때와 마찬가지로 이야기가 표시되는 것을 알 수 있습니다.

하지만 StoriesEdit 컴포넌트를 **\$route**에서 분리하는데 훨씬 좋은 해결책이 있습니다. **id** 매개변수를 라우트 설정안에서 숫자로 캐스팅하고 StoriesEdit 컴포넌트에서 prop로 사용할 수 있습니다.

src/main.js

---

```
const routes = [
  ...
  {
    path: ':id/edit',
    props: (route) => ({ id: Number(route.params.id) }),
    name: 'stories.edit',
    component: StoriesEdit
  },
  ...
]
```

---

이제 stories/:id/edit에 다다르면 라우터는 StoriesEdit 컴포넌트에 :id의 숫자를 속성으로 전달합니다. 빠진 것은 속성을 정의하고 **this.\$route.params.id**를 제거하는 것 입니다. 컴포넌트는 아래와 같습니다.

src/components/StoriesEdit.vue

---

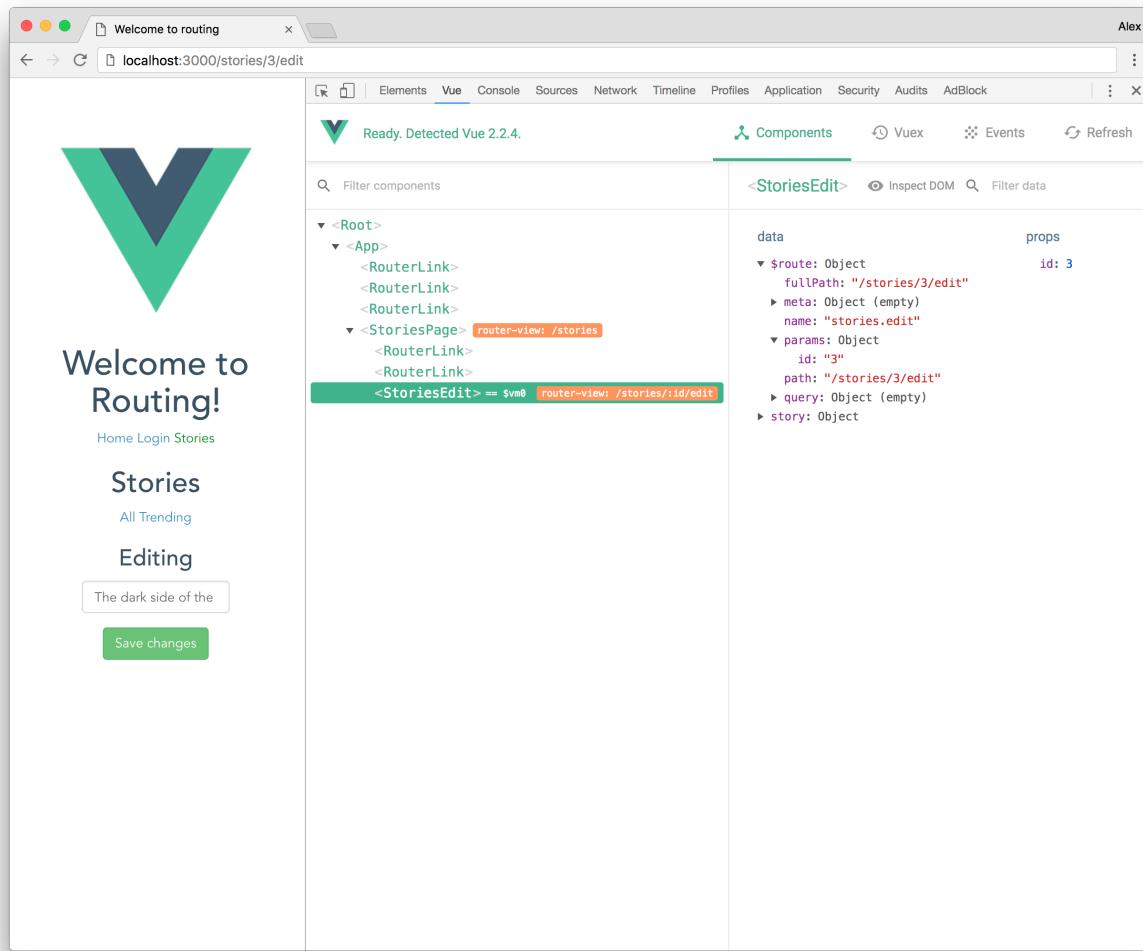
```
<script>
import {store} from '../store.js'

export default {
  props: [ 'id' ],
  data () {
    return {
      story: {}
    }
  },
  methods: {
    isTheOne (story) {
      return story.id === this.id
    }
  },
  mounted () {
    this.story = store.stories.find(this.isTheOne)
  }
}
</script>
```

---

이제 전부입니다. 이제 StoriesEdit 컴포넌트는 라우트에서 분리되어 다음과 같이 사용할 수 있습니다. <stories-edit :id= " story.id " ></stories-edit>.

Vue Devtools를 사용하여 \$route 객체의 속성을 살펴볼 수 있습니다.



