

Vue.js

김 순곤

soongon@hucloud.co.kr

프론트엔드 개발

- ☑ 프론트엔드 개발의 기술 발전..
 - 라이브러리 등장 (개발을 도와주고 쉽게 해주는 기술)
 - prototype.js -> jQuery(브라우저호환성)
- ☑ 2009년 웹 표준이 등장
 - 프론트엔드 개발의 혁신이 일어난다!
 - 컴포넌트 기반의 개발방식 도입
 - 컴포넌트 기반 개발 방식의 라이브러리
 - react.js, Vue, Svelte, Angular

Vue.js 목차

1. Vue.js 와 프론트엔드 개발
2. Vue CLI (Vite) 와 Scaffolding
3. Vue 기본 문법
4. Vue 컴포넌트
5. HTTP 요청을 통해 웹(백엔드)에서 데이터 가져오기 (API)
6. 상태관리 (pinia)
7. Vue Framework 와 다른 기술과의 비교

Vue.js 시작하기

Vue.js란?

The **Progressive** JavaScript Framework

An approachable, performant and versatile framework for building web user interfaces.



Why Vue

Get Started →

Install

Vue.js란?

2014년 Evan You 에 의해 개발

2016년 Version 2

2020년 Version 3 - Composition API 등장

Approachable

Builds on top of standard HTML, CSS and JavaScript with intuitive API and world-class documentation.

Performant

Truly reactive, compiler-optimized rendering system that rarely requires manual optimization.

Versatile

A rich, incrementally adoptable ecosystem that scales between a library and a full-featured framework.

프론트 엔드 개발

☑ 프론트엔드 개발의 다양성

- 웹 프론트엔드 (웹브라우저 기반, HTML/CSS/JS, 웹 표준 기반(2009))
- 앱 - 모바일 앱(안드로이드, 아이폰 앱)
- 하이브리드 앱 (하나 만들어서 -> 웹앱, 안드로이드앱, 아이폰앱 빌드)
 - react native, Flutter, Xamarin(MS)
- Single-Page Application (SPA)
- Fullstack / Server Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- 데스크탑, 모바일, WebGL, 기타 터미널

프론트 엔드 개발

- ☑ 프론트엔드 개발의 중요성
 - 백엔드 아키텍처에 미치는 영향 최소화
 - 프론트엔드 개발 문제를 신속하게 해결
 - UI의 중요성 - 사용자 경험, 디자인
 - 끊임없이 진화하는 제품(서비스) 요구 사항
 - Apple, Google → 프론트엔드 중요성 입증

Angular vs. Vue

☑ Angular

- From Google
- MVVM 프레임워크
- AOT 렌더링, 라우터, CLI
- 기본적으로 TypeScript 채택
- 디펜던시 인젝션 기능
- 글로벌 상태관리 시스템 부재 -> Flux, NgRx 채택 필요

☑ React, Vue

- 컴포넌트 기반 개발에 중점을 둠 -> 프레임워크 도입에 노력과 시간을 줄여줌

React vs. Vue

- ☑ 2013년 Facebook 에서 오픈 릴리즈됨
 - JSX 패턴 (Javascript 에서 HTML과 CSS 를 작성하는 방법)
 - 모듈 방식의 애플리케이션 구축
- ☑ Vue
 - 모듈식 코딩의 핵심을 취함
 - JavaScript 단일 파일에서 HTML, CSS 요소를 포함 (JSX 와 유사)
 - 모듈식 개발 방식의 단순화

프로젝트에 Vue 를 사용할 때의 이점

- ☑️ 프론트엔드 개발은 배우기 쉽지만 마스터하기 어려운 패턴
 - 신규 및 베테랑 개발자 모두에게 사용 가능
- ☑️ 개발자는 최적화 되고 성능이 뛰어난 프레임 워크를 사용하여 모든 크기의 동적 프론트엔드 애플리케이션을 구축 가능
- ☑️ 단일 파일 구성 요소 (SFC) 패턴 사용
 - 개발 과정을 단순화
- ☑️ Pinia - 유연한 글로벌 상태 패턴 을 생성

Vue 단일 페이지 패턴 (SFC)

- ☑ SFC (Single File Component) 패턴
 - 작업을 더 작은 청크 단위로 분할
 - DRY(Don't Repeat Yourself)

```
<template>
  <div>
    <!-- Write HTML syntax here -->
  </div>
</template>

<script>
  export default {
    // Write javascript here
  }
</script>

<style>
  /* Write styling here */
</style>
```

개발환경 설정

create-vue 와 스캐폴딩

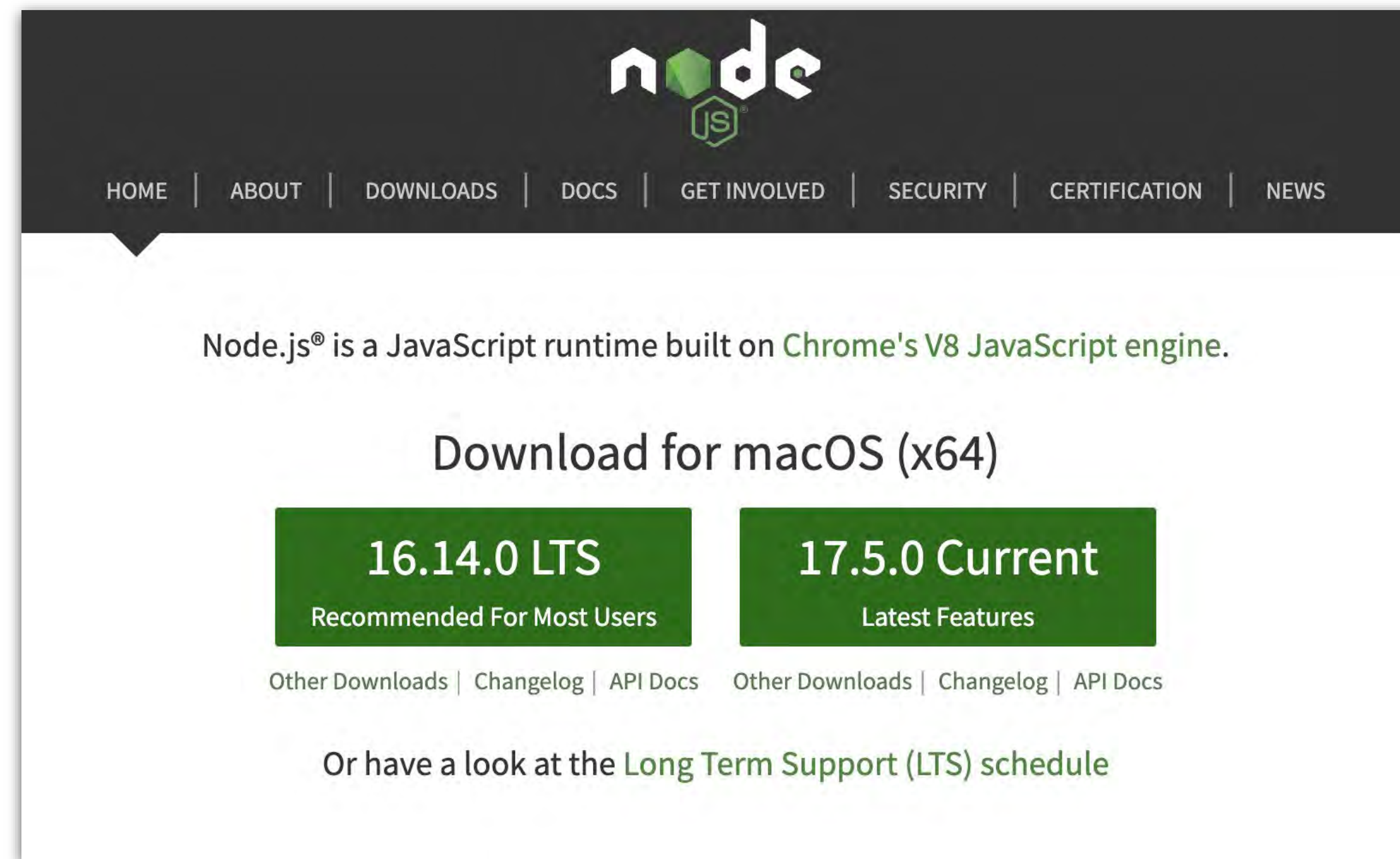
Quick Start

노드 설치


- ☑ Node.js 와 npm 설치 : <http://nodejs.org>
 - Node 설치 시 NPM 기본으로 같이 설치 됨

- ☑ 설치 확인
 - `node --version`
 - `npm --version`


위 두 가지
터미널에서 설치 확인




VS Code - <https://code.visualstudio.com/>

 Visual Studio Code

[Docs](#) [Updates](#) [Blog](#) [API](#) [Extensions](#) [FAQ](#) [Learn](#)



 [Download](#)

Version 1.64 is now available! Read about the new features and fixes from January.


Code editing. Redefined.


Free. Built on open source. Runs everywhere.


[Download Mac Universal](#)
Stable Build


[Other platforms and Insiders Edition](#)


By using VS Code, you agree to its [license and privacy statement](#).


**Python** 2019.6.24221 54.9M 4.5
Linting, Debugging (multi-threaded...
Microsoft [Install](#)


**GitLens — Git su...** 9.8.5 23.1M 5
Supercharge the Git capabilities bui...
Eric Amodio [Install](#)


**C/C++** 0.24.0 23M 3.5
C/C++ IntelliSense, debugging, and...
Microsoft [Install](#)

**ESLint** 1.9.0 21.9M 4.5
Integrates ESLint JavaScript into V...
Dirk Baeumer [Install](#)

**Debugger for C...** 4.11.6 20.6M 4
Debug your JavaScript code in the ...
Microsoft [Install](#)

**Language Sup...** 0.47.0 18.6M 4.5
Java Linting, Intellisense, formattin...
Red Hat [Install](#)

**vscode-icons** 8.8.0 17.2M 5
Icons for Visual Studio Code
VSCode Icons Team [Install](#)

**Vetur** 0.21.1 17M 4.5
Vue tooling for VS Code
Pine Wu [Install](#)

blog-post.js **index.js** **utils.js**

```
src > components > JS blog-post.js > <function> > [e]blogPost
1 import React from "react"
2 import Image from "gatsby-image"
3
4 export default ({ data }) => {
5   const blogPost = data.cms.blogPost
6   return (
7     <div>
8       {blogPost} {data} {dateFormat}
9       {blogPost} {debug}
10      {blogPost} {debugger}
11      {blogPost} {decodeURI}
12      <Image> {decodeURIComponent}
13    )
14    <h1>{blogPost} {defaultStatus}
15    <div>Post {delete}
16    <div> {dang} {departFocus}
17    </div> {devicePixelRatio}
18    {dispatchEvent}
19  )
20
21  export const query = graphql`
```

PROBLEMS

TERMINAL

2: Task - develop

+

□

✖

^

×

```
Info [wdim]: Compiling...
DONE Compiled successfully in 26ms 3:57:58 PM
Info [wdim]:
Info [wdim]: Compiled successfully.
```

master

0 ↓ 1 ↑

0 0

1

Gatsby Develop (gatsby-graphq-app)

Ln 6, Col 21

Spaces: 2

UTF-8

LF

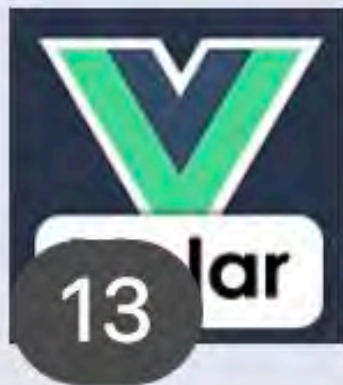
JavaScript

🔔

 IntelliSense Run and Debug Built-in Git Extensions

VsCode 익스텐션 - Volar

- ☑ Vue Volar extension Pack
 - VSCode 와 Volar 의 조합으로 사용
 - Vue2 에서 사용하던 Vetur는 Vue3.0에서 삭제가 필요



Vue Volar extension Pack

This extension packs uses volar for vue projects.

BroJenuel



create-vue 와 스캐폴딩

vite 와 스캐폴딩

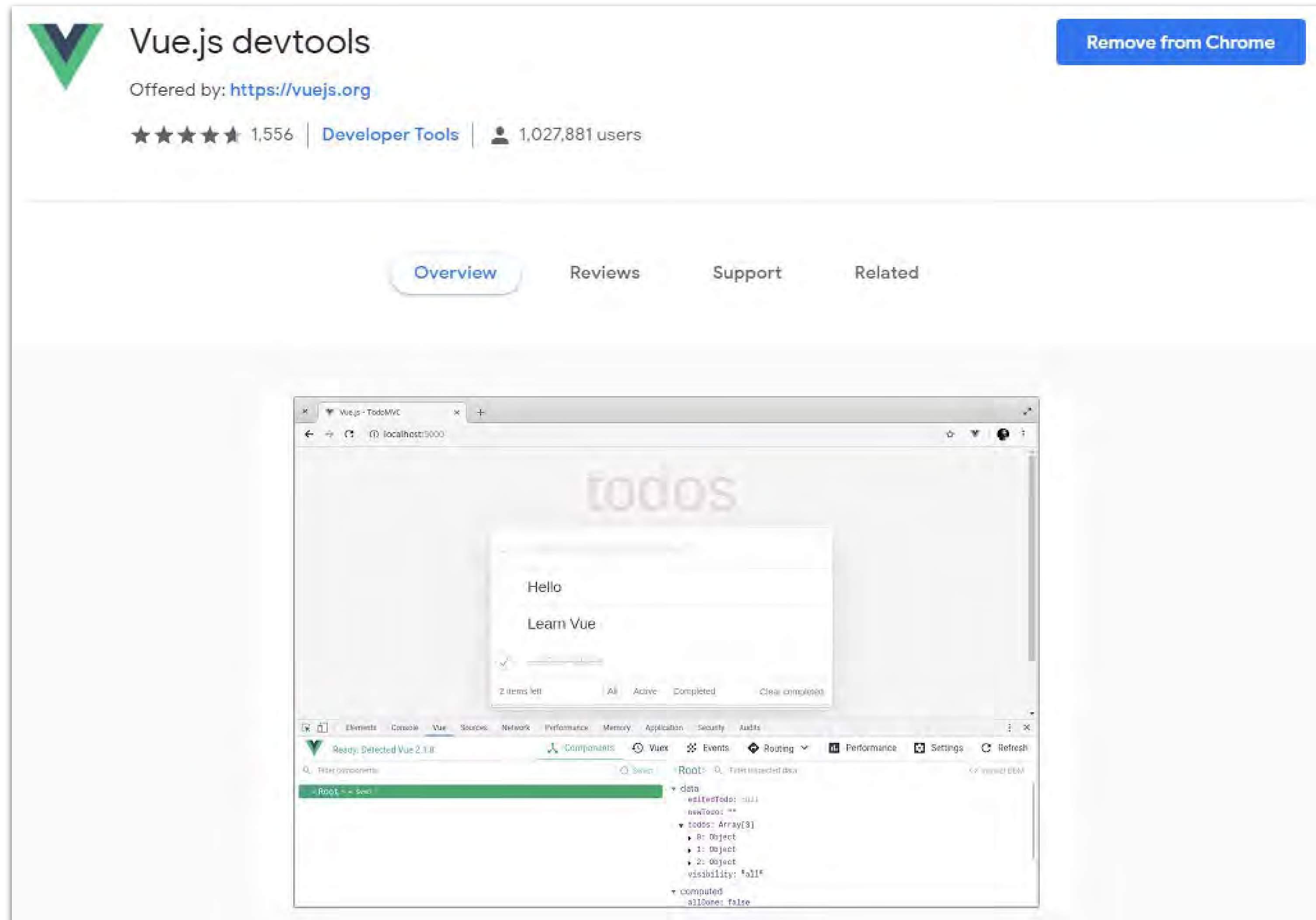
☑ vite 사용 프로젝트 생성

```
> npm init vue@latest
```

- create-vue 설치와 실행
- 공식 vue 스캐폴딩 툴

```
> cd <your-project-name>  
> npm install  
> npm run dev
```

