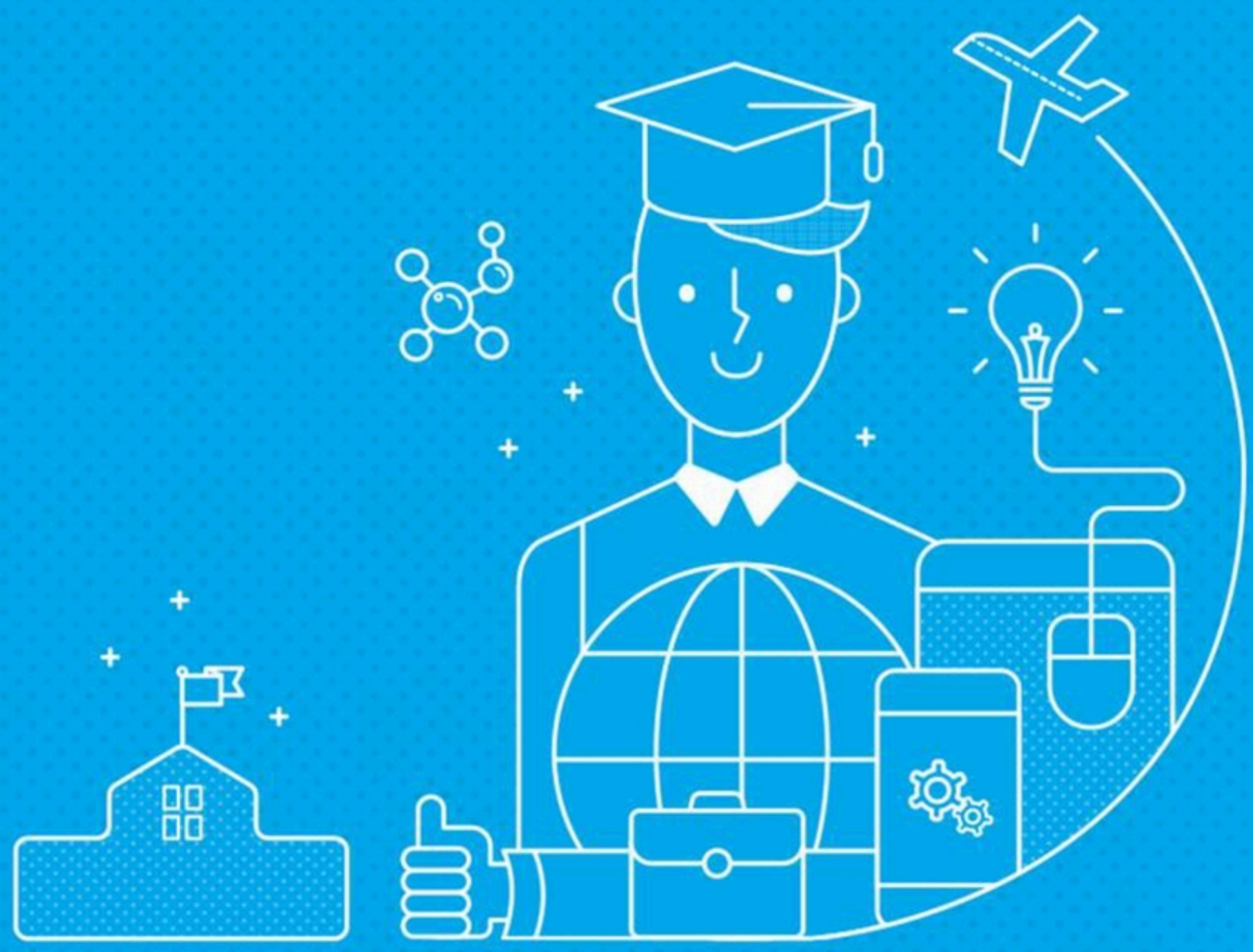
SAMSUNG
SW
ACADEMY
FOR
YOUTH

Vue.js

삼성 청년 SW 아카데미



CONTENTS

Samsung Software Academy For Youth

1장. Vue 시작하기 ---------- 3

2장. Vue인스턴스 ---------- 17

3장. Vue 디렉티브 ---------- 35

4장. Vue와 CSS ---------- 57

5장. Rendering ---------- 60

6장. 이벤트 핸들링 ---------- 70

7장. 폼 데이터 처리 ---------- 76

8장. Vue를 이용한 애플리케이션 -- 86

9장. Component ---------- 94

10장. 라우팅 ---------- 108

11장. Vue 프로젝트 1 ---------- 115

12장. Vue 프로젝트 2 ---------- 159

13장. 로그인과 상태관리 ---------- 174

1. Vue 시작하기 3
1.1 개발환경 3
1.2 개발순서 6
2. Vue인스턴스 17
2.1 Vue 인스턴스 생성 17
2.2 Vue 인스턴스 옵션 20
3. Vue 디렉티브 35
3.1 기본 디렉티브 35
3.2 바인딩 48
3.3 사용자 정의 디렉티브 54
4. Vue와 CSS 57
4.1 CSS 기본 57
5. Rendering 60
5.1 HTML리스트 데이터와 폼 데이터 바인딩 60
5.2 HTML리스트 데이터와 폼 데이터 렌더링 63
6. 이벤트 핸들링 70
6.1 이벤트 70
7. 폼 데이터 처리 76
7.1 리스트 데이터 76
7.2 폼 데이터 79
8. Vue를 이용한 애플리케이션 86
8.1 스프링 부트를 이용한 Restful 서비스 구축 86
8.2 Vue에서 서비스 사용하기 91
9. Component 94
9.1 템플릿 만들기 95
9.2 컴포넌트 99

9.3 컴포넌트 통합하기 103
10. 라우팅 108
10.1 라우팅 108
10.2 SPA 구현 111
11. Vue 프로젝트1 115
11.2 Visual Studio Code 120
11.3 서비스 확인 124
11.5 사원정보 삭제, 찾기 132
11.6 사원정보 추가 142
11.7 사원정보 수정 154
12. Vue 프로젝트2 159
12.1 HRM 아키텍쳐 159
13. 로그인과 상태관리 174
13.1 회원가입 174