\$route 내부

## 라우트 별칭

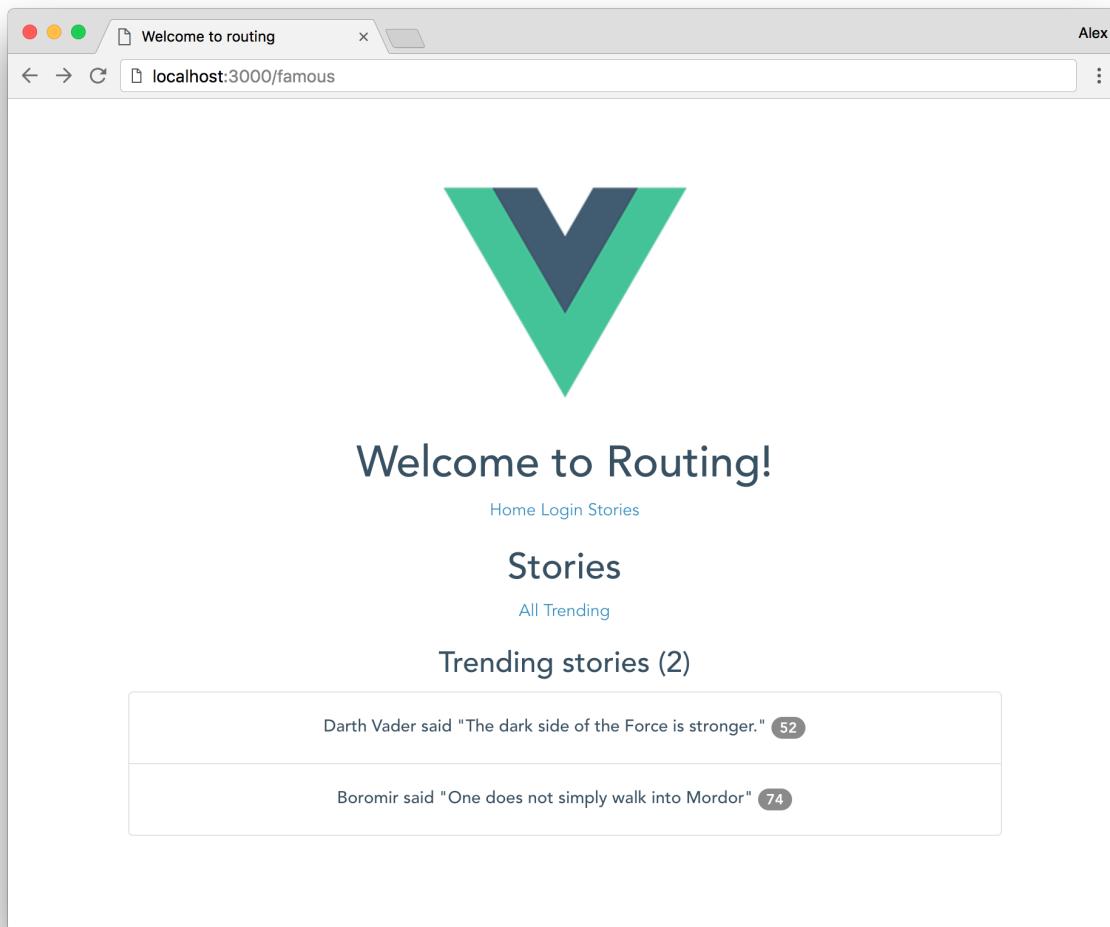
새로운 경로를 정의할 때 일반적으로 경로를 명확하고 대표성을 띠게 만듭니다. 때로 긴 경로나 복잡한 경로를 사용할 수 있습니다. 나중에 처리하기 힘들어질 수도 있습니다.

브라우저를 통해 famous 경로를 보고 싶을 때 '/stories/famous'를 방문해야 합니다. 전역 별칭을 정의하면 이 경로를 더 짧게 만들 수 있습니다. URL이 짧을수록 같은 위치로 연결합니다.

src/main.js

```
{  
  path: 'famous',  
  name: 'stories.famous',  
  // '/stories/famous' 와 같이 '/famous' 와 일치시킵니다.  
  alias: '/famous',  
  component: StoriesFamous  
}
```

이 설정을 사용하면 path 대신 alias를 사용할 수 있습니다.



Using route alias

## 라우트 푸시

특정 시점에서 링크를 이용하지 않고 코드를 이용해 라우트를 탐색할 경우가 있습니다.

라우트를 탐색하려면 **router.push(path)**를 사용합니다. 경로는 문자열 또는 객체를 사용할 수 있습니다.

문자열의 경우 일반적인 경로 형식입니다. 즉 동적 세그먼트를 포함할 수 없습니다. 예를 들어 **router.push( '/stories/11/edit' )**를 사용합니다.

객체를 사용하는 경우 필요한 전달인자를 사용할 수 있습니다.