크롬 확장프로그램 - Vue.js DevTools



데이터(States) 와 데이터 바인딩

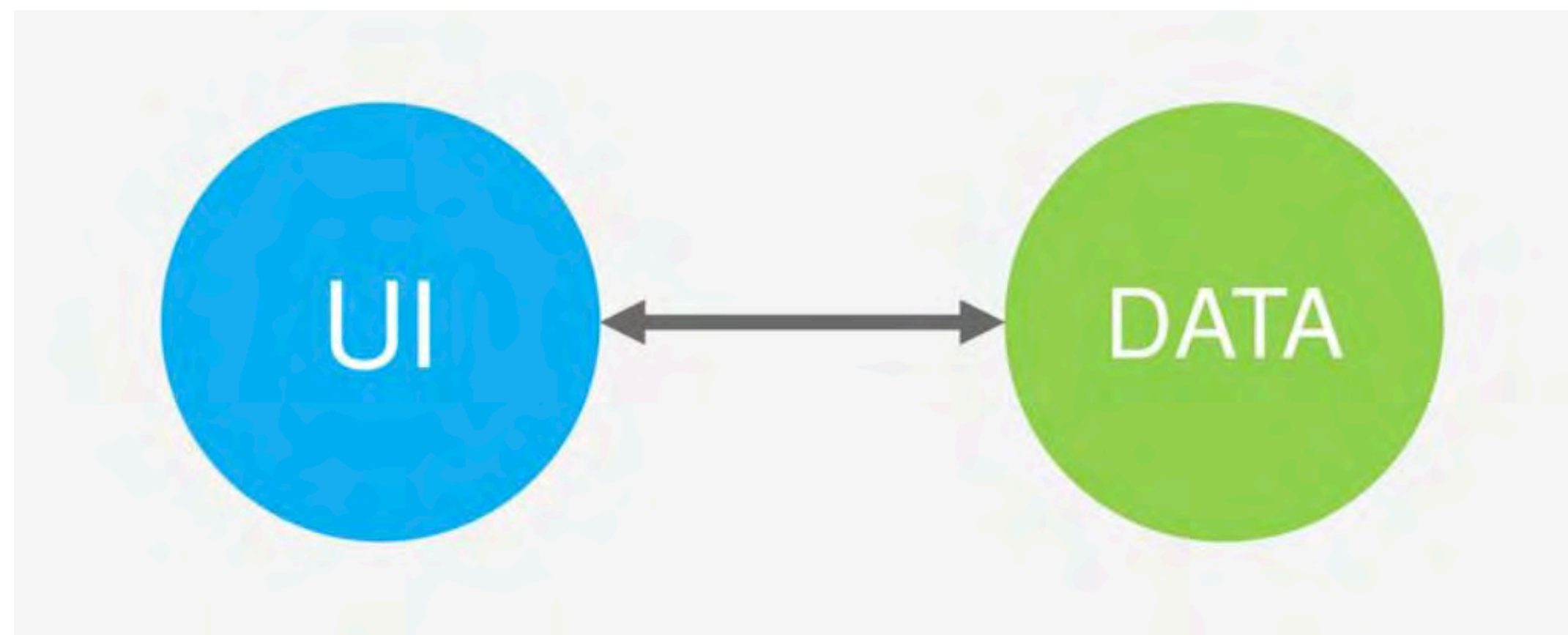
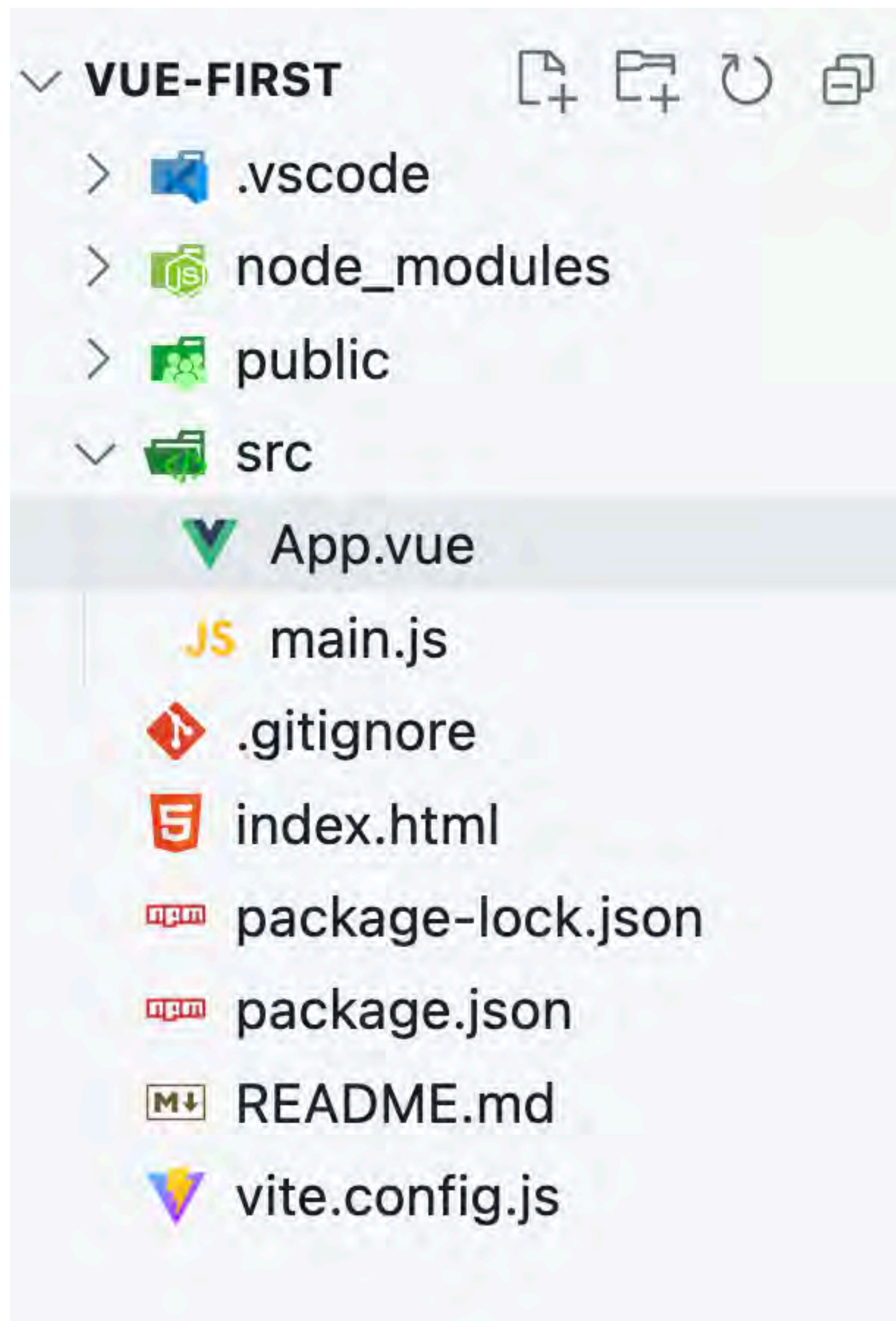
어트리뷰트 바인딩

디렉티브

뷰 함수

이벤트 바인딩

필수 문법



데이터 (States) 와 데이터 바인딩

- ☑ 화면은 UI 와 데이터로 구성됨
 - States 를 통해 Vue 템플릿 내에서 사용하는 모든 정보를 저장 가능
 - 이 데이터가 변경되면 해당 템플릿이 자동으로(반응적으로) 업데이트됨
 - Vue 는 데이터(state)를 리액티브(reactively)하게 관리

```
<template>
  <div>{{color}}</div>
</template>

<script setup>
import {ref} from 'vue';

const color = ref('red');
</script>
```


데이터 바인딩

☑ 데이터 바인딩, Interpolation

- {{ }} : 일명 mustache 표기법 - 모든 자바스크립트 표현식 사용 가능

```
<template>
  <div>
    <h1>{{ isUppercase ? title.toUpperCase() : title }}</h1>
  </div>
</template>
<script setup>
  import ref from 'vue';
  const isUppercase = ref(false);
  const title = ref('hello');
</script>
```

```
{{ number + 1 }}
```

```
{{ ok ? 'YES' : 'NO' }}
```

```
{{ message.split('').reverse().join('') }}
```

- ☑ v-bind : 자바스크립트 표현식으로 HTML 속성을 바인딩

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

- ☑ Boolean Attribute

```
<button :disabled="isButtonDisabled">Button</button>
```

- ☑ 단축표현

```
<div :id="dynamicId"></div>
```

- ☑ v-text: {{ }} 대신 사용
- ☑ v-once: 컴포넌트를 초기에 한번만 렌더링(static)
- ☑ v-html: 태그를 포함해서 렌더링 가능
- ☑ v-bind(:src): 이미지의 URL 을 바인딩 할 때
- ☑ v-if: 조건문 사용시
- ☑ v-show: 해당 태그(엘레먼트)가 보여질지 여부를 true/false 설정
- ☑ v-for: 반복문 사용시

Directives

- ☑ 기본 디렉티브, v-text, v-once, v-html, v-bind, v-if, v-show

```
<template>
  <div>
    <h1 v-show="true" v-once v-text="text">Loading...</h1>
    <h2 v-show="false" v-html="html" />
    <a
      :href="link.url"
      :target="link.target"
      :tabindex="link.tabindex"
      v-text="link.title"
    />
  </div>
</template>
```

Vue 함수

☑ 함수 호출

```
<template>
  <div>
    <h1>Triggering Vue Methods</h1>
    <ul>
      <li v-for="n in 5" :key="n">
        <a href="#" @click="triggerAlert(n)">Trigger {{ n }}</a>
      </li>
    </ul>
  </div>
</template>
```

Triggering Vue Methods

Trigger 1

Trigger 2

Trigger 3

Trigger 4

Trigger 5

```
<script setup>
function triggerAlert(n) {
  alert(`${n} has been clicked`)
}
</script>
```

함수 작성과 이벤트 바인딩

- ❑ 자바스크립트로 작성되어짐
- ❑ 해당 Vue 컴포넌트 내에서만 호출 및 동작 가능
- ❑ 주로 이벤트 바인딩시 메소드가 호출됨
- ❑ 이벤트 바인딩은 `v-on:click` 또는 `@click` 으로 표현

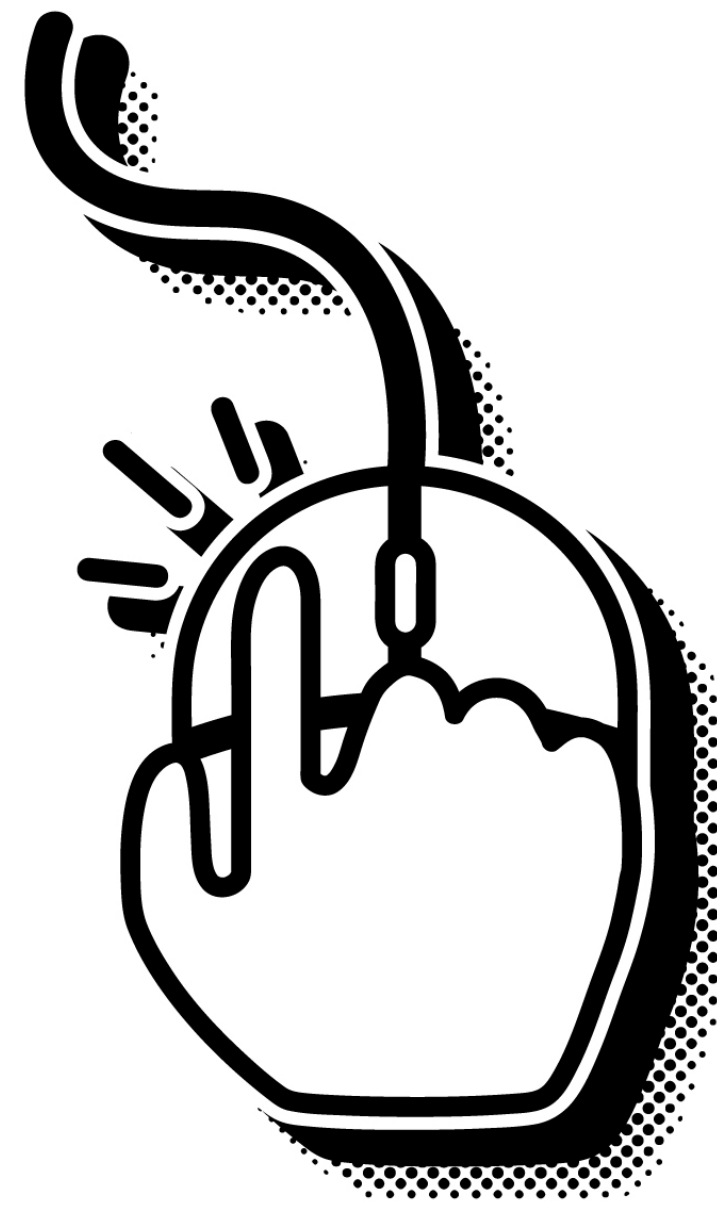
```
<a v-on:click="doSomething"> ... </a>
```

```
<!-- shorthand -->
```

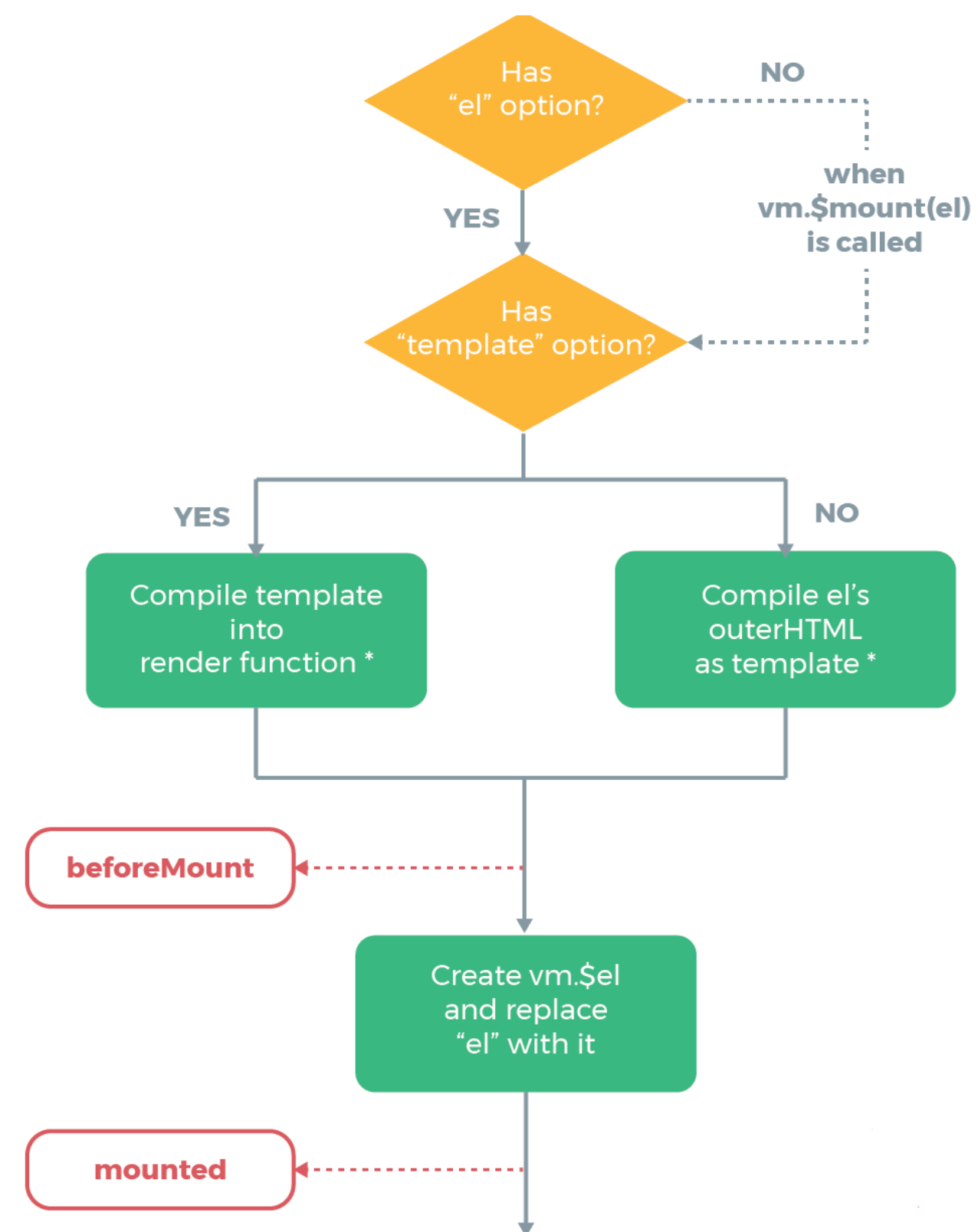
```
<a @click="doSomething"> ... </a>
```