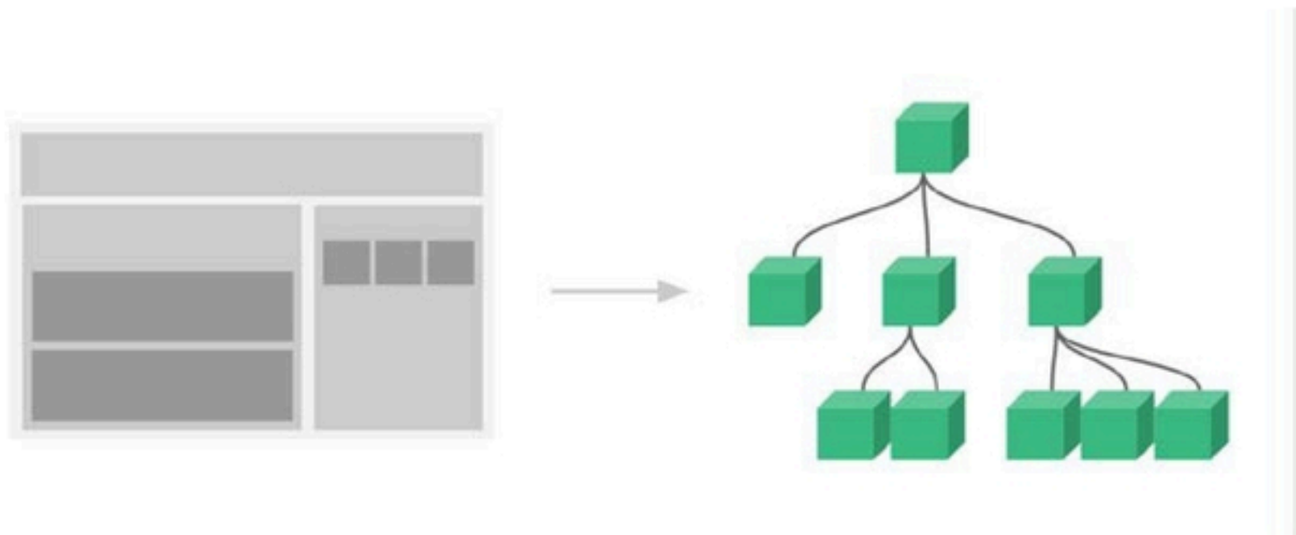
1. Vue 시작하기

1.1 개발환경

1.1.1 Vue.js devtools

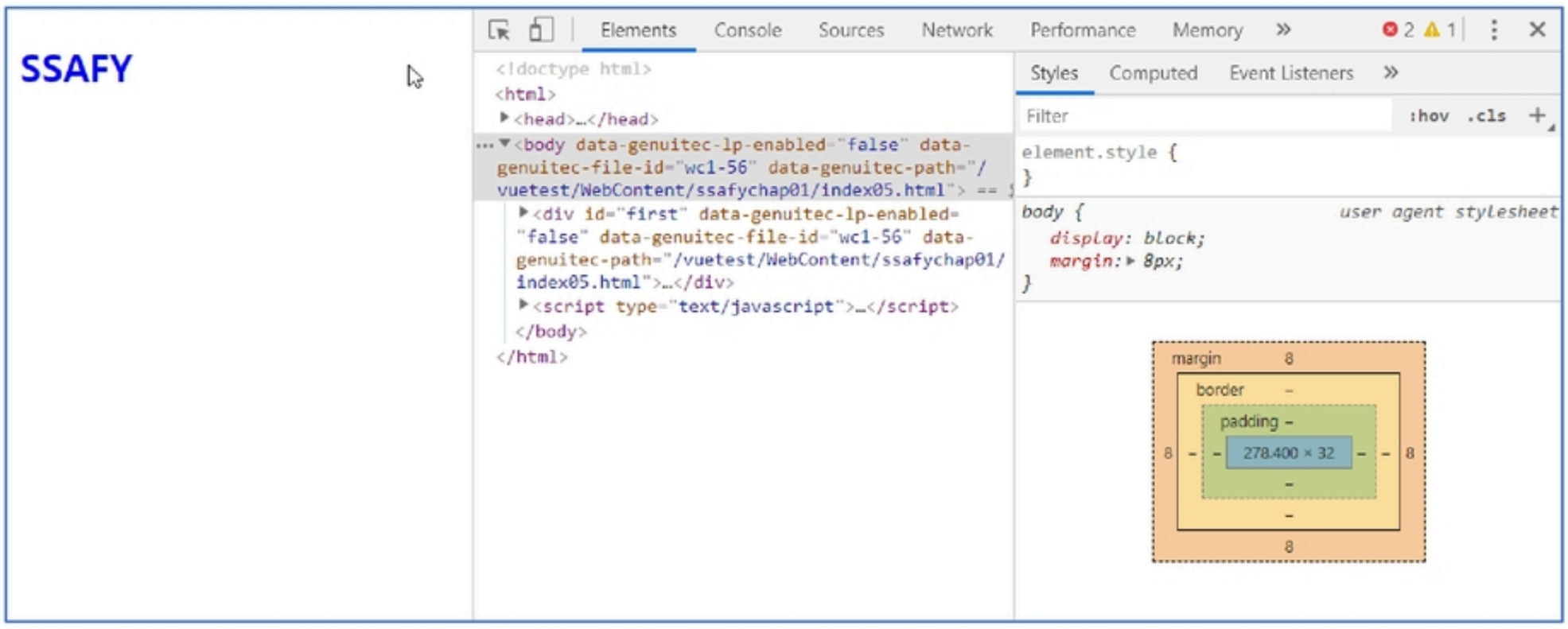
Vue 는 사용자 인터페이스를 위한 Front-End 에 적용할 수 있는 프레임워크 입니다. jQuery 등이 작업했던 화면의 데이터나, 동적 처리 기능(함수) 등을 컴포넌트화하여 제공합니다.

※ Component란 UI를 구성하는 각각의 부품들이라고 생각하면 됩니다. UI상에 같은 부분을 재사용하고 싶은 경우 component화 해서 재사용할 수 있는 것이 component의 장점입니다. 대부분의 애플리케이션들이 트리 형태의 component들로 구성되어 있습니다.