프로그래밍 방식 네비게이션

---

```
router.push({ path: '/stories/11/edit' })
router.push({ name: 'stories.edit', params: {id: '11'} })
```

---

수정이 완료되면 **router.push**를 사용하여 이야기 목록 페이지로 이동할 수 있습니다.

src/components/StoriesEdit.vue

---

```
1 <script>
2 import {store} from '../store.js'
3
4 export default {
5   props: [ 'id' ],
6   data () {
7     return {
8       story: {}
9     }
10 },
11   methods: {
12     saveChanges (story) {
13       console.log('Saved!')
14       this.$router.push('/stories')
15     },
16     isTheOne (story) {
17       return story.id === this.id
18     }
19 },
20   mounted () {
21     this.story = store.stories.find(this.isTheOne)
22   }
23 }
24 </script>
```

---

`saveChanges` 메소드를 수정했습니다. 호출되면 콘솔에 메시지를 출력하고 `this.$router.push()`를 사용하여 `/stories`로 돌아갑니다.

특정 URL 대신 이전에 방문한 URL로 이동시키려면 `router.back()`을 사용할 수 있습니다.

이 경우 버튼을 추가하고 `router.push` 대신 `router.back`을 사용할 수 있으며 사용자가 이전 페이지로 이동할 수 있습니다. (예: <https://google.com>)

src/components/StoriesEdit.vue

```

1 ...
2 <button @click= "goBack" >Go back</button>
3
4 methods: {
5   ...
6   goBack () {
7     this.$router.back()
8   },
9   ...
10 }
```

이를 위한 또 다른 방법은 `router.go(n)` 메소드입니다. 이 메소드는 히스토리 스택<sup>121</sup>에서 앞 또는 뒤로 이동하는 단계 수를 나타내는 정수 속성의 전달인자를 사용합니다.

```
goBack () {
  this.$router.go(-1)
}
```



## 경고

`$router.back()` 또는 다른 라우터의 메소드를 사용하여 컴포넌트를 라우터에 연결합니다. 연결 해제를 원하면 `window.history.back()`를 사용할 수 있습니다.

## 트랜지션

### 소개

애플리케이션의 다른 페이지로 이동하는 것이 멎진 경우는 없었습니다. 트랜지션을 사용하면 페이지에 들어가는 컴포넌트와 남아있는 컴포넌트에 애니메이션을 적용하여 변경할 수 있습니다.

Vue는 엘리먼트가 DOM에 삽입, 간신 또는 제거될 때 트랜지션 효과를 적용하는 다양한 방법을 제공합니다. 여기에는 다음과 같은 도구가 포함됩니다.

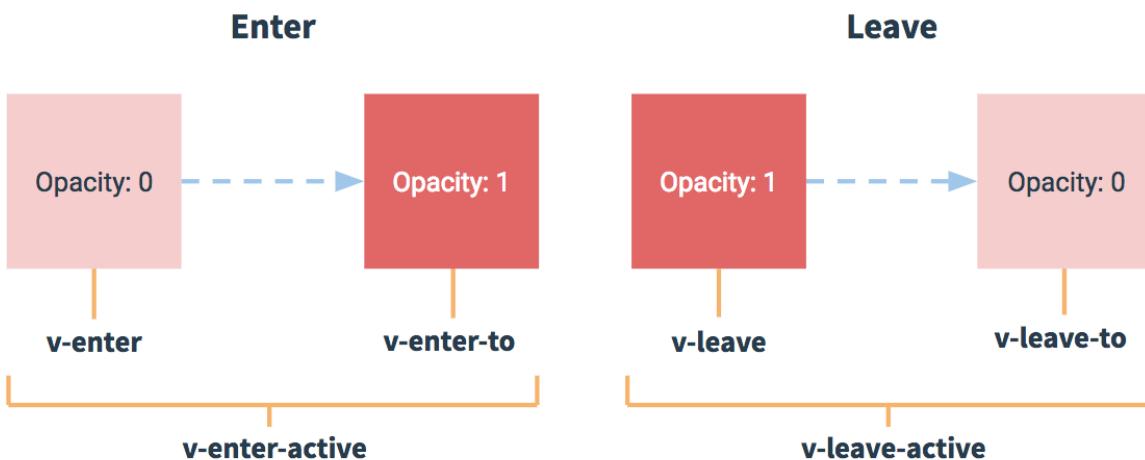
<sup>121</sup> <http://router.vuejs.org/en/essentials/history-mode.html>

- 자동으로 트랜지션 및 애니메이션 클래스를 적용합니다.
- [Animate.css<sup>122</sup>](#), [Velocity.js<sup>123</sup>](#) 등의 써드파티 CSS 또는 JavaScript 애니메이션 라이브러리와 함께 사용할 수 있습니다.
- 트랜지션 혹은 동안 JavaScript를 사용하여 DOM을 직접 조작할 수 있습니다.

여기서는, CSS 클래스를 사용하여 진입과 진출만 다룹니다. 트랜지션에 대해 더 자세히 알고 싶으면 [가이드<sup>124</sup>](#)를 확인하세요.

트랜지션을 사용하려면 `router-view` 컴포넌트를 `transition` 컴포넌트로 감싸야합니다.

Vue는 엘리먼트가 삽입되기 전에 `v-enter` CSS 클래스를 엘리먼트에 추가하고 진입 단계에서 `v-enter-active`를 추가합니다. `v-enter-to`는 트랜지션이 완료되기 전에 추가될 마지막 클래스입니다. 실제로 진입의 종료 상태입니다. 따라서 엘리먼트가 DOM에서 제거될 때 `v-leave`, `v-leave-active` 그리고 `v-leave-to`가 적용됩니다.