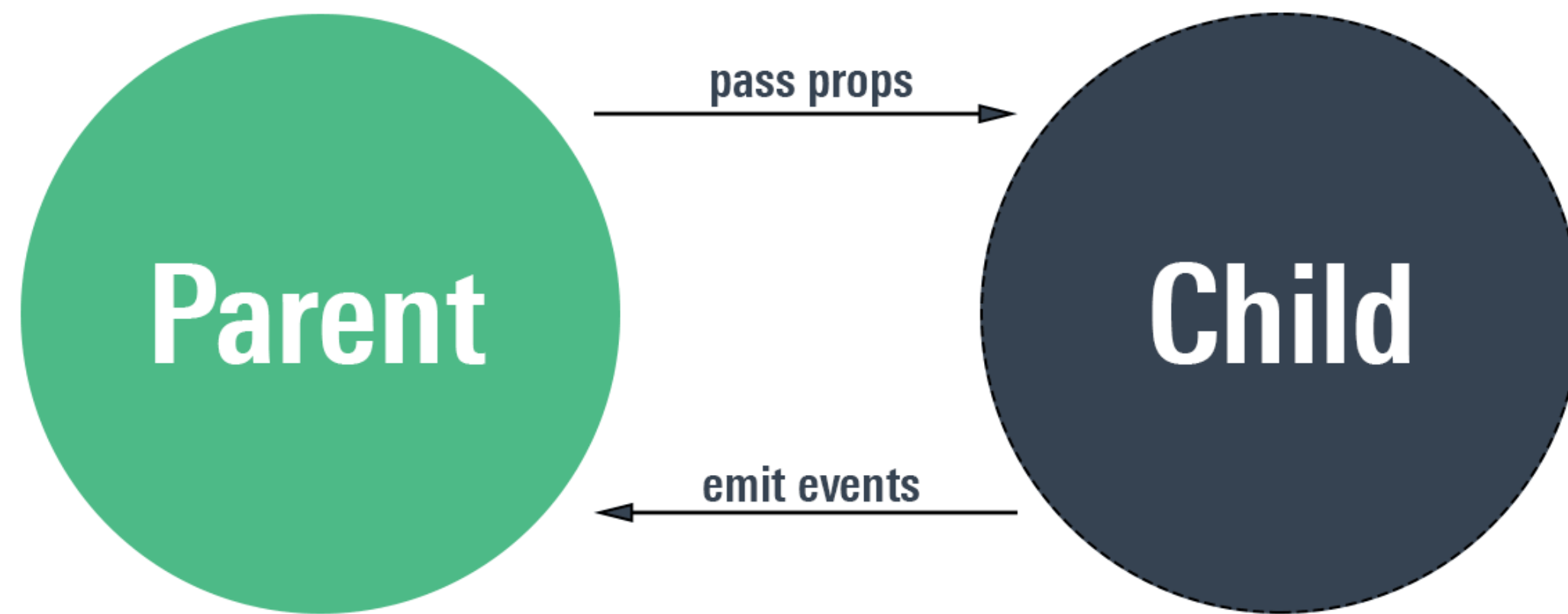
Reactive

리액티브 데이터



- banana
- strawberry
- apple
- melon

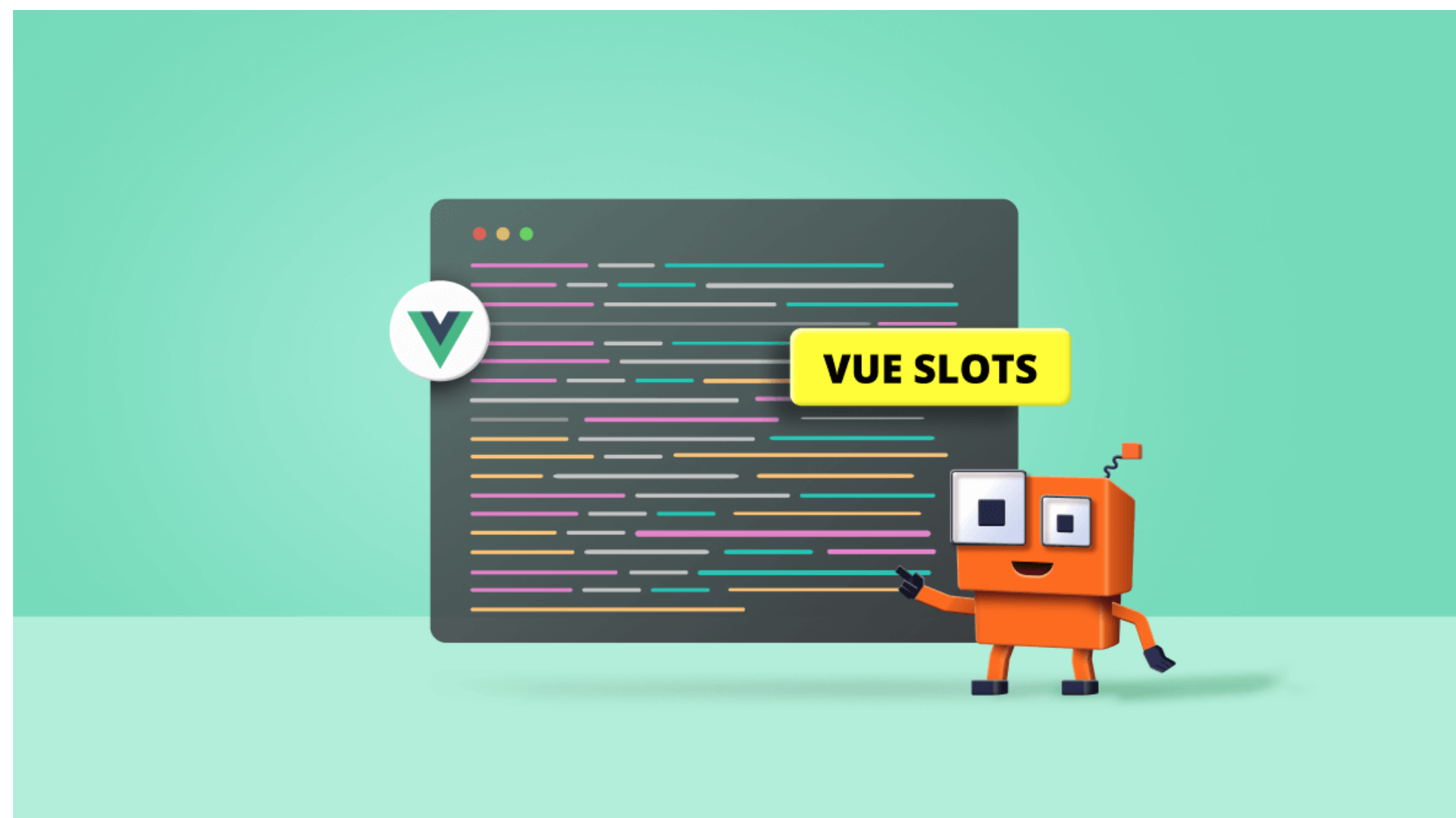





Vue Router

The official Router for Vue.js

Expressive, configurable and convenient routing for Vue.js



Pinia

The intuitive store for Vue.js

Type Safe, Extensible, and Modular by design.
Forget you are even using a store.

[Get Started](#)[Demo](#)[▶ Watch Video Introduction](#)

Vue Router

The official Router for Vue.js

Expressive, configurable and convenient routing
for Vue.js



[Docs](#)[Modules](#)[Showcase](#)[Support](#)[Blog](#)

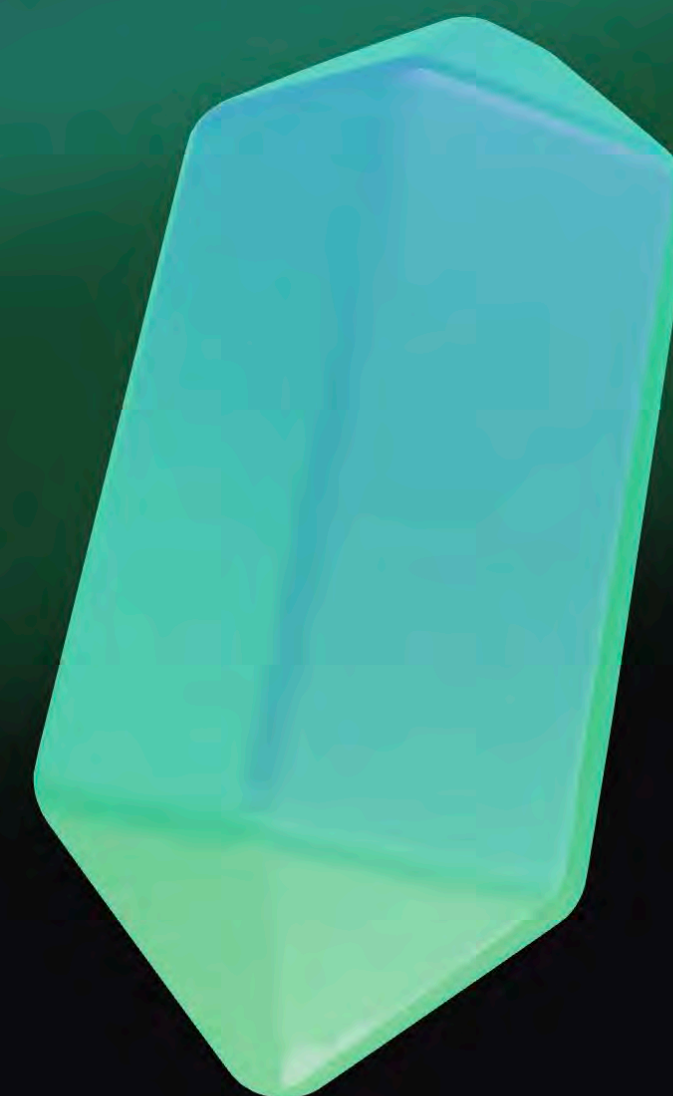
Nuxt 3.5 is out 🚀

The Intuitive Web Framework

Build your next Vue.js application with confidence using Nuxt. An open source framework under MIT license that makes web development simple and powerful.

[Get started](#)

```
> npx nuxi@latest init my-app
```



리액티브 데이터 선언

- ☑ 데이터를 사용하려면 리액티브하게 하자
 - 주로 객체형 또는 배열로 데이터 선언

```
<script setup>
import { reactive } from 'vue'
const state = reactive({ count: 0 })

function increment() {
  state.count++
}
</script>
<template>
  <button @click="increment">
    {{ state.count }}
  </button>
</template>
```

```
import { reactive } from 'vue'

const state = reactive({ count: 0 });
```

리액티브 데이터 제약 사항

- ☑ reactive() 의 두 가지 제약 사항
 - 객체 유형에만 작동 - Object & Array
 - 리액티브 객체에 동일한 참조를 유지 해야만 함

```
const state = reactive({ count: 0 })  
  
// 참조가 깨짐  
let n = state.count  
// 원본 데이터에 영향을 주지 않음  
n++  
  
// 참조가 깨짐  
let { count } = state  
// 원본 데이터에 영향을 주지 않음  
count++
```

```
let state = reactive({ count: 0 })
```

```
// 참조가 깨져서 더이상 리액티브 하지 않음  
state = reactive({ count: 1 })
```


리액티브 변수 ref()

- ❑ 리액티브의 제약 사항을 해결하기 위해 ref() 제공

```
import { ref } from 'vue'

const count = ref(0)
```

- value 속성을 통해 ref 객체내의 래핑된 인수를 반환

```
const count = ref(0)

console.log(count) // { value: 0 }
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

CSS 처리

styling

HTML 클래스 바인딩

☑ CSS 클래스 인라인 바인딩

```
const isActive = ref(true)  
const hasError = ref(false)
```

```
<div  
  class="static"  
  :class="{ active: isActive, 'text-danger': hasError }"  
></div>
```

☑ 인라인이 아닌 데이터 전체 바인딩

```
const classObject = reactive({  
  active: true,  
  'text-danger': false  
})
```

```
<div :class="classObject"></div>
```

HTML 클래스 바인딩

☑ 배열에 바인딩

```
const activeClass = ref('active')  
const errorClass = ref('text-danger')
```

```
<div :class="[activeClass, errorClass]"></div>
```

```
<div :class="[isActive ? activeClass : '', errorClass]"></div>
```

☑ 인라인이 아닌 데이터 전체 바인딩

```
const classObject = reactive({  
  active: true,  
  'text-danger': false  
})
```

```
<div :class="classObject"></div>
```

v-if , v-for

조건과 반복

v-if 와 v-else

- ☑ 디렉티브 v-if 는 조건부로 블록을 렌더링

- v-if 디렉티브 표현식이 참일 때만 해당 블록이 렌더링

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<button @click="awesome = !awesome">Toggle</button>
```

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<h1 v-else>Oh no 😞</h1>
```

- v-else-if

```
<div v-if="type === 'A'">
```

```
  A
```

```
</div>
```

```
<div v-else-if="type === 'B'">
```

```
  B
```

```
</div>
```

```
<div v-else-if="type === 'C'">
```

```
  C
```

```
</div>
```

```
<div v-else>
```

```
  Not A/B/C
```

```
</div>
```

v-if 와 v-else

- ☑ 디렉티브 v-if 는 조건부로 블록을 렌더링

- v-else-if

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

v-if 에 <template>

- ☑ v-if 는 디렉티브이기 때문에 단일 요소에만 연결되어야 함

- 둘 이상의 요소에서 사용할때는 <template> 사용

```
<template v-if="ok">  
  <h1>Title</h1>  
  <p>Paragraph 1</p>  
  <p>Paragraph 2</p>  
</template>
```

- ☑ v-show 와의 차이점

- v-show는 항상 렌더링 되고 DOM 이 남아있음
- <v-else> 지원하지 않으며 <template> 도 지원하지 않음

```
<h1 v-show="ok">Hello!</h1>
```


v-for, 리스트 렌더링

☑ 배열을 기반으로 리스트 목록을 렌더링

- items : 소스 데이터
- item : 배열 요소의 별칭

```
const items = ref([ { message: 'Foo' }, { message: 'Bar' } ])
```

```
<li v-for="item in items">  
  {{ item.message }}  
</li>
```

☑ 범위로

```
<span v-for="n in 10">{{ n }}</span>
```

v-for 와 <template>

- ☑ 여러 요소의 블록을 반복적으로 렌더링 할 때

```
<ul>  
  <template v-for="item in items">  
    <li>{{ item.msg }}</li>  
    <li class="divider" role="presentation"></li>  
  </template>  
</ul>
```

:key 속성 사용

- ☑ v-for 에서는 반드시 key 속성을 사용해야 함
 - “in-place patch” 전략
 - 항목의 순서가 변경된 경우 순서와 일치하도록 DOM 요소 이동 대신 사용

```
<div v-for="item in items" :key="item.id">  
  <!-- content -->  
</div>
```

```
<template v-for="todo in todos" :key="todo.name">  
  <li>{{ todo.name }}</li>  
</template>
```

양방향 바인딩

폼 바인딩



v-model, 양 방향 데이터 바인딩




```
2 프로그램 시작
3 {
4   변수 = 0
5   조건 반복 ( 변수 < 10 )
6   {
7     SMART: 화면 출력 후 줄 바꿈 = 100
8     변수 = 변수 + 1
9   }
10 }
```

양방향 바인딩 :: v-model

☑ 양방향 데이터 바인딩

◦ 주로 form 요소에서 사용됨



◦ input text, select box, textarea, radio button, check box..

```
<template>
  <input v-model="name" />
</template>
<script setup>
const name = ref("");
</script>
```

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

```
<div>Picked: {{ picked }}</div>

<input type="radio" id="one" value="One" v-model="picked" />
<label for="one">One</label>

<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
```

양방향 바인딩 :: v-model

☑ 선택 옵션

```
const selected = ref('A')

const options = ref([
  { text: 'One', value: 'A' },
  { text: 'Two', value: 'B' },
  { text: 'Three', value: 'C' }
])
```

```
<div>Selected: {{ selected }}</div>

<select v-model="selected" multiple>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```

```
<select v-model="selected">
  <option v-for="option in options" :value="option.value">
    {{ option.text }}
  </option>
</select>

<div>Selected: {{ selected }}</div>
```


라이프 사이클 훅

Life Cycle (of component) Hooks

라이프 사이클 훅 등록

- ☑ 컴포넌트가 초기 렌더링을 완료하고 호출 되는 함수
 - onMounted
 - cf) onBeforeMount - 컴포넌트가 마운트 되기 전 호출
- ☑ 화면이 업데이트 된 후 호출되는 함수
 - onUpdated
 - cf) onBeforeUpdate - 화면(컴포넌트) 업데이트 직전 호출

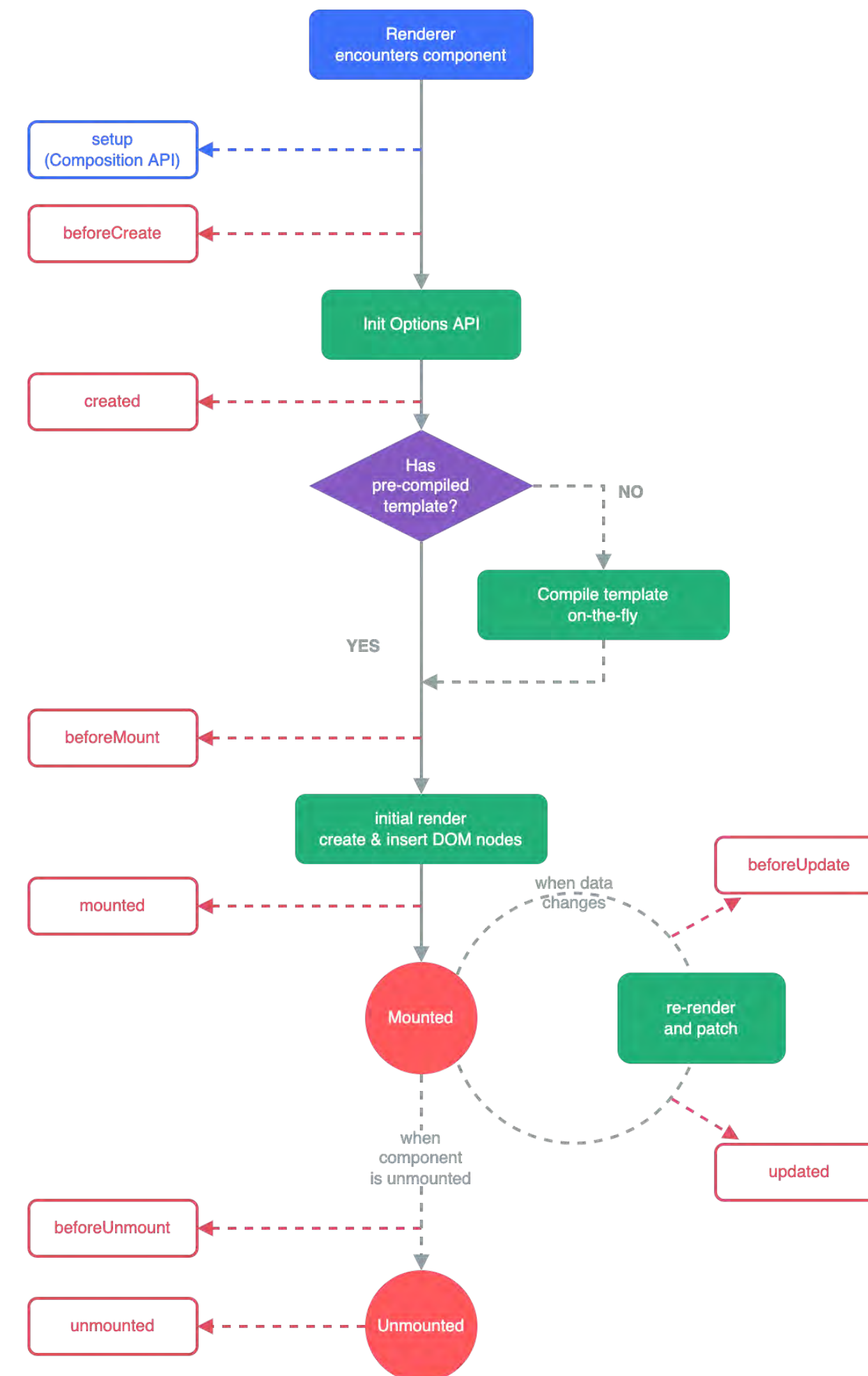
```
<script setup>
import { onMounted } from 'vue'

onMounted(() => {
  console.log(`the component is now mounted.`)
})
</script>
```