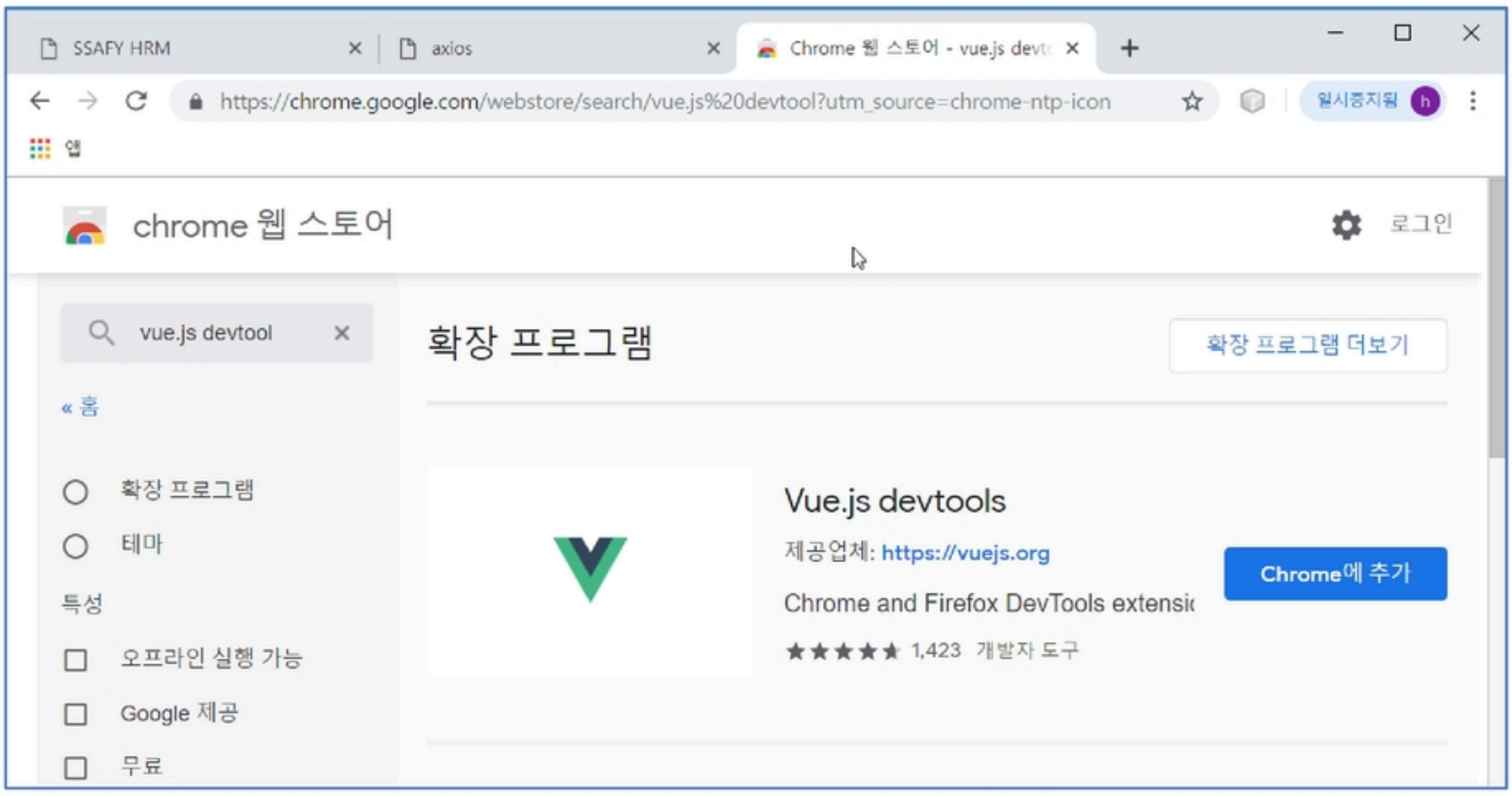
예를 들면, 웹 페이지의 헤더, 사이드 바, 본문에 해당되는 component를 만들 수도 있고, 헤더부분은 네비게이션 링크들, 본문 부분은 포스트들 등 각각의 component들은 또 다른 component들로 이루어집니다.

Vue 를 사용하고 편집을 원활하게 하기 위해 개발자 도구를 설치 합니다.
크롬 브라우저에서 [Control+Shift+I] 혹은 F12 를 입력하면 개발자도구를 사용할 수 있습니다.
개발자도구는 부모태그와 자식태그의 계층구조인 DOM 구조와 여백, 테두리, 패딩 및 실제 내용으로 구성되는 CSS 박스모델을 쉽게 볼 수 있도록 해줍니다.