### 트랜지션 클래스

`transition`의 이름을 정의하면 위에서 언급한 모든 클래스는 `v` 대신 이름을 사용합니다 그 예는 `fade-enter`, `fade-leave-to` 등입니다.

## 사용하기

`fade`이름을 가지는 트랜지션 효과를 라우트 아울렛에 추가합니다.

<sup>122</sup><https://daneden.github.io/animate.css/>

<sup>123</sup><http://velocityjs.org/>

<sup>124</sup><https://kr.vuejs.org/v2/guide/transitions.html>

```
src/App.vue
<template>
  <div>
    ...
    <transition name= " fade ">
      <router-view></router-view>
    </transition>
  </div>
</template>

<style type= " text/css " >
  .fade-enter{
    opacity: 0
  }
  .fade-enter-active {
    transition: opacity 1s
  }
  .fade-enter-to {
    opacity: 0.8
  }
</style>
```

CSS클래스를 살펴보세요. 트랜지션은 1초 동안 점차 증가하는 불투명도 0으로 시작합니다. .fade-enter-to는 필요하지 않습니다. 이렇게 정의하면 트랜지션이 끝나기 전에 0.8에서 1로 반짝임이 생깁니다.

컴포넌트가 진출 트랜지션에 들어갔을 때 역방향 애니메이션을 만드려면 다음과 같이 CSS를 변경합니다.

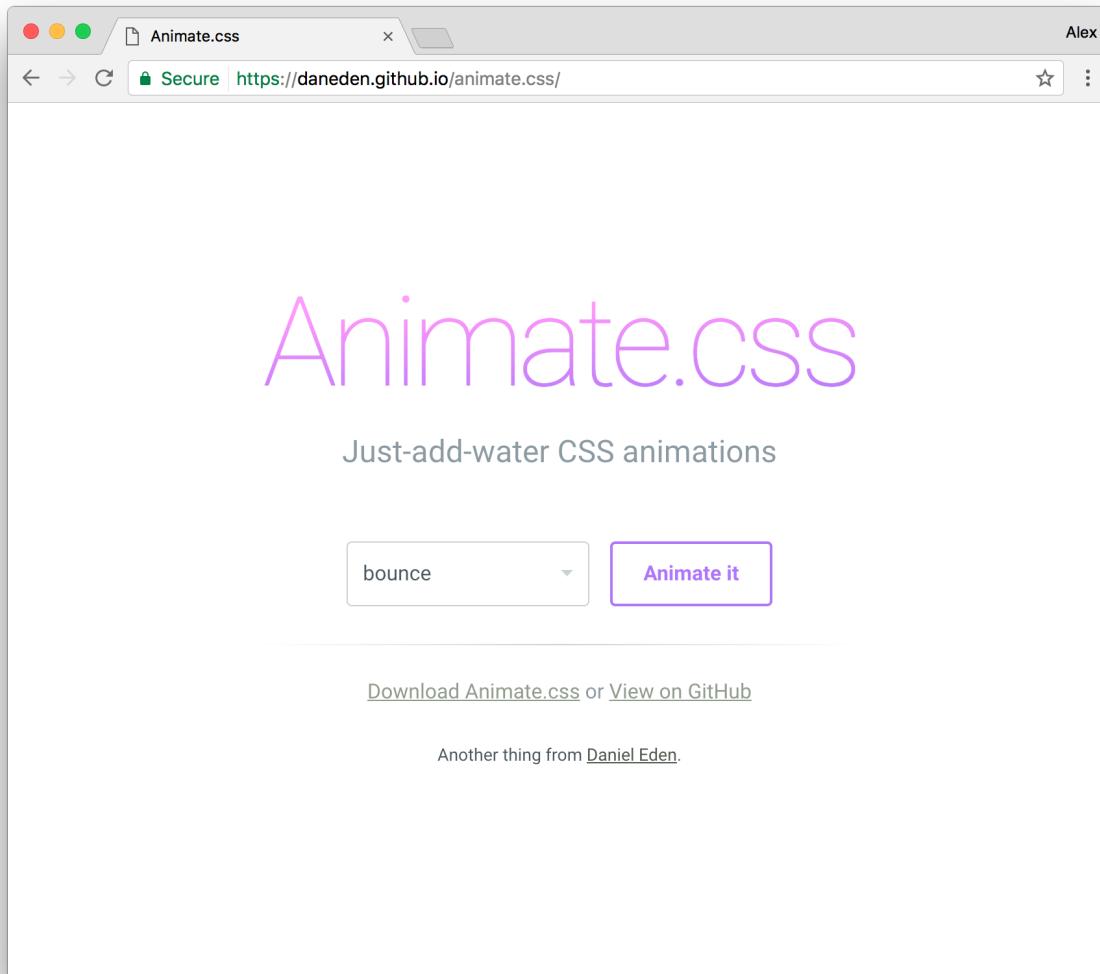
```
<style type= " text/css " >
  .fade-enter, .fade-leave-to{
    opacity: 0
  }
  .fade-enter-active, .fade-leave-active {
    transition: opacity 1s
  }
  .fade-enter-to, .fade-leave {
    opacity: 0.8
  }
</style>
```

## 써드파티 CSS 애니메이션

밑바닥부터 애니메이션을 설계하고 만드는 것은 매우 어렵습니다. 그래서 써드파티 라이브러리를 사용합니다. 운좋게도, Vue의 트랜지션은 써드파티 라이브러리와 통합하는 것이 매우

쉽습니다.