Vue Lifecycle Hooks - Managed

☑ Vue3 컴포넌트의 라이프사이클 이벤트

- **onBeforeMount:**
- **onMounted:** 렌더링 직후 모든 DOM 사용 가능
- **onBeforeUpdate:**
- **onUpdated:**
- **onActivated:**
- **onDeactivated:**



DOM 직접 참조

refs : DOM 직접 참조하기

- ☑ 기본적으로 DOM 참조는 선언적으로 수행
 - 예외적으로 DOM 을 직접 참조하기 위해서 사용

```
<input ref="input">
```

- 참조를 얻으려면 동일한 이름의 참조를 선언해야 함

```
<script setup>
import { ref, onMounted } from 'vue'

const input = ref(null)

onMounted(() => {
  input.value.focus()
})
</script>

<template>
  <input ref="input" />
</template>
```



Working with Data

Vue 에서 Data 관련 기능

계산된 속성

Computed properties

Computed Properties

- ❑ 템플릿에는 간단한 표현식만 사용
 - 템플릿에 로직이 많아지면 코드가 복잡해지고 유지관리가 어려움
 - 데이터가 변하면 자동적으로 호출되어 수행됨 (반응적, reactively)
 - Vue에 의해 캐시됨 (캐시가 필요하면 Computed Properties 사용)

```
const author = reactive({  
  name: 'John Doe',  
  books: [  
    'Vue 2 - Advanced Guide',  
    'Vue 3 - Basic Guide',  
    'Vue 4 - The Mystery'  
  ]  
})
```

잘못된 사용예 - 표현식이 너무 복잡함

```
<p>Has published books:</p>  
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

Computed Properties

☑ 리팩토링한 이전 페이지와 동일한 예

```
<script setup>
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})

// a computed ref, 계산된 데이터
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Computed 속성 :: setter 함수

- ☑ computed 속성은 기본적으로 getter 함수만 포함
 - 필요한 경우 setter 함수를 만들어 쓸 수 있음

```
<script setup>
import { ref, computed } from 'vue'

const firstName = ref('John')
const lastName = ref('Doe')

const fullName = computed({
  // getter
  get() {
    return firstName.value + ' ' + lastName.value
  },
  // setter
  set(newValue) {
    // Note: we are using destructuring assignment syntax here.
    [firstName.value, lastName.value] = newValue.split(' ')
  }
})
</script>
```


Computed 속성 :: 모범 사례

- ☑ getter 는 부작용(side-effect)이 없어야 합니다.
 - 함수 내에서 순수(pure)한 계산만 수행
 - 외부 I/O 를 수행하면 안됨 - DOM 변경, 비동기 요청..
- ☑ 계산된 값을 변경하지 마세요.
 - 계산된 속성(Computed Properties)에서 반환된 값은 파생된 데이터
 - 임시 스냅샷임
 - 원본 데이터가 변경 될 때마다 새 스냅샷이 생성됨
 - 즉 계산된 값은 읽기 전용
 - 새로운 계산된 상태가 필요하면 원본 데이터를 변경해서 사용

Watchers

Watcher

- ☑ Watcher 기본 예
 - 상태 변경에 대한 반응으로 “부작용” 을 수행해야 하는 경우 사용
 - DOM 변경, 비동기 작업의 결과에 따른 상태 변경
 - 상태가 변경될 때 마다 호출됨

Watcher

☑ 상태를 watch

- watch 의 첫번째 파라미터는 변경을 감지할 데이터(상태)임
 - ref(), 리액티브 객체, getter 함수..

```
const x = ref(0)
const y = ref(0)

// single ref
watch(x, (newX) => {
  console.log(`x is ${newX}`)
})

// getter
watch(
  () => x.value + y.value,
  (sum) => {
    console.log(`sum of x + y is: ${sum}`)
  }
)

// array of multiple sources
watch([x, () => y.value], ([newX, newY]) => {
  console.log(`x is ${newX} and y is ${newY}`)
})
```

```
const obj = reactive({ count: 0 })

// this won't work because we are passing a number to watch()
watch(obj.count, (count) => {
  console.log(`count is: ${count}`)
})
```

```
// instead, use a getter:
watch(
  () => obj.count,
  (count) => {
    console.log(`count is: ${count}`)
  }
)
```

watchEffect()

- ❑ 데이터 변경을 명시적으로 하지 않아도 데이터 변경시 콜백됨
 - 비동기로 서버 데이터를 가져올 때 사용할 수 있음

```
const url = ref('https://...')
const data = ref(null)

async function fetchData() {
  const response = await fetch(url.value)
  data.value = await response.json()
}
```

```
// 데이터 변경시 함수 호출됨
watch(url, fetchData)
```

```
// 함수 내에 데이터 변경시 자동 호출됨
watchEffect(async () => {
  const response = await fetch(url.value)
  data.value = await response.json()
})
```

데이터 교환의 사실상 표준

- tcp/ip -> http -> restful -> API -> json

HTTP 요청을 통해 웹(서버)에서 데이터 가져오기

HTTP 요청을 위한 환경 구성

- ☑ HTTP 서버 (API 서버) 필요
 - Fake server 사용 - http, restful api server, json
 - <https://jsonplaceholder.typicode.com/>

- ☑ HTTP 클라이언트 툴 - HTTP 테스트 클라이언트

- Postman



{JSON} Placeholder

Free fake API for testing and prototyping.

Powered by [JSON Server](#) + [LowDB](#)

As of Dec 2020, serving ~1.8 billion requests each month.

Vue 에서 사용하는 Http client

☑ **Axios** <https://github.com/axios/axios>

- 브라우저 자바스크립트와 노드에서 모두 사용가능 - universal library
- 가장 유명하고 많이 사용됨
- 자바스크립트에 내장된 HTTP client -> fetch() 함수
- 설치

```
$ npm install axios
```

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

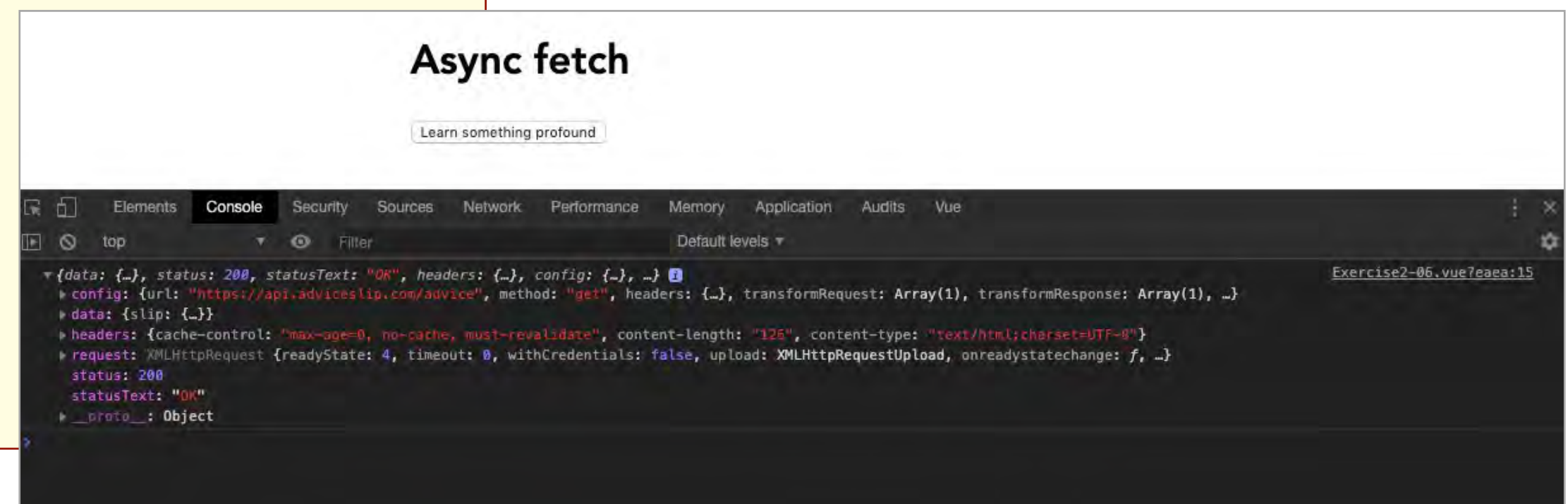
```
axios.get('https://jsonplaceholder.typicode.com/todos/1')  
  .then(res => console.log(res))  
  .catch(error => {  
    console.error(error);  
  });
```

비동기 메소드와 데이터 가져오기

☑ 자바스크립트의 비동기 함수 사용

○ Vue에서는 async 키워드를 메소드 앞에 포함하는 것으로 구현 시킴

```
export default {  
  data() {  
    return {  
      axiosResponse: {},  
    }  
  },  
  methods: {  
    async getApi() {  
      return axios.get('https://api.adviceslip.com/advice')  
        .then(response => {  
          this.axiosResponse = response.data  
        })  
    },  
  },  
}
```



Components

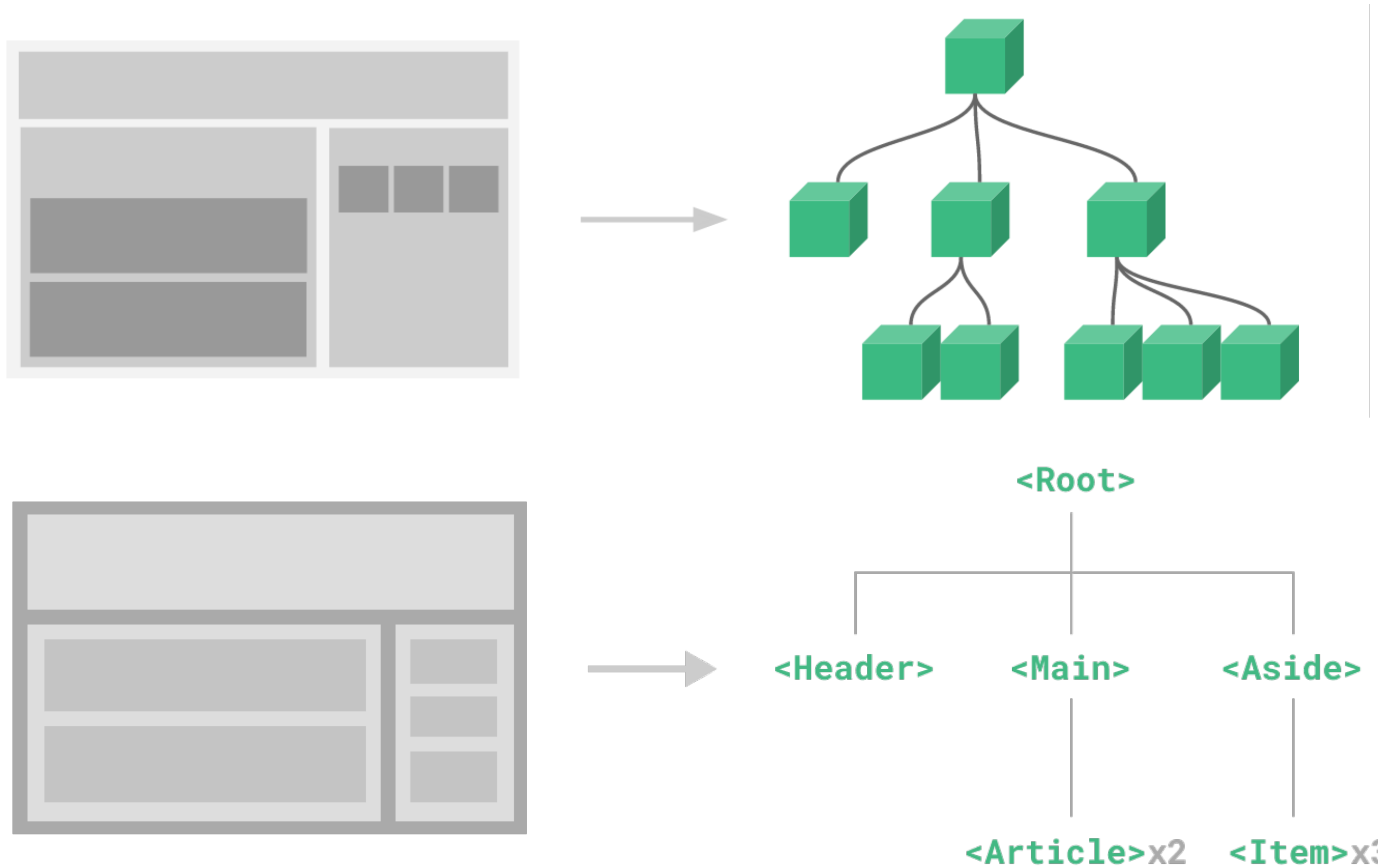
컴포넌트

개요

- ☑ Vue 애플리케이션 모듈화
 - 컴포넌트 계층구조(hierarchical architecture)와 중첩화(nesting)
- ☑ 컴포넌트 간 데이터 전달
 - 부모 -> 자식 : Props
 - 자식 -> 부모 : custom events
- ☑ 슬롯과 네임드 슬롯
- ☑ Mixin
- ☑ Global vs Local 컴포넌트
- ☑ 컴포넌트 Lifecycle Hooks

컴포넌트 트리

- ☑ 컴포넌트 = 화면 구성요소
 - 컴포넌트는 트리 구조 또는 중첩된(nested) 구조



컴포넌트 작성

☑ SFC (.vue 확장자) 를 사용하여 별도 파일에 Vue 컴포넌트 정의

```
<script setup>
import { ref } from 'vue'

const count = ref(0)
</script>

<template>
  <button @click="count++">You clicked me {{ count }} times.</button>
</template>
```

```
<script setup>
import ButtonCounter from './ButtonCounter.vue'
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

```
<h1>Here are many child components!</h1>
<ButtonCounter />
<ButtonCounter />
<ButtonCounter />
```

```
<button-counter></button-counter>
<button-counter></button-counter>
<button-counter></button-counter>
```

부모 컴포넌트가 자식 컴포넌트에 전달되는 데이터

Props

Props 선언

- ❑ 뷰 컴포넌트에서 props 를 사용하려면 먼저 선언이 필요

```
<script setup>
const props = defineProps(['foo'])

console.log(props.foo)
</script>
```

- 객체 구문 사용

```
// <script setup>
defineProps({
  title: String,
  likes: Number
})
```

Props 전달

- ☑ 컴포넌트는 화면과 데이터를 포함
 - 데이터는 Props 로 전달될 수 있음

```
<!-- BlogPost.vue -->
<script setup>
defineProps(['title'])
</script>

<template>
  <h4>{{ title }}</h4>
</template>
```

```
<BlogPost title="My journey with Vue" />
<BlogPost title="Blogging with Vue" />
<BlogPost title="Why Vue is so fun" />
```

부모 컴포넌트가 자식 컴포넌트에게 데이터 전달

Props 전달

- ☑ 정적 전달
 - ☑ 값을 직접 전달

```
<BlogPost title="My journey with Vue" />
```

- ☑ 동적 전달
 - 변수를 전달

```
<!-- 변수값 할당 -->  
<BlogPost :title="post.title" />  
  
<!-- 표현식을 이용한 변수값 전달 -->  
<BlogPost :title="post.title + ' by ' + post.author.name" />
```

Props Validation

☑ Props 타입 검증

```
defineProps({
  propA: Number,
  propB: [String, Number],
  propC: {
    type: String,
    required: true
  },
  propD: {
    type: Number,
    default: 100
  },
  propE: {
    type: Object,
    default(rawProps) {
      return { message: 'hello' }
    }
  },
  propF: {
    validator(value) {
      return ['success', 'warning', 'danger'].includes(value)
    }
  },
  propG: {
    type: Function,
    default() {
      return 'Default function'
    }
  }
})
```

타입 종류

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol

자식 → 부모 방향으로 데이터 전달 시 이벤트 활용

이벤트를 통한 데이터 전달

이벤트를 통한 데이터 전달

- ☑ 자식 → 부모에게 데이터를 전달 시
 - 이벤트 발생(emit) 과 이벤트 듣기(listening)
 - emit : 이벤트 등록

```
import { defineEmits } from 'vue'
const emits = defineEmits(['my-event'])
<!-- MyComponent -->
<button @click="emit('someEvent')">click me</button>
```

- v-on: 또는 @ : 이벤트 리스

```
<MyComponent @some-event="callback" />
```

```
<MyComponent @some-event.once="callback" />
```

이벤트 파라미터

- ☑ 이벤트 발생 시 파라미터로 데이터 포함 가능
 - 파라미터의 데이터를 이벤트 리스너에게 전달 가능

```
<button @click="emit('increaseBy', 1)">  
  Increase by 1  
</button>
```

- 이벤트 리스너에서 데이터 확보