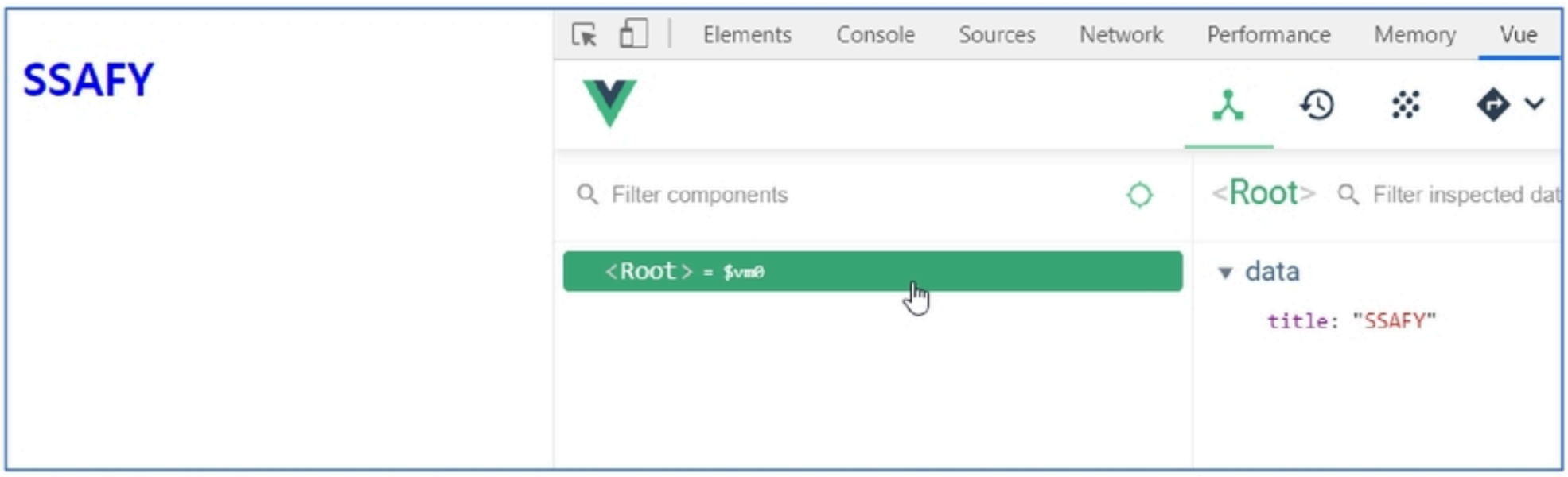
[그림 1.1] 크롬의 개발자도구 화면

크롬의 웹 스토어에서 Vue.js devtools 를 크롬에 추가합니다. Vue.js devtools 를 선택하여 크롬에 추가합니다.



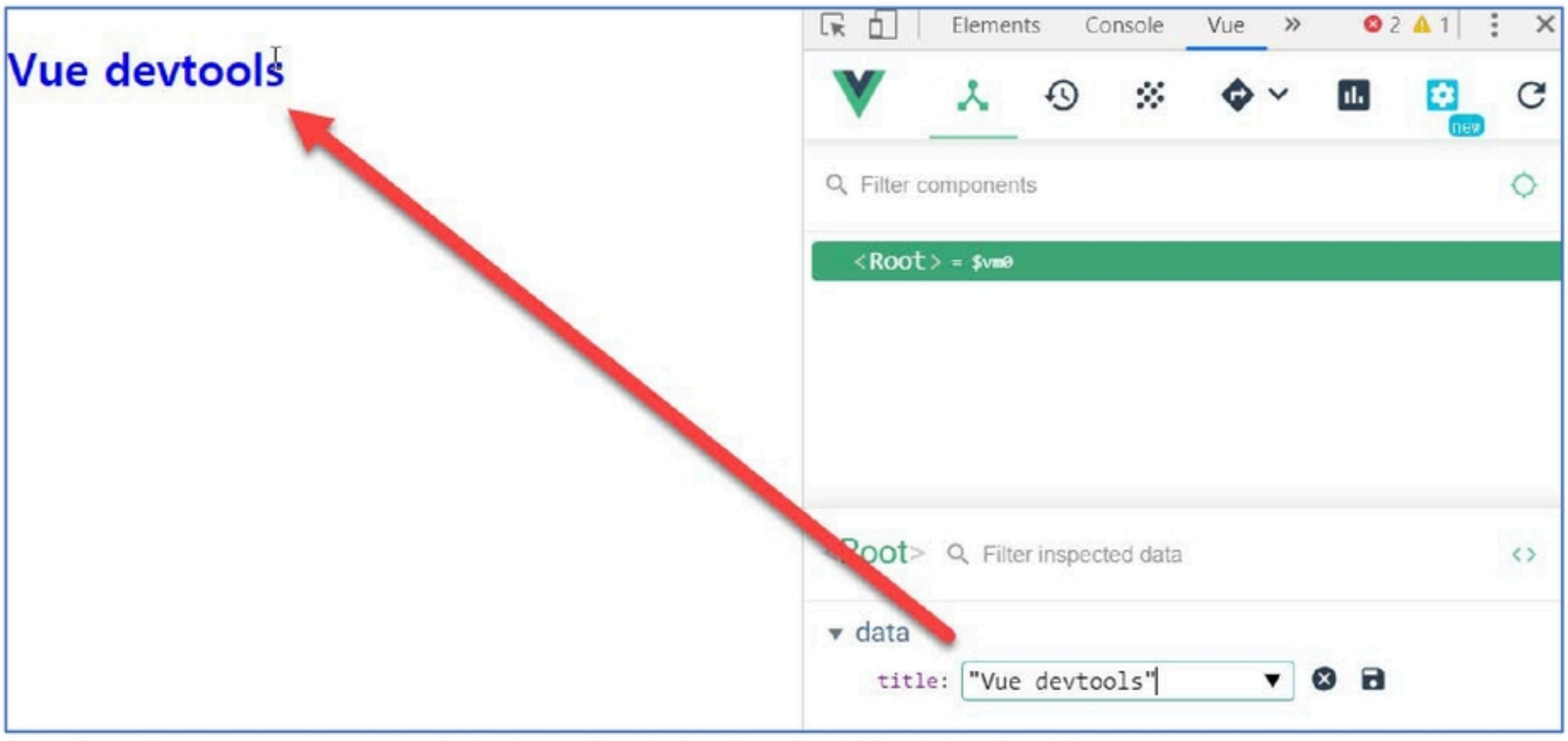
[그림 1.2] 웹 스토어에서 Vue.js devtools 를 크롬에 추가하기 위한 화면

Vue.js devtools 는 Vue 의 DOM 구조를 보여줍니다. 모든 컴포넌트가 <root>로 시작하는 트리 형태로 나타납니다. 그리고 연관된 data 인 Model 도 보여줍니다.



[그림 1.3] Vue.js devtools 실행 화면

Vue.js devtools 에서 data 를 변경했을 때 화면의 값도 변경되는 것을 확인할 수 있습니다. 이것을 이용하여 모델(Model)의 값을 변경하면 뷰(View)에도 반영되는 것을 테스트 할 수 있습니다.



[그림 1.4] Vue.js devtools 에서 data 를 변경했을 때 뷰에 반영되는 화면

1.1.2 이클립스와 톰캣

Vue.js 는 Front-End 이므로 브라우저만 있으면 됩니다.
스프링과 스프링 부트 등 자바 서버환경과 연동을 하거나 AJAX 와 연동 등 서버에서 제공하는 서비스를 더 효과적으로 사용하기 위하여 톰캣 서버를 이용하겠습니다.

1.2 개발순서

여기서는 간단한 개발순서를 통해서 Vue 의 특징을 알아봅니다. Vue 를 이해하고 쉽게 사용할 수 있게 되면 자동 개발 툴을 사용하는 것이 좋습니다.