예제에서는 [Animate.css<sup>125</sup>](#)를 사용합니다.



### Animate.css

Animate.css를 사용하려면 아래의 작업을 해야합니다.

1. 문서의 <head>에 추가합니다.
2. 애니메이션 하려는 엘리먼트에 `animated` 클래스를 추가합니다. 무한 루프를 위해 `infinite` 클래스를 추가할 수 있습니다.

<sup>125</sup><https://daneden.github.io/animate.css/>

- 마지막으로 `bounce`, `rollIn`, `fadeIn` 등과 같이 사용가능한 클래스 중 하나를 추가해야 합니다. 사용 가능한 모든 클래스 리스트는 [여기<sup>126</sup>](#)에 있습니다.

`index.html` 파일의 헤더에 CDNJS에서 가져온 파일을 추가하여 시작할 수 있습니다.

이번에는 `transition` 컴포넌트의 `name` 속성에 의존하는 대신 `v-enter-active`의 커스텀 클래스를 사용할 것입니다.

src/App.vue

---

```
<template>
  <div>
    ...
    <transition enter-active-class= "animated rollIn " >
      <router-view></router-view>
    </transition>
  </div>
</template>
```

---

또한 `leave-active-class= "animated rollOut "`와 같이 커스텀 `leave-active-class`를 추가하여 컴포넌트가 DOM에서 제거될 때 애니메이션을 적용할 수 있습니다.

## 네비게이션 가드

`vue-router`은 라우트 변경을 필터링하기 위한 편리한 메커니즘을 제공합니다. 각 변경 전에 트리거되는 `router.beforeEach()` 및 변경 후 트리거되는 `router.afterEach()`를 사용할 수 있습니다. 여기서 `afterEach()`는 탐색에 영향을 미치지 않습니다.

`router.beforeEach()`는 인증과 관련된 시나리오에서 유용하게 사용할 수 있습니다. 예를 들어 사용자가 앱 페이지에서 액세스 권한이 없으면 로그인 페이지로 이동해야 합니다.

간단한 예로 확인해보겠습니다.

---

<sup>126</sup> <https://github.com/daneden/animate.css>

src/main.js

---

```
1 // 더미 사용자 객체 생성
2 let User = {
3   isAdmin: false
4 }
5
6 router.beforeEach((to, from, next) => {
7   if (to.path !== '/login' && !User.isAdmin) {
8     // 관리자가 아니면 login 페이지로 리다이렉트
9     next('/login')
10    } else {
11      // 인증된 사용자는 계속 진행합니다
12      next()
13    }
14 })
```

---

다음은 규칙을 적용하여 라우터가 login을 제외한 다른 페이지로 이동하지 않도록 합니다.  
항상 `next()`함수를 호출해야 합니다. 그렇지 않으면 흑이 절대 해결되지 않습니다.



## 예제 코드

이 장의 예제는 [GitHub<sup>127</sup>](#)에서 확인하실 수 있습니다.

---

<sup>127</sup><https://github.com/hoottex/the-majesty-of-vuejs-2/tree/master/codes/chapter19>

## 혼자해보기

이 장에서 우리는 많은 것들을 살펴보았습니다.  
 이번에는 미니 포켓몬 도감을 만들어야합니다.  
 홈 페이지에는 불, 물 등의 포켓몬 카테고리 목록을 표시합니다.  
 카테고리를 탐색하여 해당 포켓몬들 보고 새 포켓몬을 추가할 수 있어야합니다.  
 라우트는 다음과 같습니다.

라우트	상세
/	카테고리 목록.
/category/:name	해당 카테고리의 포켓몬 목록.
/category/:name/pokemons/new	카테고리에 새 포켓몬 추가.

각 변경마다 콘솔에 출력하세요 예를들어 Fire 카테고리 탐색을 시작하면 사용자가 /category/Fire를 방문할 것을 알리는 메시지를 출력해야합니다.

시작하기 쉽게 포켓몬 도감 객체를 만들어두었습니다. [여기<sup>128</sup>](#)에서 확인하세요.



### 정보

`/category/:name/pokemons/new` 라우트는 `/category/:name`의 서브 라우트입니다.  
 사용자가 `/category/:name/pokemons/new`를 방문하면 카테고리의 포켓몬 목록과 함께 새로운 포켓몬을 추가하는 품을 출력합니다.