```
<MyButton @increase-by="increaseCount" />
```

```
function increaseCount(n) {  
  count.value += n  
}
```

슬롯

slot

Slots, 기본 자식 Slots

- ☑ 슬롯, Slots
 - 컴포넌트 재사용 기법 중 하나, Component composition pattern
- ☑ 하위 컴포넌트로 HTML을 렌더링 가능

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

Box 컴포넌트

```
<template>  
  <div>  
    <Box>  
      <h3>This whole h3 is rendered in the slot</h3>  
    </Box>  
  </div>  
</template>  
<script>  
  import Box from './components/Box.vue'  
</script>
```

App 컴포넌트

Slots, Named Slots

☑ 네임드 슬롯, Named Slots

Article 컴포넌트

```
<template>
  <article>
    <div>Title: <slot name="title" /></div>
    <div>Excerpt: <slot name="excerpt" /></div>
  </article>
</template>
```

Title:

My Article Title

Excerpt:

First paragraph of content

Second paragraph of content

App 컴포넌트

```
<template>
  <div>
    <Article>
      <template v-slot:title>
        <h3>My Article Title</h3>
      </template>
      <template v-slot:excerpt>
        <p>First paragraph of content</p>
        <p>Second paragraph of content</p>
      </template>
    </Article>
  </div>
</template>
<script>
import Article from './components/Article.vue'
export default {
  components: {
    Article
  }
}
</script>
```

컴포넌트 글로벌 등록

- ❑ 컴포넌트를 전역으로 설정하여 어디서나 사용

```
<template>
  <button @click="$emit('click', $event)">
    <slot />
  </button>
</template>
```

CustomButton.vue

```
import { createApp } from 'vue'
import App from './App.vue'
import CustomButton from './components/CustomButton';

createApp(App)
  .component('CustomButton', CustomButton)
  .mount('#app');
```

main.js

```
<template>
  <div>
    <custom-button>Click Me</custom-button>
  </div>
</template>
```

컴포넌트 유연성 극대화

- ☑ 컴포넌트는 props 와 slot 을 입력으로 사용
 - 출력은 HTML로 렌더링됨
 - 컴포넌트의 유연성을 최대화 하려면 props와 슬롯을 활용

5장.

Vue Router

라우팅

라우팅이란?

- ☑ 동적 웹앱에서 중요한 부분 중 하나
 - 사용자가 원하는 위치(페이지)로 이동
 - 메뉴와 관련이 있음
 - 예) website.com/about URL 요청에 정보 페이지로 라우팅 됨
 - SPA 라우팅은 거의 동일 (Angular, React, Vue ..)
 - HTTP 요청을 처리하는 코드에 연결하는 방법을 결정하는 매칭 매커니즘

URL을 기반으로 원하는 리소스(페이지)에 연결 주는 기능

라우팅 : Vue Router

- ☑ Vue Router - Vue 의 공식적인 라우팅 라이브러리
 - 다양한 기능으로 페이지를 새로 고침 필요없이 전환

```
npm init vue
```

- ☑ Vue Router 설정

- Vue Router 는 기본적으로 제공되지 않음, create-vue 설정시 옵션 선택

```
✓ Project name: ... vue-router
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? > No
✓ Add ESLint for code quality? ... No / Yes
```

```
Scaffolding project in /Users/soongonkim/Desktop/vue-router...
```

- 또는 별도로 설치 가능

```
npm install vue-router
```

<router-view> 엘리먼트

☑ router-view 엘리먼트

- 사용자 요청 URL에 따라 해당 최신 뷰를 로딩해주는 Functional 컴포넌트
- 런타임 시ダイナ믹하게 콘텐츠 렌더링

◦ 기능

- 하위 컴포넌트 렌더링
- 라우팅 경로에 따라 컴포넌트 마운트/언마운트

```
<div id="app">  
  <router-view/>  
</div>
```

routes : 경로 설정

☑ router 폴더에 index.js 파일 생성

```
import router from './router'

const app = createApp(App)
app.use(router)
app.mount('#app')
```

☑ 경로 정의

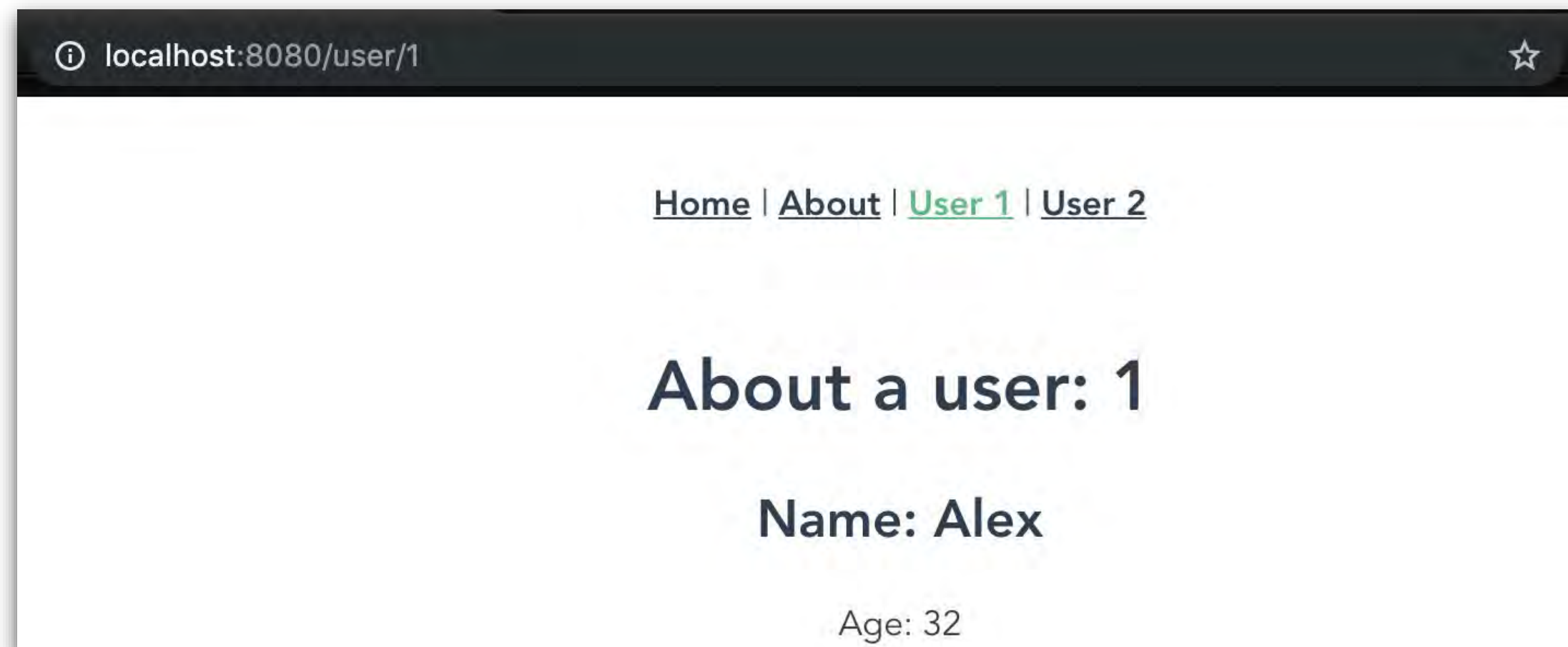
```
{
  path: '/',
  name: 'home',
  component: HomeView
},
{
  path: '/about',
  name: 'about',
  component: () => import('../views/AboutView.vue')
}
```

동적 라우팅

- ☑ 경로를 변수로 사용해서 여러 데이터 전달 가능

```
{  
  path: '/user/:id',  
  name: 'user',  
  component: () => import('../views/UserView.vue')  
}
```

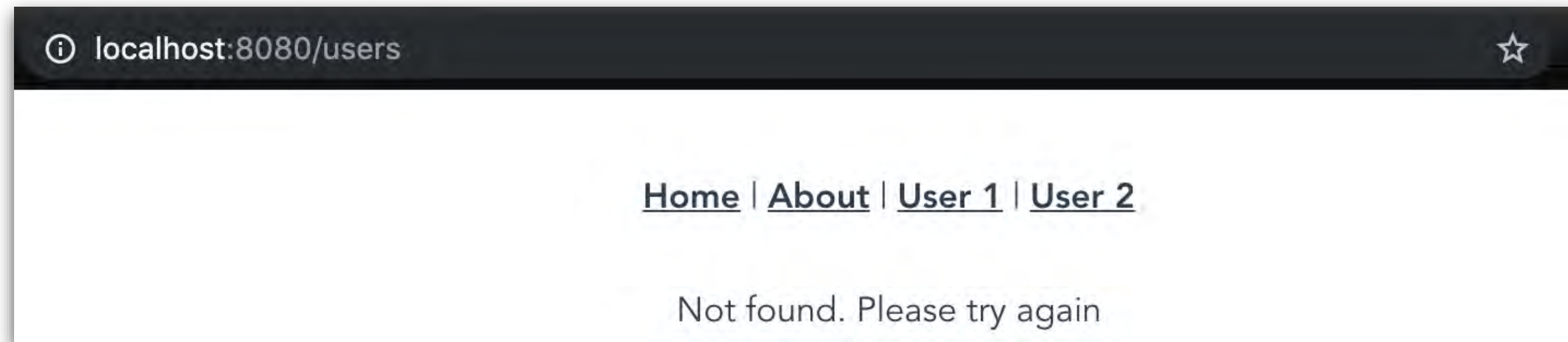
```
<template>  
  <div>  
    <h1>About a user: {{$route.params.id}}</h1>  
  </div>  
</template>
```



에러 경로 잡기

- ❑ Home 페이지 (/) 외 경로 URL 이 일치하지 않을 때
 - Error 페이지, 404 Not found 페이지 보여줘야 함

```
{  
  path: '/*',  
  name: '404',  
  component: () => import('../views/404.vue'),  
}
```



메뉴 링크 설정, Navigation links

☑ <router-link> : 경로 또는 이름으로 설정 가능

```
<div id="nav">  
  <router-link to="/">Home</router-link> |  
  <router-link to="/about">About</router-link>  
</div>
```

By url

```
<div id="nav">  
  <router-link to="home">Home</router-link> |  
  <router-link to="about">About</router-link> |  
</div>
```

By name

```
<router-link :to="{path: '/'}">Home</router-link>
```

```
<router-link :to="{ name: `messageFeed` }">Message Feed</router-link>
```

프로그래머틱 하게 접근

```
// literal string path
router.push('/users/eduardo')

// object with path
router.push({ path: '/users/eduardo' })

// named route with params to let the router build the url
router.push({ name: 'user', params: { username: 'eduardo' } })

// with query, resulting in /register?plan=private
router.push({ path: '/register', query: { plan: 'private' } })

// with hash, resulting in /about#team
router.push({ path: '/about', hash: '#team' })
```

```
<script setup>
  import { useRouter } from 'vue-router';
  const router = useRouter();

  function buttonClicked() {
    router.push('/');
  }
</script>
```

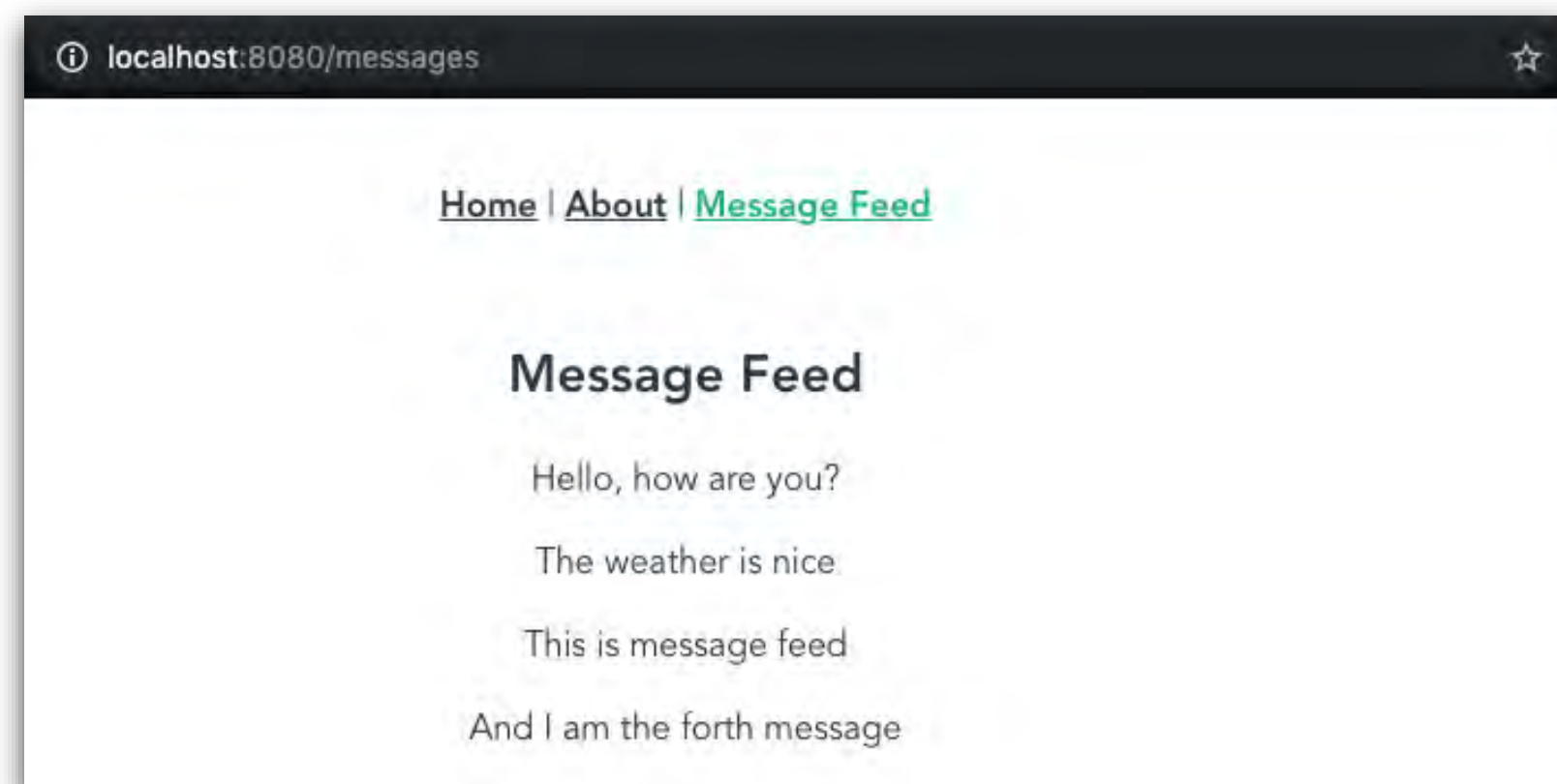
메뉴 링크 설정, Navigation links (계속)

- ☑ 뒤로가기 버튼 구형
 - router.go(steps) 사용

router.go(-1) // window.history.back() 과 유사 - 한칸 뒤로 가기

router.go(1) // window.history.forward() 과 유사 - 한칸 앞으로 가기

- ☑ 연습 ::
 - 메뉴링크 추가로 MessageFeed 메뉴 추가



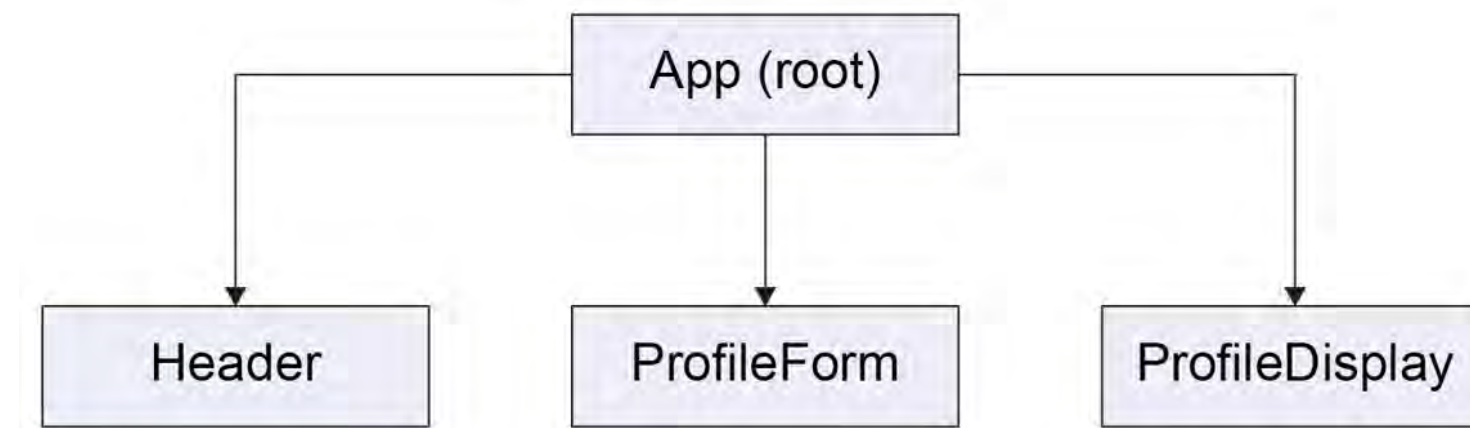
Pinia - global state management

상태관리, State management

글로벌 상태관리

- ☑ 로컬 컴포넌트에 데이터를 저장

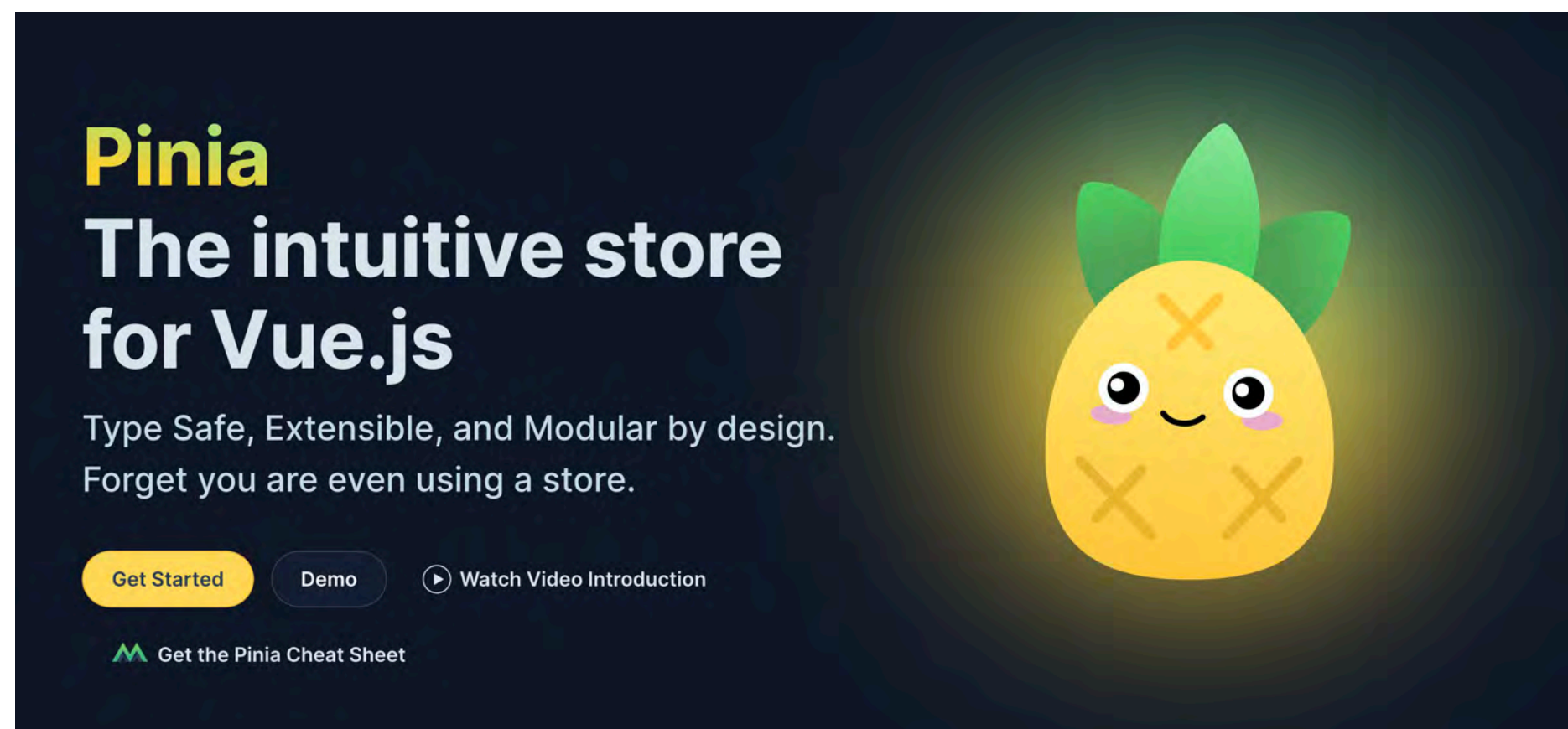
- Props 를 사용해 상태를 공유



컴포넌트 트리의 예

- ☑ 글로벌에 데이터를 저장 (global store)

- pinia 사용



pinia 란?

- ☑ Vue.js 애플리케이션에서 상태 관리를 위한 경량 라이브러리
 - Vue.js 공식 상태 관리 라이브러리인 Vuex의 대안으로
 - Eduardo San Martin Morote에 의해 개발
 - Vue 3의 Composition API를 활용하여 작성됨
 - 복잡성을 줄이면서 Vuex에서 제공하는 대부분의 기능을 제공

pinia 란?

☑ Pinia의 주요 기능

- 중앙 집중식 스토어: 애플리케이션의 모든 상태를 중앙에서 관리
- 타입스크립트 지원: 코드의 안정성과 개발자의 생산성을 높여줌
- Devtools 통합: 상태 변화를 시각적으로 쉽게 추적 및 디버깅 가능
- 서버 사이드 렌더링(SSR): 빠른 첫 페이지 로딩과 SEO 최적화
- 테스트 용이성:
 - 각 스토어는 독립적으로 생성되고 조작될 수 있어,
 - 테스트에서 필요한 상태를 쉽게 설정할 수 있습니다.
- Vue 2 및 Vue 3 모두와 호환

pinia 설치

- ❑ 처음부터 : npm init vue@latest 에서 선택