1.2.1 Vue.js 라이브러리 얻기

Vue.js 라이브러리를 얻는 방법은 라이브러리를 다운로드하는 방법과 CDN 으로 라이브러리를 자동(default)으로 포함시키는 방법이 있습니다.
여기서는 간단하게 CDN 으로 라이브러리를 자동으로 포함시키기 위해 <script>태그를 이용하겠습니다.
https://unpkg.com/vue 는 npm 에 올라간 최신 버전을 반영합니다.
Vue 라이브러리는 단일 JavaScript 파일이므로 HTML <script>태그로 감싸야 합니다.

[소스-index01.html]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SSAFY</title>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
  </body>
</html>
```

1.2.2 모델(데이터 값)만들기

데이터의 값을 의미하는 모델을 만듭니다. 뷰에서는 일반적으로 {key:value} 형태를 사용하는 JSON 방법으로 데이터를 만듭니다. value 에 다시 {}나 배열인 []를 넣어서 XML 같은 부모 자식 엘리먼트 효과를 만듭니다.

데이터와 화면을 분리시켜서 데이터만 뒤에서 주고받는 비동기에서는 XML 과 JSON 을 주로 사용합니다. JSON 은 .(마침표) 또는 ["객체명"]을 이용하여 계층구조를 추적할 수 있고 객체도 표현할 수 있습니다.

[표 1-1]에서 nationalflags[0].name 으로 "Ghana"를 얻을 수 있습니다. []은 배열이고 0 은 배열의 첫번째 요소의 주소를 뜻합니다.

[표 1-1] 데이터 종류와 표현 방법

데이터 종류	데이터 표현 방법
XML	<?xml version='1.0' encoding='utf-8' ?> <nationalflags> <nationalflag> <name>Ghana</name> <shortname>GH</shortname> <code>288</code> <flag> https://flag_of_ghana.svg/44px-flag_of_ghana.svg.png </flag> <korname>가나</korname> </nationalflag> </nationalflags>
JSON	{"nationalflags":[{ "name":"Ghana", "shortname":"GH", "code":"288", "flag":"https://flag_of_ghana.svg/44px-flag_of_ghana.svg.png", "korname":"가나"}]}

[소스-index02.html]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SSAFY</title>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <script type="text/javascript">
```

```
        var model = {title : 'SSAFY'};
      </script>
    </body>
</html>
```

1.2.3 뷰 만들기

HTML 태그의 속성이나 컨텐트(value)에 모델(데이터)를 입력합니다.

{{ }}는 콧수염 표현식(Mustache Expression)이라는 문자열 보간(interpolation)법으로 브라우저가 HTML 을 렌더링할 때 Vue 의 모델이라는 것을 알려주어서 반영시키는 선언적인 템플릿 표현식입니다. 렌더링이란 계층적(부모 자식 관계가 있음)이면서 논리적인 문서의 표현식을 그래픽 표현식으로 변형시키는 과정입니다. 렌더링은 계층적으로 되는데 DOM 과 박스모델인 스타일을 기반으로 레이아웃이 준비되고 엘리먼트(요소-태그)을 찾아서 화면에 보입니다.
그러므로 브라우저는 {{}}을 만나면 그 안의 내용을 모델(데이터) 객체의 속성 값을 반영해서 결과화면을 보입니다. 단, 콧수염 표현식 {{}}은 모델(데이터)을 HTML 이 아닌 일반 텍스트로 해석합니다. HTML 을 출력하려면, 뒤에서 배울 v-html 디렉티브를 사용해야 합니다.

템플릿이란?
템플릿이란 뷰로 화면을 조작하기 위해 제공되는 문법입니다. 뷰 인스턴스에서 관리하는 데이터를 화면에 연결하는 데이터 바인딩과 화면의 조작을 편하게 할 수 있는 디렉티브로 나뉩니다.

[소스-index03.html]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SSAFY</title>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <div id='first'>
      <h2>{{title}}</h2>
```