### 힌트 1

특정 카테고리의 포켓몬 도감에 액세스하려면 JavaScript의 `find` 메소드<sup>129</sup>를 사용하세요

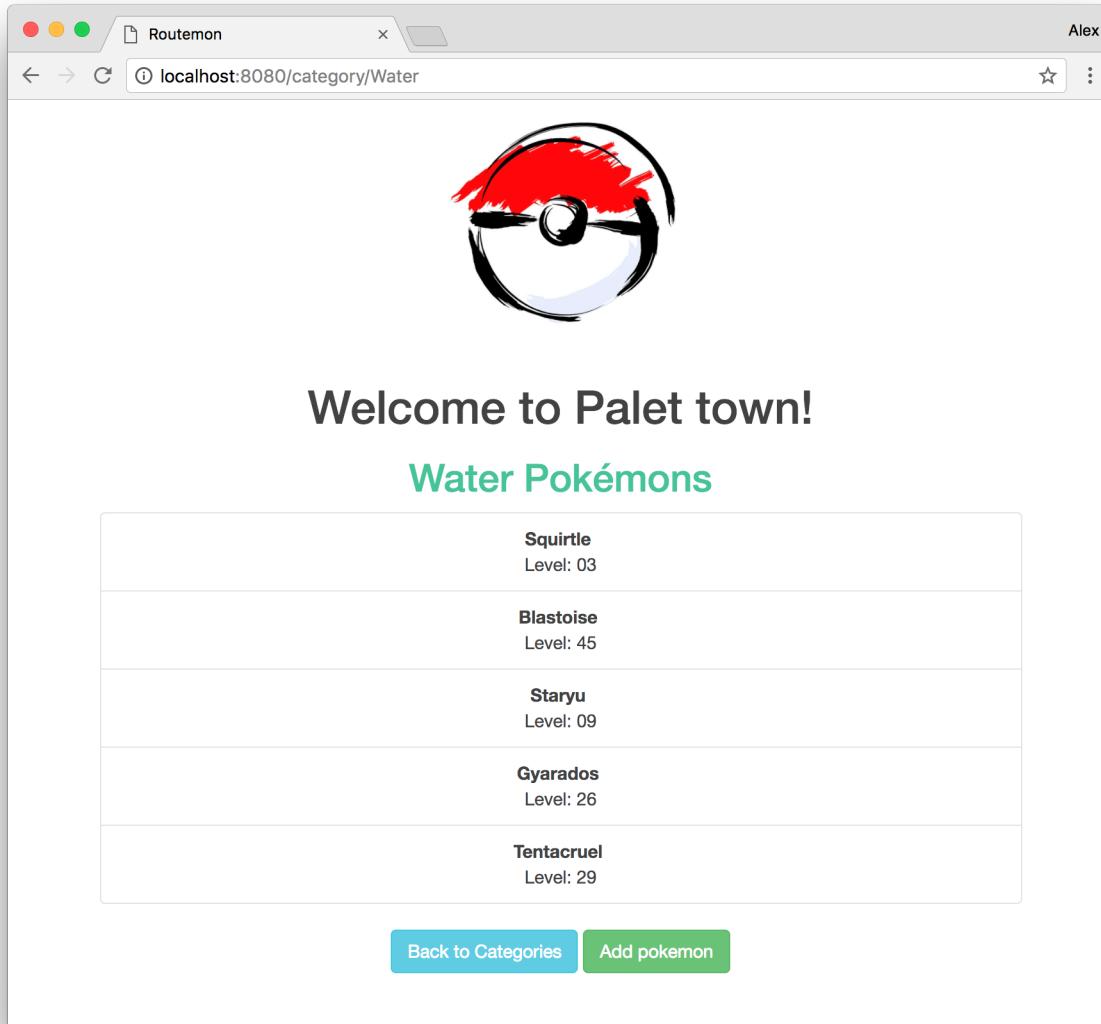


### 힌트 2

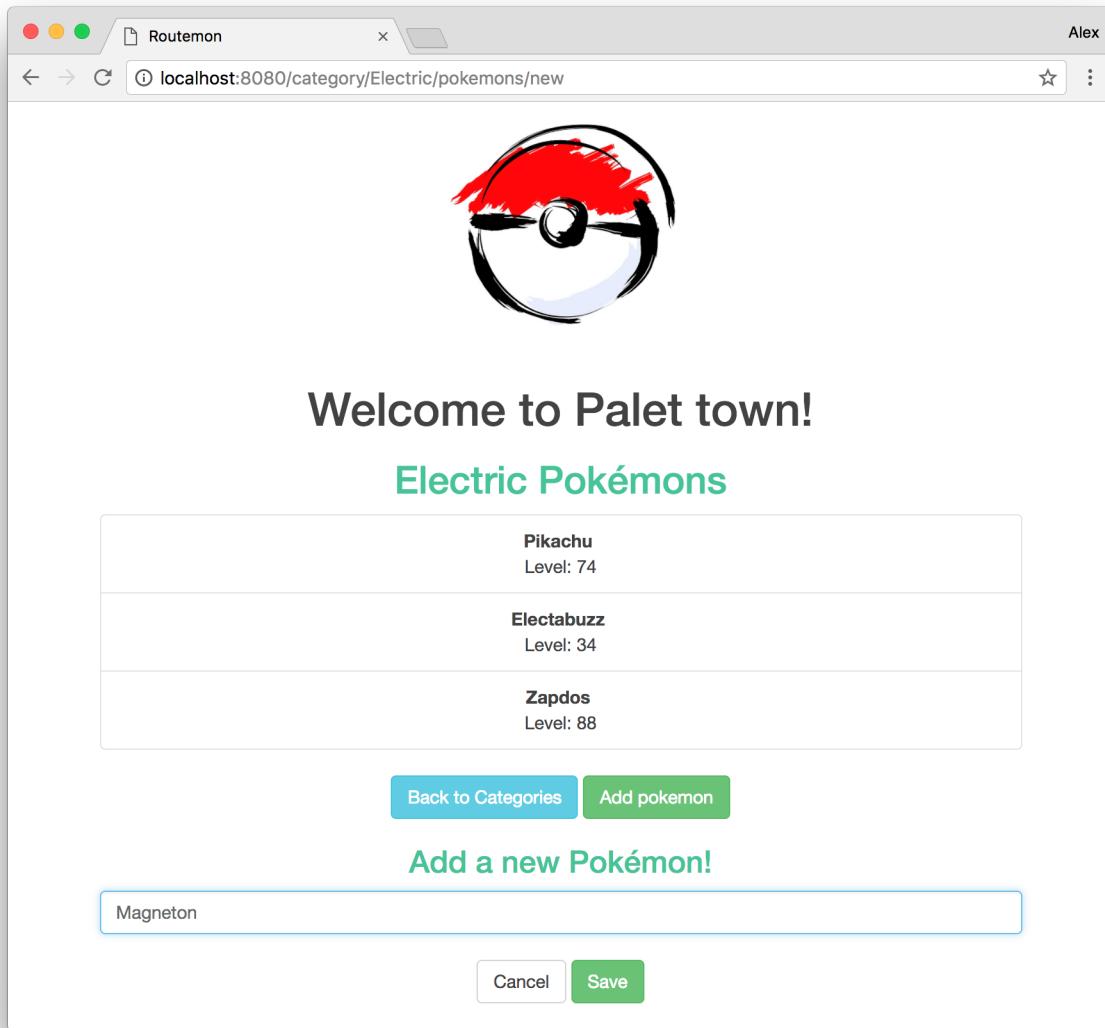
각 변경 전에 콘솔에 메시지를 출력하려면 `router.beforeEach()`를 사용하세요.

<sup>128</sup> <https://github.com/hoottex/the-majesty-of-vuejs-2/blob/master/homework/Chapter19/pokedex.js>

<sup>129</sup> [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/find](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find)



결과 예



결과 예

이상의 예제 코드는 [Github<sup>130</sup>](#)에 있습니다.

<sup>130</sup><https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter19>

# 마무리하며

자, 이제 긴 여행이 끝났습니다. 책을 읽는 동안 즐거웠기를 바랍니다. 우리의 유머에 조금이나마 웃었기를 바랍니다.

읽어준 모두에게 고맙습니다. 그리고 이 책을 실제로 만들기 위해 도와준 분들 또한 고맙습니다. 문법, 구문 및 코드에 시간을 할애한 [기여자<sup>131</sup>](#)들에게 큰 감사를 전합니다. 우리는 [Vue.js Feed<sup>132</sup>](#)를 출시했습니다. 꼭 확인해보세요. Vue.js와 관련된 뉴스, 자습서, 코드 및 플러그인이 매일 올라옵니다. 여러분의 작업물이 Vue.js Feed에 올라오기를 희망합니다. 정말 기쁠 것입니다 :)