```
→ ~ npm init vue@latest  
  
Vue.js - The Progressive JavaScript Framework  
  
✓ Project name: ... vue-project  
✓ Add TypeScript? ... No / Yes  
✓ Add JSX Support? ... No / Yes  
✓ Add Vue Router for Single Page Application development? ... No / Yes  
? Add Pinia for state management? > No / Yes
```

- ❑ 나중에 설정 가능

- npm install pinia
- main.js 에 코드 추가

```
import { createApp } from 'vue'  
import { createPinia } from 'pinia'  
import App from './App.vue'  
const app = createApp(App)  
app.use(createPinia())  
app.mount('#app')
```

Pinia 로 Store 작성

☑ Store 정의

```
import { ref, computed } from 'vue'
import { defineStore } from 'pinia'

export const useCounterStore = defineStore('counter', () => {
  // state 정의
  const count = ref(0)
  // computed() 는 getters 에 해당
  const doubleCount = computed(() => count.value * 2)
  // 일반 함수는 actions 에 해당
  function increment() {
    count.value++
  }

  return { count, doubleCount, increment }
})
```

Store 사용

- ☑ Store 는 여러개를 작성/사용 가능
 - 각각의 store 는 별도의 파일에 작성

```
<script setup>
import { useCounterStore } from '@stores/counter'