```
    </div>
    <script type="text/javascript">
      var model = {title : 'SSAFY'};
    </script>
  </body>
</html>
```

1.2.4 뷰 모델 만들기

뷰 모델(View Model)은 모델(Model)과 뷰(View)사이에서 관찰자 및 중재자 역할을 합니다. 뷰 모델이 없다면 뷰에 이벤트가 발생해도 모델의 변함이 없고, 모델이 변경되어도 뷰에 반영되지 않습니다. 프런트 엔드의 MVVM은 백엔드의 MVC와 거의 비슷합니다.

MVC의 컨트롤러(Controller)가 MVVM 에서는 뷰 모델(View Model)의 역할을 합니다. 프런트 엔드는 여러 레이어간의 화면흐름과 데이터 흐름을 제어하는 것이 아니라 화면 단만의 데이터(모델) 변화를 화면(뷰)에 반영시키는 것입니다.

[소스-index04.html]에서 first라는 Vue 객체가 뷰 모델(View Model) 입니다. Vue의 생성된 객체 first 인스턴스는 모델과 뷰를 지켜보면서 모델의 데이터 변화를 뷰에 반영하도록 합니다.

이를 위해서는 Vue 인스턴스를 DOM 요소에 마운트 해야됩니다. "마운트"는 기존 DOM 요소를 Vue.js가 생성하는 DOM 요소로 치환하는 것을 말합니다. 마운트를 하는 방법은 2가지가 있습니다.

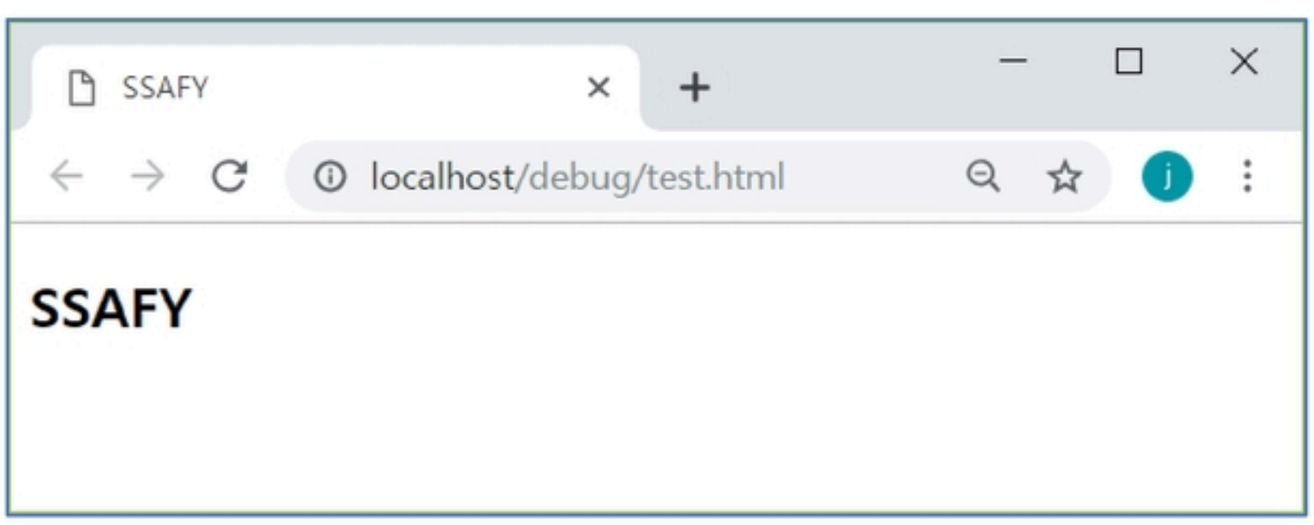
(1) 첫 번째는 $mount 메서드를 호출해서 마운트 해주는 방법이고, (2) 두 번째는 Vue 인스턴스의 옵션 객체의 "el" 속성에 마운트 대상을 지정해주는 방법입니다. "el"은 element를 의미하며, "el"에 등록한 "#first"는 CSS 셀렉터를 의미합니다. 결국 **el : '#first'**는 id가 first인 태그 내 HTML 태그(엘리먼트)들을 관찰대상(도메인)으로 보게 됩니다. data는 view model 처럼 model의 {title : 'SSAFY'};를 참조합니다. (만약 .first 처럼 여러 개의 요소에 동시에 들어갈 수 있는 class 셀렉터를 사용할 경우 가장 먼저 나오는 요소에 적용됩니다)

결론적으로 뷰 모델은 인 뷰 내부에 {{}}등 여러 보간법을 이용하여 뷰에 모델 변화를 반영하도록 합니다. {{title}}은 관찰대상(도메인)인 내부 에서만 모델인 {title : 'SSAFY'}의 값인 'SSAFY'를 반영하기 때문에 **도메인 특화 데이터**라고 하며 **SSAFY**로 렌더링됩니다.

[소스-index04.html]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SSAFY</title>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <div id='first'>
      <h2>{{title}}</h2>
    </div>
    <script type="text/javascript">
      var model = {title : 'SSAFY'};
      var first = new Vue({
        el : '#first',
        data : model
      });
    </script>
  </body>
</html>
```

실행결과는 다음과 같습니다.



1.2.5 속성 보간법 사용하기(디렉티브)

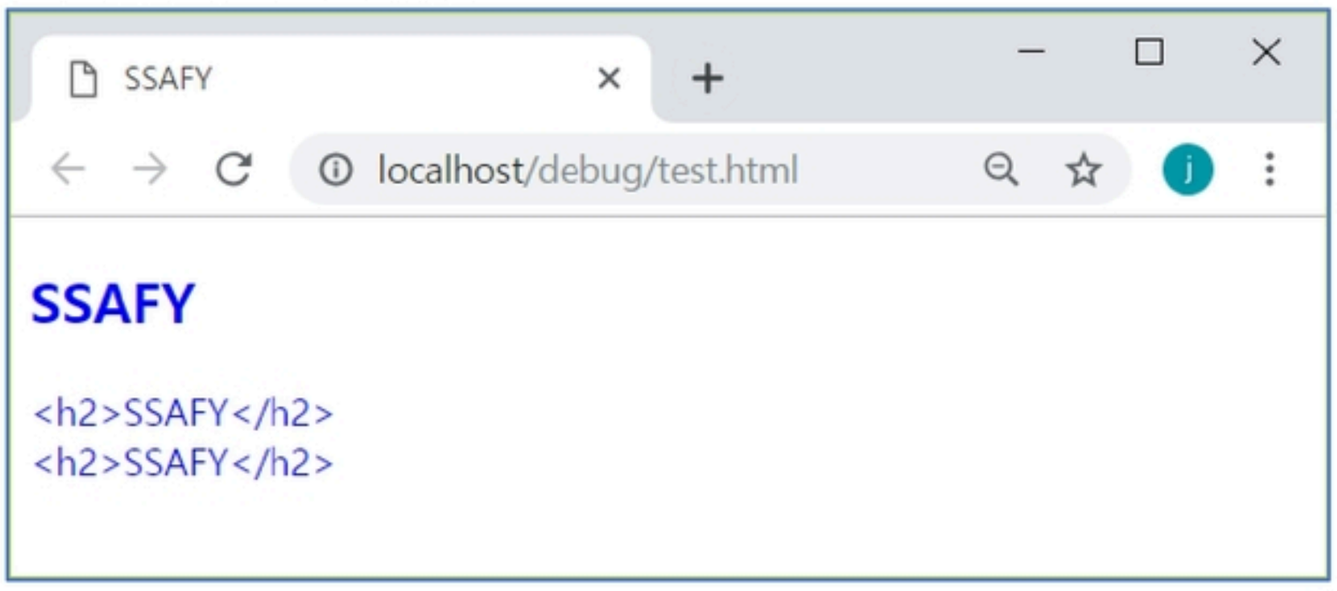
{{ }}은 data 에 선언했던 model 객체의 데이터를 참조합니다. 그러므로 {{title}}은 SSAFY 가 됩니다.
속성이란 엘리먼트 옆에 선언한 것으로 “ ” 나 ‘ ’ 로 속성 값을 감싸야 합니다.
 에서 div 엘리먼트의 v-html 속성이 됩니다. 그래서 속성 보간법이라 하고, 이를 “디렉티브”라고도 합니다. 콧수염 표현식 {{}}은 데이터를 HTML 이 아닌 일반 텍스트로 해석합니다.