## 낯선 사람이 되지 마세요!

무엇이든 물어보거나 단지 인사만이라도 하고 싶다면 [Twitter<sup>133</sup>](#)를 이용해 연락해주세요. 책의 업데이트를 꾸준히 제공하고 있습니다.

---

<sup>131</sup><https://github.com/hootlex/the-majesty-of-vuejs-2/graphs/contributors>

<sup>132</sup><https://vuejsfeed.com/>

<sup>133</sup><http://twitter.com/tmvuejs>

# 더 배우려면

우리는 읽어볼만한 튜토리얼과 책을 나열할 것입니다. 또한 Vue를 사용한 실제 사례(오픈소스)를 모았습니다. 일부는 직접적으로 Vue.js와 연관성이 없을 수 있지만 어느정도 연관된 주제를 가지고 있습니다.

더 고급 주제를 배우고 실제 애플리케이션을 작성하는 방법을 배우려면 Alex Kyriakidis의 [Vue School<sup>134</sup>](#)을 확인해보세요.

## 튜토리얼

- [Vue.js 소개 \(연재\)<sup>135</sup>](#) – Vue의 많은 부분을 다루는 연재글입니다. 렌더링과 디렉티브, 이벤트, 컴포넌트, Vuex와 애니메이션 등을 다룹니다.
- [Firebase를 사용하는 Vue.js 애플리케이션 인증<sup>136</sup>](#) – Firebase를 이용한 매우 간단한 앱을 만듭니다. Vue.js와 함께 사용하기 매우 쉽습니다.
- [네이티브 모바일 앱을 위한 Vue와 Weex<sup>137</sup>](#) – 이 안내서는 Vue의 핵심 컨셉을 이용한 Weex 플랫폼을 다룹니다. Vue를 이용해 네이티브 모바일 앱을 만들 수 있습니다!
- [어떻게 반응형 JavaScript 엔진을 만드는가<sup>138</sup>](#) – 이 안내서는 읍저버의 getters/setters를 이용해 실시간으로 변경사항을 알아채는 방법을 배웁니다.
- [Vue.js 컴포넌트를 오픈소스로 작성하는 방법<sup>139</sup>](#) – 직접 Vue.js를 이용해 컴포넌트를 만들고 배포하는 내용을 다룹니다.