// 컴포넌트에서 `store` 사용 가능 ✨
const store = useCounterStore()
</script>
```

```
<template>
  <p>{{ store.counter }}</p>
</template>
```

state 값에 직접 접근

VUE3 영화 앱

Vue 영화 앱 - 소개

 뷰 무비앱

현재 상영중

인기 상영작

높은 평점

 Search...



트랜스포머: 비스트의 서막

전 우주의 행성을 집어삼키는 절대자, '유니크론'의 부하 '스커지'는 '테러콘'들을 이끌고 지구에 당도한다. 그에 맞서기 위해 지구에 정체를 숨기고 있던 트랜스포머 '오투부' 구단이 무섭게 드러내고 또 다른 트랜스포머



세인트 세이아: 더 비기닝

강력한 힘을 가진 전쟁과 지혜의 신 '아테나'의 환생! 세상을 구할 수호 기사가 깨어난다! 어린 시절 누나 '패트리샤'가 납치된 후 홀로 자란 '세이아' 슬럼가 지하 격투장에서 벌어지는 격투로 구구이 삭아가던 중 위기의 사



사운드 오브 프리덤

설명 없음

Vue 영화 앱 - 컴포넌트 구조

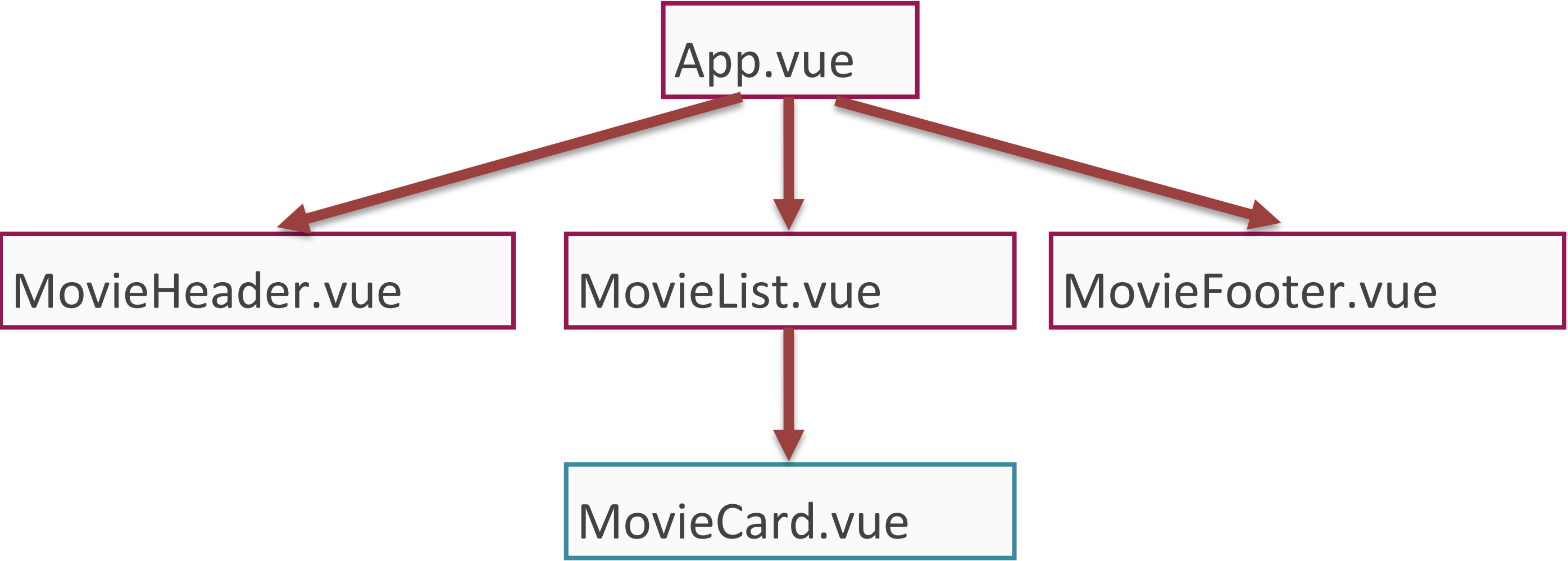
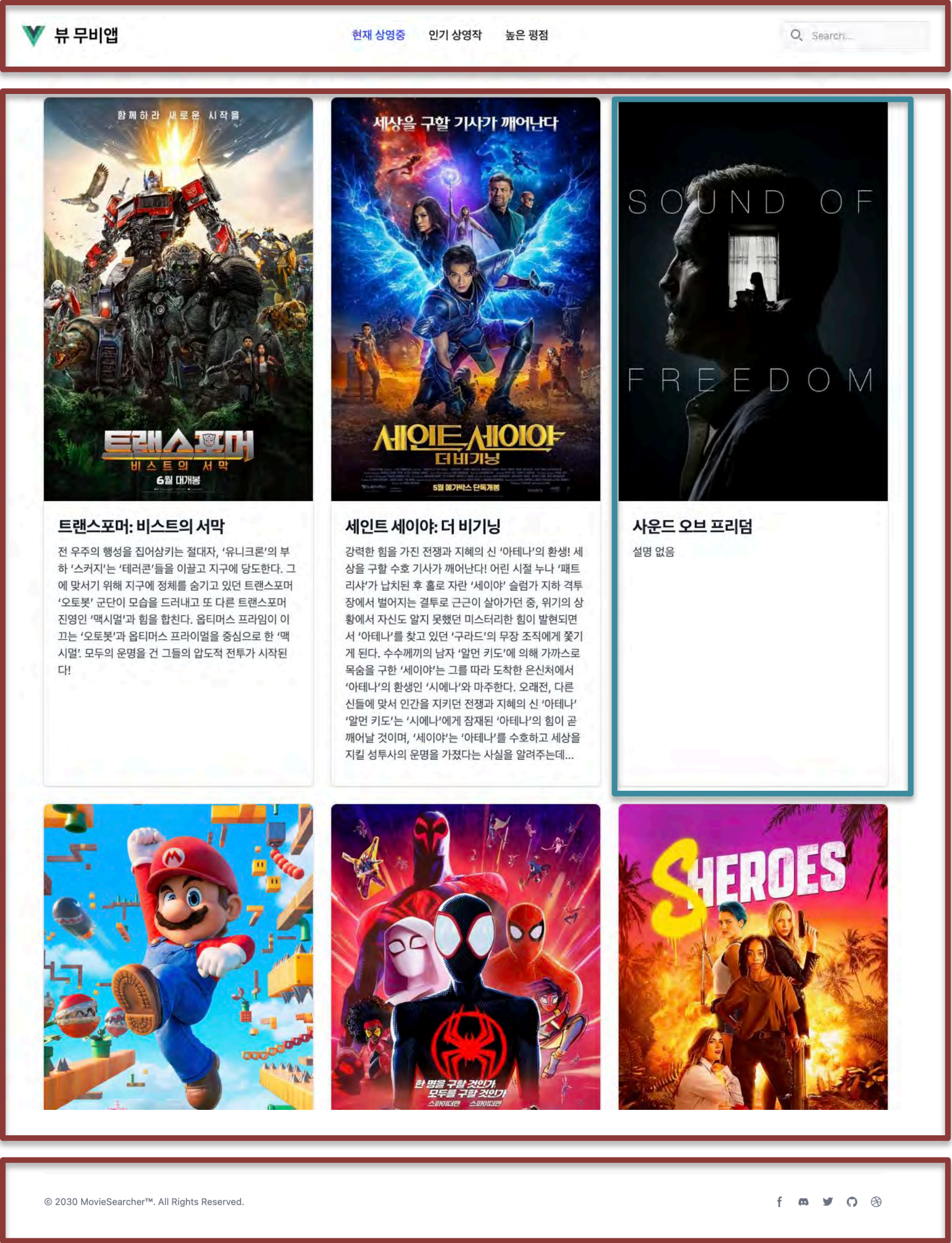
MovieHeader.vue

MovieCard.vue

MovieList.vue

MovieFooter.vue

Vue 영화 앱



Vue 영화 앱 - 앱 구성

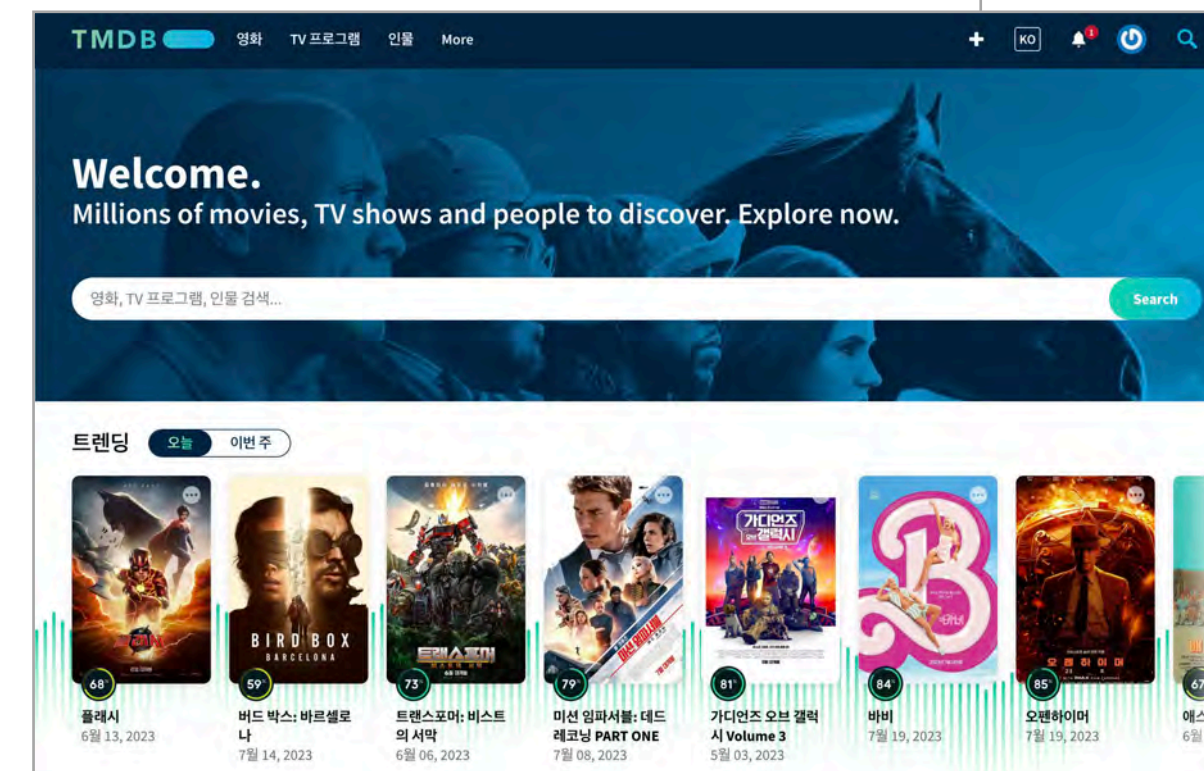
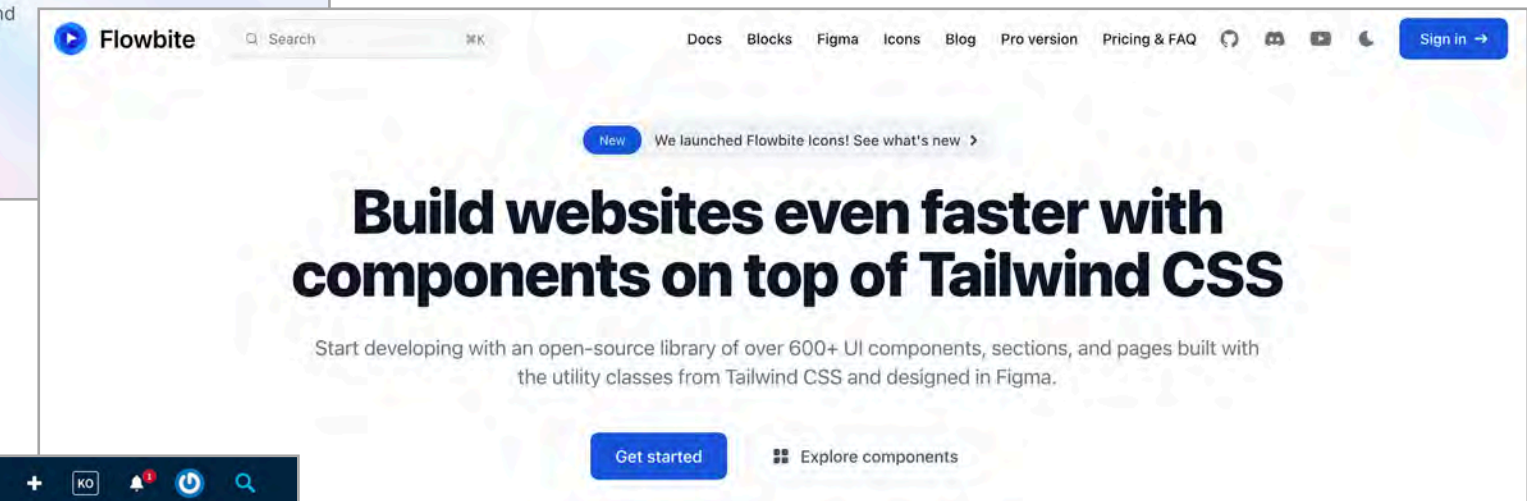
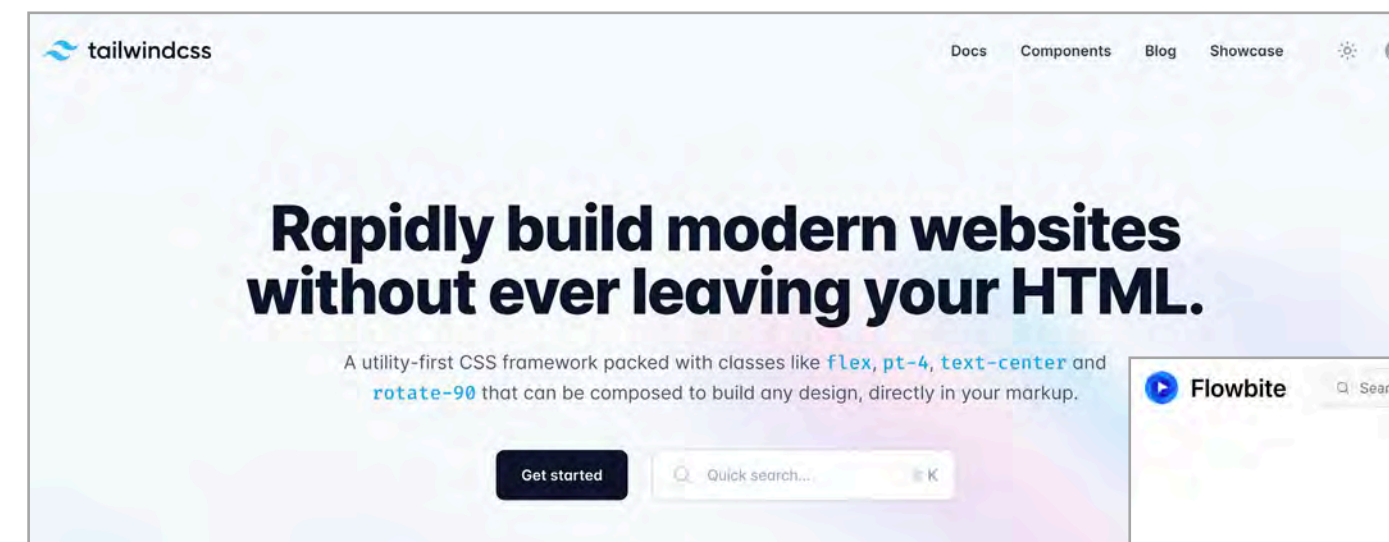
☑ 화면 구성 - CSS 프레임워크

○ Tailwind CSS

- 유틸리티 기반 CSS 프레임워크
- Tailwind 를 이용한 Flowbite 사용

☒ 기능 구현

- 영화정보 API - TMDB 사용
- Data Fetching
- 컴포저블 (Composables)



Vue 영화 앱 - 개발 순서 및 구현 기능

- ☑ vite 로 react 앱 스캐폴딩
- ☑ Tailwind CSS 설정
- ☑ 기능 구현
 - TMDB API 사용 - 영화 데이터 Fetching
 - 현재 상영중, 인기 상영작, 높은 평점 기능 구현
 - 영화 검색 기능 구현
- ☑ 컴포저블 사용으로 리팩토링
 - 재사용성과 가독성 증가

Vue 영화 앱 - 최종 구현

 뷰 무비앱

현재 상영중

인기 상영작

높은 평점

 Search...



트랜스포머: 비스트의 서막

전 우주의 행성을 집어삼키는 절대자, '유니크론'의 부하 '스커지'는 '테러콘'들을 이끌고 지구에 당도한다. 그에 맞서기 위해 지구에 정체를 숨기고 있던 트랜스포머 '오투부' 구단이 무섭게 드러내고 또 다른 트랜스포머



세인트 세이아: 더 비기닝

강력한 힘을 가진 전쟁과 지혜의 신 '아테나'의 환생! 세상을 구할 수호 기사가 깨어난다! 어린 시절 누나 '패트리샤'가 납치된 후 홀로 자란 '세이아' 슬럼가 지하 격투장에서 벌어지는 격투로 구구이 삶아가더 쥔 위기의 삶



사운드 오브 프리덤

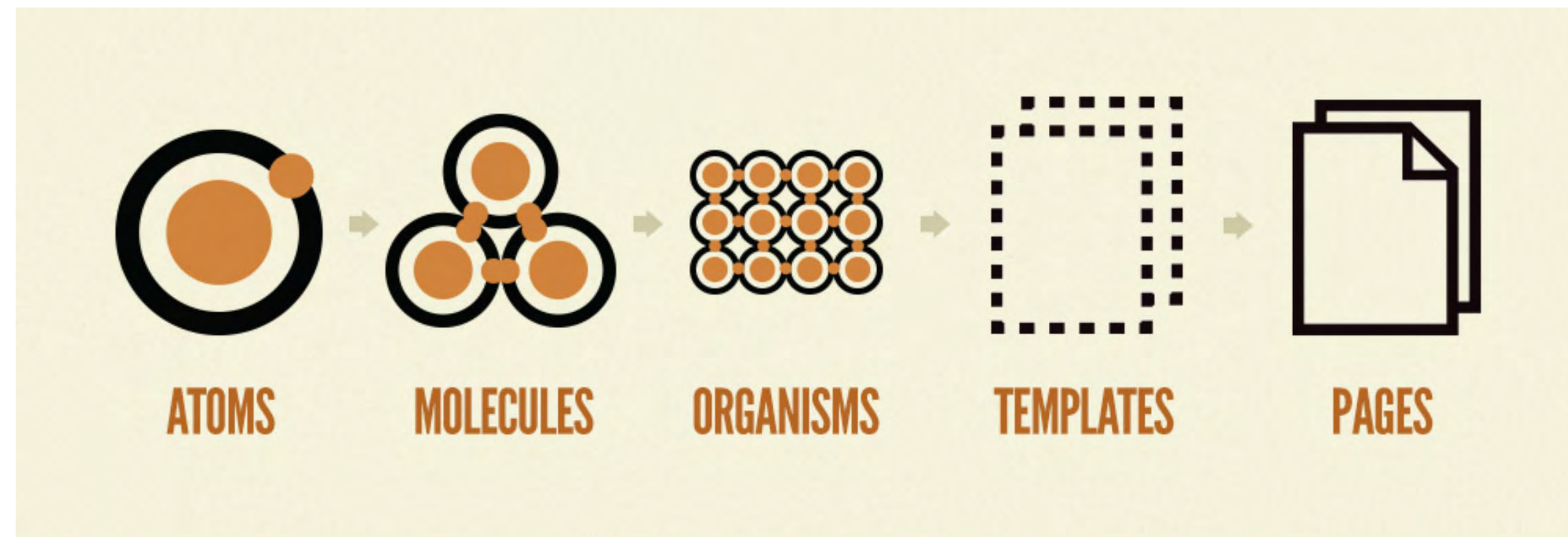
설명 없음

Atomic Design

컴포넌트와 아토믹 디자인

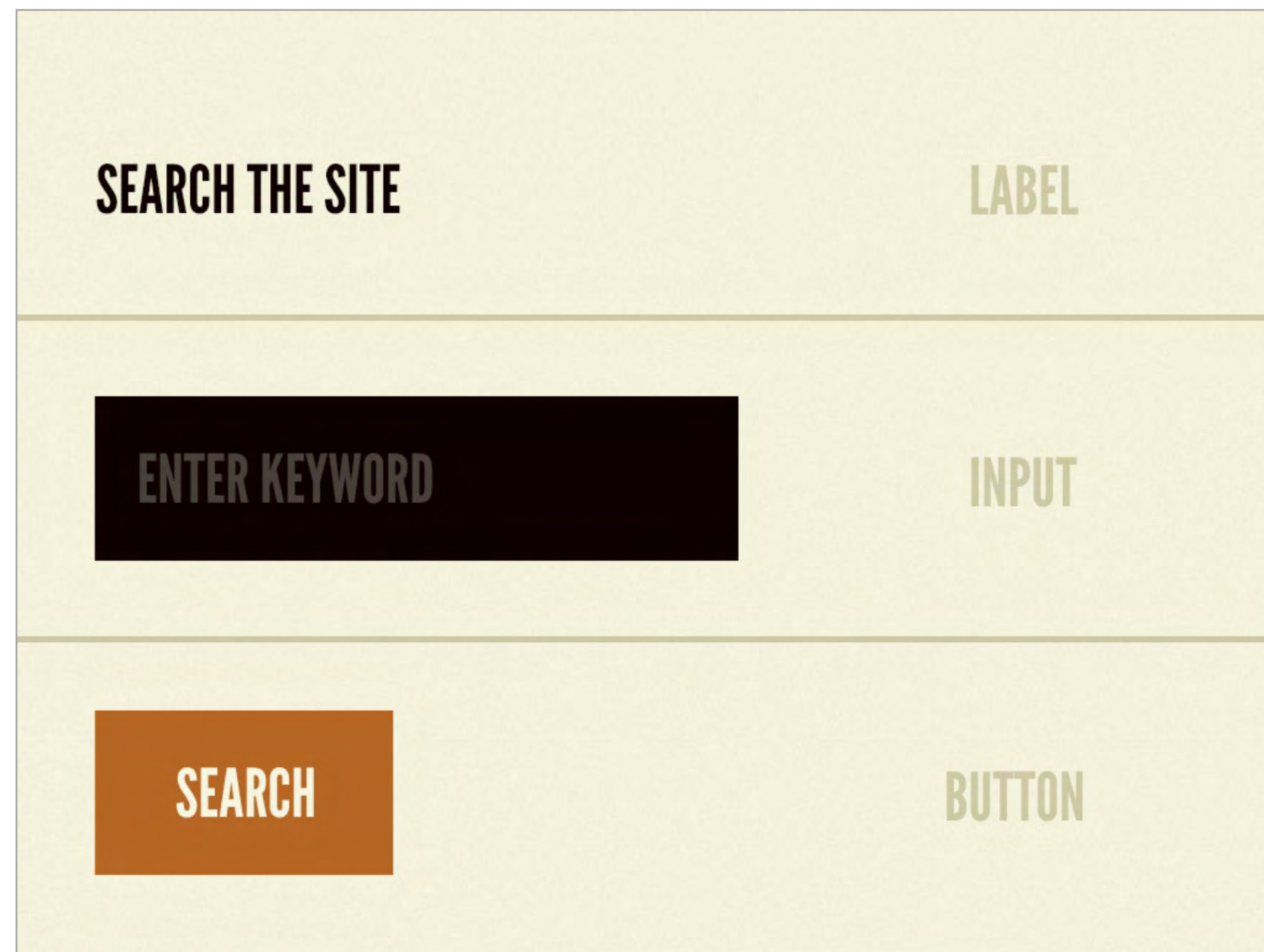
아토믹 디자인 구성

- ☑ 아토믹 디자인은 디자인 시스템을 만드는 하나의 방법론
- ☑ 5개의 구분된 단계가 있음
 - Atoms (원자)
 - Molecules (분자)
 - Organisms (유기체)
 - Templates
 - Pages



Atoms

- ☑ 원자는 물질의 기본 빌딩 블록
- ☑ 웹 인터페이스의 텍스트 레이블, 인풋필드, 버튼 등
 - 컬러 팔레트
 - 폰트
 - 애니메이션
- ☑ 재사용이 중요



Molecules

- ☑ 원자가 결합된 형태
 - 화합물의 가장 작은 기본 단위
 - 그 자체의 특성을 가지고 있음
 - 디자인 시스템의 중추역할

SEARCH THE SITE

ENTER KEYWORD

SEARCH

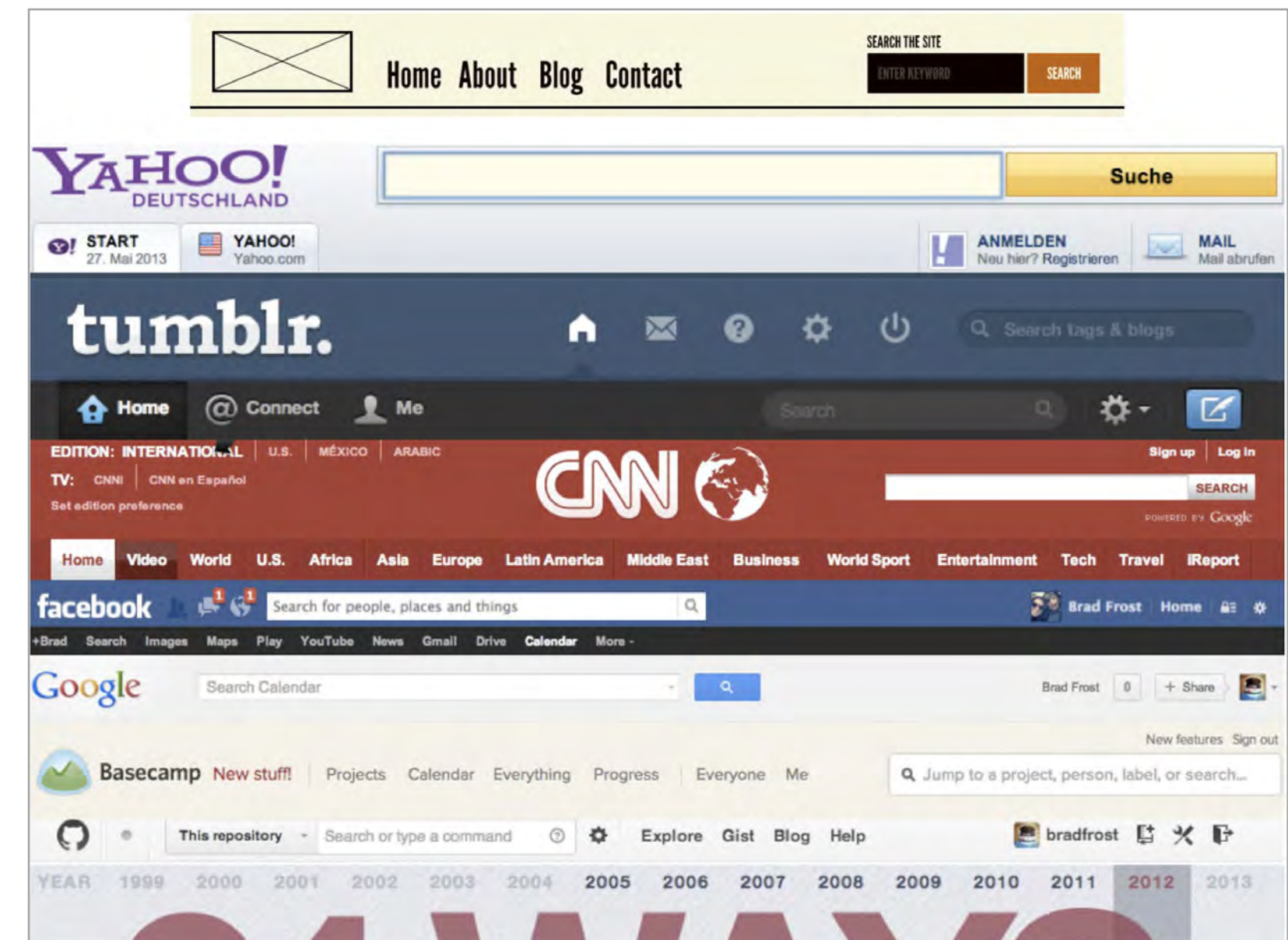
Organisms

- ☑ 분자들을 결합하여 유기체를 형성
 - 비교적 복잡하며 인터페이스에서 구분된 영역을 형성
 - 헤더, 콘텐츠, 푸터, 사이드 바 등으로 구성 가능
 - 특정요소가 반복될 때 사용(카드 등)
 - 섹션으로 구성 가능

Standalone

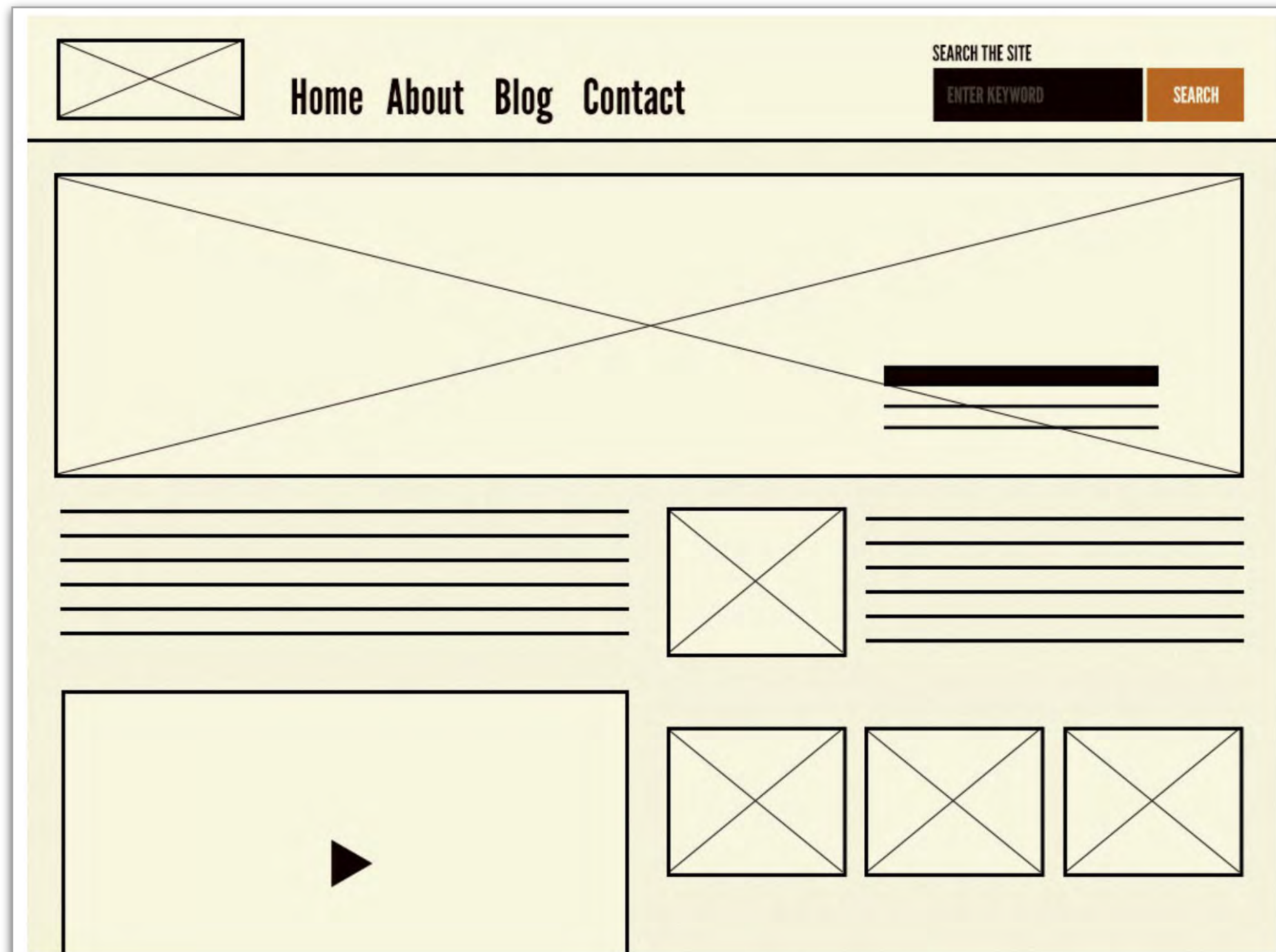
Portable

Reusable



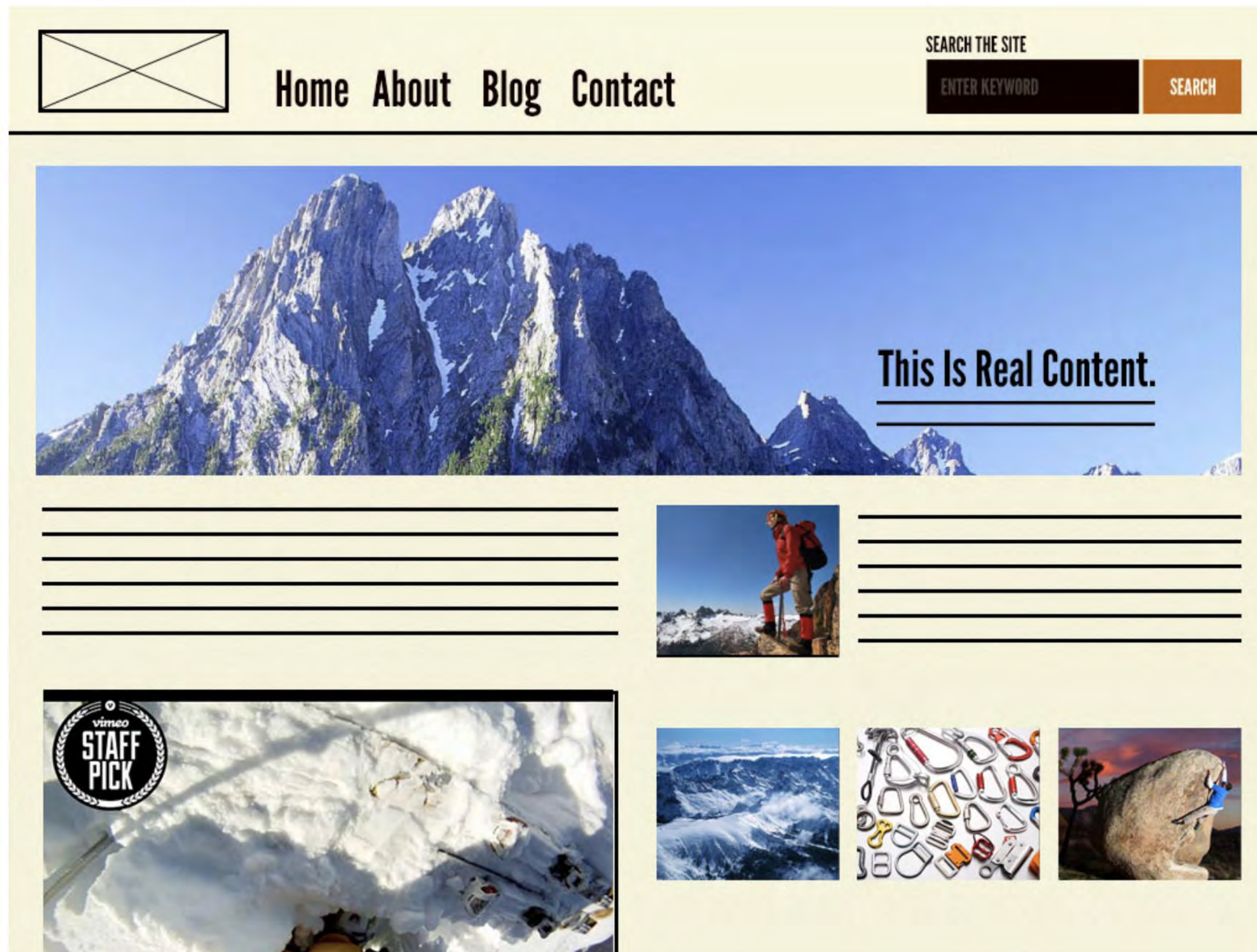
Templates

- ❑ 고객사와 최종 결과물에 더 적합한 언어 적용
- ❑ 페이지를 구성하기 위해 유기물을 적절히 레이아웃 (Wireframe)
- ❑ 분자와 유기체에 대한 맥락(context)을 제공



Pages

- ☑ 템플릿의 특정 인스턴스
 - 템플릿에 데이터가 추가되면 페이지가 됨



아토믹 디자인의 장점

- ☑ 구조적 사고방식

- UI 분해 : 틀과 부품
- 프론트엔드 개발에서의 컴포넌트 접근법에 적용 가능

- ☑ 팀 개발에 응용

- 디자이너와 개발자 간의 의사소통이 개선됨

컴포넌트와 아토믹 디자인과의 궁합

☑ 비대해지지 않는 컴포넌트

- 컴포넌트에 너무 많은 기능이 있는 것이 좋지 않음
- Atoms 이나 Molecules 로 컴포넌트 설계
- 설계가 무너지는 컴포넌트 패턴을 피할 수 있음
- 최적화, 리팩토링, 유지보수가 쉬워짐 => evolution!
- Make each program do one thing well!

☑ 확장성

- 컴포넌트 단위를 기반으로 조합하는 방법을 사용
- 컴포넌트의 집합체로 확장이 가능

Software Quality

제품의 품질

품질 유지

- ☑ 프론트엔드 역할이 중요해 짐
- ☑ 품질 유지와 접근법과 관련된 라이브러리
 - 테스트 프레임워크
 - Jest - from Facebook
 - Enzyme
 - Lint
 - ESLint
 - StyleLint
 - 정적 자료형
 - Flow
 - Typescript
 - 스타일 가이드 활용

제품의 품질

☑ 소프트웨어 품질

- 사양을 만족하는가? 버그는 없는가?
- 성능(Performance) - 전환율(CVR)과 관련
- 접근성(Accessibility)
- 확장성(Scalability)
- 유지보수성
- 문서화

☑ 소스코드 품질 - 아름다운 코드에 대한 평가

- 가독성
- 소스코드 변경의 편의성
- 테스트 가능 여부

품질 유지의 필요성과 핵심 포인트

- ☑ 프로그램(서비스)은 시간이 지날수록 품질이 떨어짐
 - 기술부채가 쌓임 - 새로운 기술이 계속해서 등장
 - 서비스 변경이 어려워짐
 - 소스코드 품질이 나빠지면 개선 비용이 증가해 개선이 어려워짐

- ☑ 품질 유지의 포인트
 - 가시성
 - 제품 구축 과정에서 가능한 실시간으로 현재의 품질 상태 시각화 필요
 - 테스트와 리팩토링
 - 구축 프로세스에 적절히 포함시켜 계속 품질유지를 실현
 - 조직
 - 팀 구성, 의사소통

컴포넌트 테스트

- ☑ 컴포넌트 정해진 '동작' 이 있음
 - 컴포넌트의 정해진 동작을 일괄적으로 확인하는 작업
- ☑ 리액트 컴포넌트는 함수와 같이 동작함
 - 함수형(Functional) 컴포넌트는 props 를 입력받아 render 를 반환
 - 컨테이너형 컴포넌트도 입력이 없을 뿐 출력이 존재
- ☑ 리액트 컴포넌트는 각각의 컴포넌트를 단위로 봄
 - 단위 테스트(Unit Test) 의 단위가 컴포넌트가 됨
- ☑ 컴포넌트의 로직과 역할에 따른 관점을 가지고 테스트 수행
 - 컴포넌트 개발자 테스트 케이스를 작성하는 것이 일반적
 - TDD 도 고려

테스트 프레임워크

- ☑ 단위 테스트는 하나씩 수행하지 않는다.
 - 단위 테스트를 일괄적으로 수행
 - 테스트 코드의 작성부터 실행까지 지원하는 것이 테스트 프레임워크
- ☑ Jest
 - 리액트 컴포넌트 테스트 프레임워크의 최신판
 - 페이스북에서 개발
 - create-react-app 에서 기본으로 제공
 - npm run test 로 테스트 수행

Jest 테스트 구조 분석

☑ App 컴포넌트의 테스트 코드

○ src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
  ReactDOM.unmountComponentAtNode(div);
});
```