{{title}} 은 <h2>SSAFY</h2> 이 되는데 파싱이 되지 않아 <h2>SSAFY</h2>가 그대로 브라우저에 보입니다. 같은 원리로 > 도 파싱되지 않고 <h2>SSAFY</h2>가 그대로 브라우저에 보입니다.
> 은 파싱되어 SSAFY 가 h2 크기로 브라우저에 반영되어 보입니다.
여기까지 했다면 CSS 스타일을 반영해 보세요.
*JQuery 의 text()와 Vue 의 v-text 는 같은 역할을 합니다. JQuery 의 html()와 Vue 의 v-html 는 같은 역할을 합니다.

[소스-index05.html]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SSAFY</title>
    <style type="text/css">
      #first div {color: blue ;}
    </style>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <div id='first'>
      <div v-html="title"></div>
      <div v-text="title"></div>
      <div>{{title}}</div>
    </div>
    <script type="text/javascript">
      var model = {title : '<h2>SSAFY</h2>'};
      var first = new Vue({
        el : '#first',
        data : model
      });
    </script>
  </body>
</html>
```

실행 결과는 다음과 같습니다.



1.2.6 문자열 보간법 {{}}과 메서드

Vue 인 뷰 모델에는 data 와 함께 data 를 처리하는 메서드를 선언할 수 있습니다.
메서드를 호출하려면 {{메서드()}}와 같이 보간법 사이에 메서드를 입력하면 됩니다. 메서드에 아규먼트가 있다면 {{메서드(아규먼트)}}와 같이 메서드의 ()에 아규먼트를 입력합니다.

메서드를 선언할 때 이벤트 객체를 사용하는 경우는 (1) 이벤트 객체만 아규먼트로 사용하는 경우와 (2) 다른 아규먼트와 같이 사용하는 경우가 있습니다.

(1) 이벤트 객체만 아규먼트로 사용하는 경우 메서드를 선언할 때는 아규먼트의 이름을 임의로 설정해서 이벤트 객체로서 사용하고, 호출할 때는 아규먼트를 생략해도 됩니다.

(2) 이벤트 객체를 다른 아규먼트와 같이 사용하는 경우 메서드를 선언할 때는 다른 아규먼트와 같이 이름을 임의로 설정해서 사용하고, 호출할 때는 이벤트 객체를 $event 라고 써서 아규먼트로 넘겨줍니다.

[소스-index6_1.html]

```
<div id="first">
    <div>{{output()}}</div> <!—output1 메서드 호출 -->
    <div>{{output($event)}}</div> <!—output1 메서드 호출 -->
    <div>{{output2(10,$event)}}</div> <!—output2 메서드 호출 -->
</div>
<script type="text/javascript">
    var first = new Vue({
        el: "#first",
        data: {
        },
        methods: {
          output: function(event){ // (1) 이벤트 객체만을 사용할 때
                return "hello";
            },
          output2: function(number,event){ // (2)다른 아규먼트와 함께 이벤트 객체를 사용
                return number+"World";
            }
        }
    });
```