## 비디오

- [프론트엔드 프레임워크 성능에 관하여<sup>140</sup>](#) – Evan은 이 강연에서 주요 프론트엔드 프레임워크에서 사용하는 더티체킹, 가상 DOM 차이점 분석 및 의존성 추적을 이야기합니다.
- [프론트엔드 자바스크립트 프레임워크의 반응형<sup>141</sup>](#) – 프레임워크는 상태변경을 어떻게 감지하고 시스템을 통해 변경사항을 효율적으로 전파할까요? Evan은 Vue.js를 만든 경험을 토대로 이 질문에 답변합니다.

---

<sup>134</sup> <https://vueschool.io>

<sup>135</sup> <https://css-tricks.com/intro-to-vue-1-rendering-directives-events/>

<sup>136</sup> <https://medium.com/dailyjs/authenticating-a-vue-js-application-with-firebase-ui-8870a3a5cff8>

<sup>137</sup> <https://code.tutsplus.com/tutorials/introducing-vue-and-weex-for-native-mobile-apps--cms-28782>

<sup>138</sup> <http://monterail.com/blog/2016/how-to-build-a-reactive-engine-in-javascript-part-1-observable-objects/>

<sup>139</sup> <http://monterail.com/blog/2016/simple-guide-to-authoring-open-source-vue-js-components/>

<sup>140</sup> <https://vuejsfeed.com/blog/demystifying-frontend-framework-performance-video>

<sup>141</sup> <https://vuejsfeed.com/blog/demystifying-frontend-framework-performance-video>

- Learning Vue 2: Step By Step (연재)<sup>142</sup> – Vue를 단계별로 배울 수 있습니다.

## 책

- ECMAScript 6 이해하기<sup>143</sup> – ES6에서 이해해야하고 배워야할 많은 새로운 개념을 소개합니다.
- Build APIs You Won't Hate<sup>144</sup> – 요즘은 백엔드와 프론트엔드를 분리해서 개발하는 것이 거의 표준입니다. 프론트엔드는 Vue.js와 같은 멋진 프레임워크를 사용하고 백엔드는 일반적으로 JSON을 사용하는 서버측 언어를 이용합니다.
- SVG Animations<sup>145</sup> – SVG는 HTTP 요청을 줄일 수 있고 모든 화면에서 높은 선명도를 제공합니다. 이 책에서 SVG에 대한 모든 것을 배울 수 있습니다. 예를 들어 SVG 크로스 브라우저 호환, 하위호환성, 최적화 및 반응성을 만드는 방법을 배웁니다.

## 오픈소스 프로젝트

- Vuedo<sup>146</sup> – Laravel과 Vue.js로 만든 블로그 플랫폼입니다.
- Airflix<sup>147</sup> – AirPlay 친화적인 웹 인터페이스입니다. 홈서버에서 영화와 TV 프로그램을 스트리밍할 수 있습니다.
- Koel<sup>148</sup> – 잘 작동하는 개인 음악 스트리밍 서버입니다.
- Mini e-shop<sup>149</sup> – Vue.js로 만든 작은 온라인 쇼핑몰입니다. 많은 기능들과 함께 시작할 수 있습니다!
- The Movie Database App<sup>150</sup> – imdb와 같은 예쁜 애플리케이션입니다.
- Vue Wordpress PWA<sup>151</sup> – Vue.js로 만든 오프라인 우선 SPA입니다. WordPress API를 이용하는 프로그래시브 웹앱입니다.
- Hypersurface<sup>152</sup> – Hypersurface는 사람들의 개인적으로 묻고 답하고 이야기하는 곳입니다. 데이터는 실시간으로 반영되므로 누구나 현재 시점에서 참여할 수 있습니다.

<sup>142</sup><https://laracasts.com/series/learn-vue-2-step-by-step>

<sup>143</sup><https://leanpub.com/understandinges6>

<sup>144</sup><https://leanpub.com/build-apis-you-wont-hate>

<sup>145</sup><http://shop.oreilly.com/product/0636920045335.do>

<sup>146</sup><https://github.com/Vuedo/vuedo>

<sup>147</sup><https://github.com/wells/airflix>

<sup>148</sup><https://github.com/phanan/koel>

<sup>149</sup><https://github.com/BosNaufal/vue-mini-shop>

<sup>150</sup><https://github.com/dmtrbrl/tmdb-app>

<sup>151</sup><https://github.com/bstavroulakis/vue-wordpress-pwa>

<sup>152</sup><https://github.com/aswdesign/hypersurface>

## Awesome Vue

Awesome Vue.js<sup>153</sup>는 Vue.js와 관련된 멋진 것들을 모아놓은 곳입니다. 수많은 Vue 작품들을 찾을 수 있을 것입니다. 현재 Vue팀은 새로운 프로젝트인 Curated Vue<sup>154</sup>를 만들고 있습니다. Curreated Vue에서는 허가된 리소스만 포함하며 검색을 제공합니다.



The End...

<sup>153</sup><https://github.com/vuejs/awesome-vue>

<sup>154</sup><https://curated.vuejs.org/>