\$ npm test

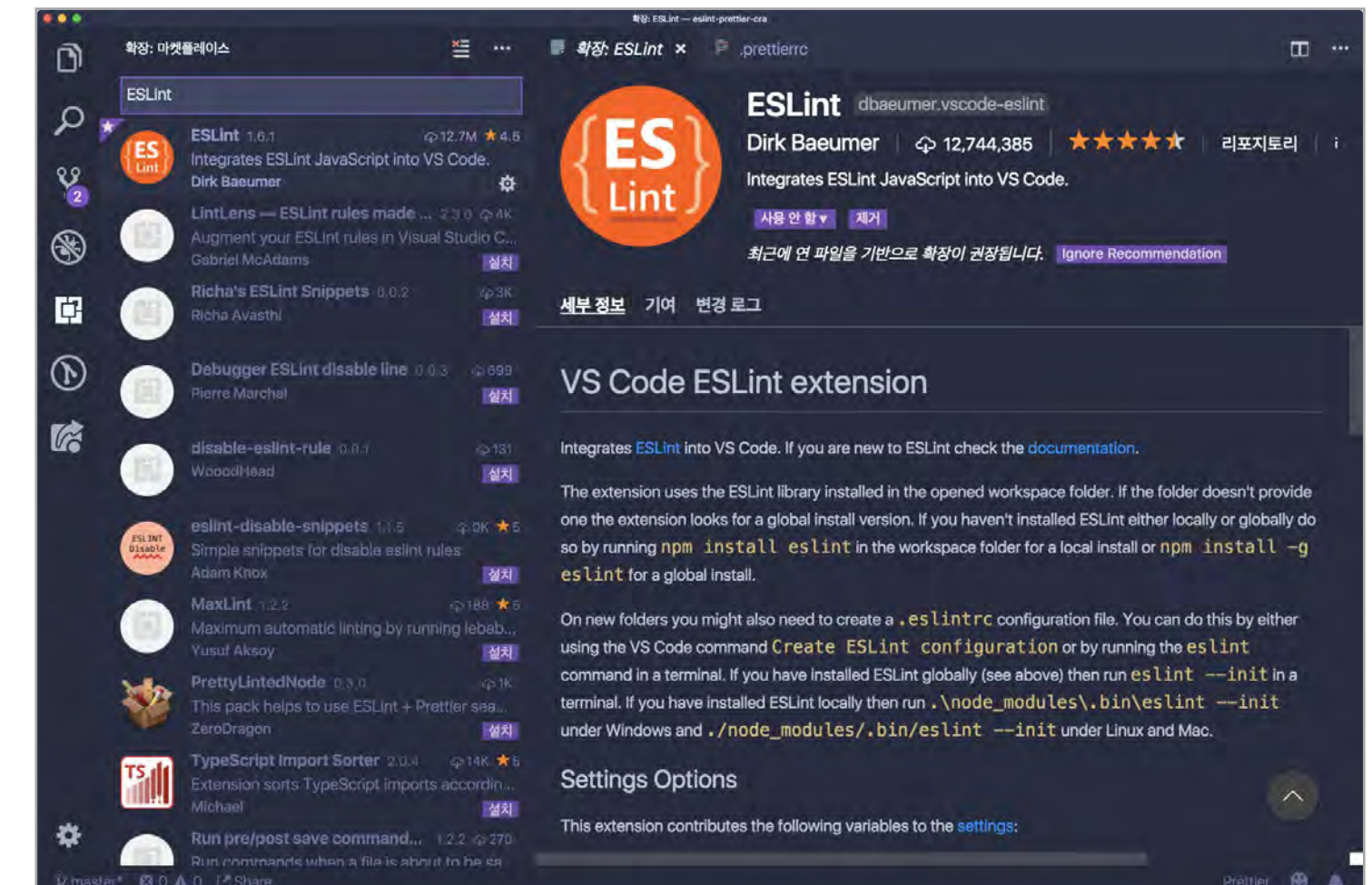
ESLint

☑ 정적 소스 분석 툴

☑ JavaScript(ECMAScript) 용 Lint 도구

- ESLint 가 사용이 쉬워 표준으로 많이 사용됨
- 독자적인 규칙을 만들 수 있음

VSCode 확장으로 설치 가능



동적 자료형 언어 vs. 정적 자료형 언어

☑ 자료형, Data Type

- 자바스크립트는 자료형을 동적으로 할당해 줌 - Dynamic Typing
- Dynamic 언어는 사용이 간편할 수는 있으나 가독성이 떨어짐

☑ 자료형의 이점

- 리액트는 컴포넌트, 컴포넌트는 재사용
- 잘못된 자료형이 전달되었을 때 어느 시점인지를 알기 어려움

☑ Typescript

- MS에서 만든 자바스크립트의 슈퍼셋

스타일 가이드

☑ 스타일 가이드

- 프론트엔드에서는 코드 스타일 가이드와 UI 스타일 가이드가 있음
- 팀 프로젝트에서 전체 코드나 UI의 스타일의 표준을 정하는 작업
- 브랜딩 이미지와 직결

☑ 에어 비엔비 스타일 가이드

- <https://github.com/airbnb/javascript>

☑ Storybook

- 스타일 가이드 만드는 도구
- <https://storybook.js.org/>

Vue 다음 과정은..

Vue 기본을 넘어서..

- ☑ Nuxt.js

- <https://kdydesign.github.io/2019/04/10/nuxtjs-tutorial/>

- ☑ Svelte?? - next vue or react

- ☑ Vue with TypeScript

- <https://www.typescriptlang.org/>
- <https://kr.vuejs.org/v2/guide/typescript.html>

- ☑ 프론트엔드 개발자 로드맵

- <https://github.com/kamranahmedse/developer-roadmap>

수고하셨습니다.

김순곤

soongon@huccloud.co.kr