```
    Vue.config.devtools = true;
    // devtools 에 보인다.  디버그용
</script>
```

클릭 이벤트가 발생했을 때 메서드가 실행되게 할 수 있습니다. v-on:click='메서드이름'을 이용합니다.
뷰 모델에 data 로 등록된 title 를 사용하기 위해서 메서드 내부에서 this.title 을 사용합니다. 뷰 모델에 data 로 등록된 title 를 this.title 을 이용하여 수정하면 수정된 내용이 뷰에 반영됩니다.
메서드를 선언할 때 자바스크립트를 사용할 수 있습니다.

자바스크립트에서는 (1) 함수 표현식 var showHrm =function(){}과, (2) 함수 선언식 function showHrm(){} 두 방법으로 함수를 선언할 수 있습니다.

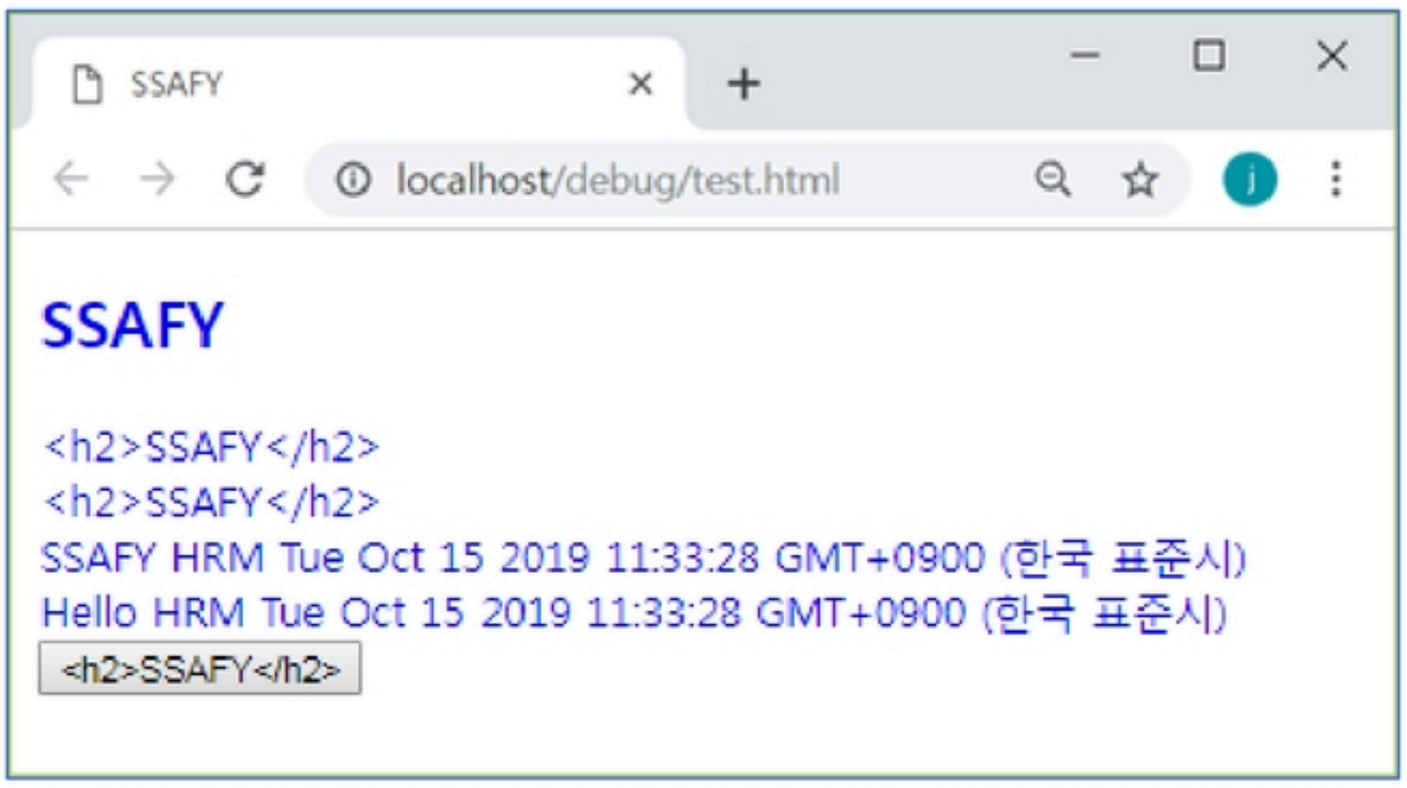
Vue 에서는 함수이름:function(){} 형식을 사용합니다.
화면(View)의 {{showHrm()}}는 뷰 모델에 선언된 showHrm:function()을 호출합니다.
{{showMyName('Hello HRM')}}는 뷰 모델에 선언된 showMyName: function (name, $event)를 호출합니다.
event 객체와 다른 아규먼트를 구별하기 위해 $를 반드시 붙여야 합니다.
뷰 모델에서는 this.변수를 이용하여 모델에 선언한 변수를 사용(get/set) 할 수 있습니다.
화면(뷰,View)의 {{title}} 에서 div 태그에 onclick 이벤트가 발생하면 changeMyName() 함수를 호출합니다. 이때 뷰 모델에 선언된 changeMyName 함수가 실행되어 this.title 에 문자열이 변경됩니다.
그리고 변경된 title 은 화면의 {{title}}부분에서 변경된 문자열로 출력됩니다.
정리하면 뷰 모델의 함수에서 this.title='SSAFY HRM SSAFY HRM'로 수정한 문자열이 {{title}}에 반영되어 SSAFY HRM SSAFY HRM 에 화면에 보입니다.

[소스-index06_2.html]

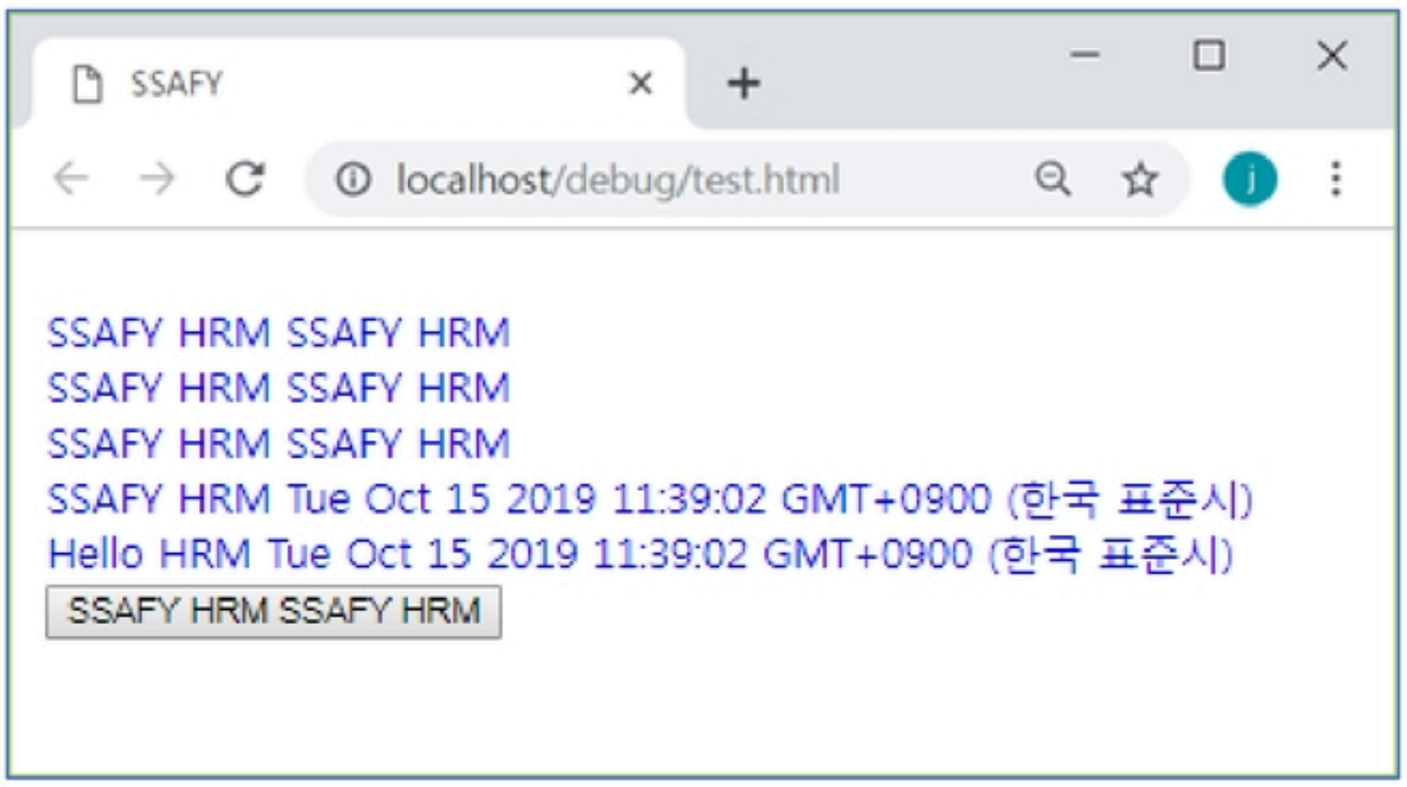
```
...
<div id='first'>
    <div v-html="title">
    <div v-text="title">
    <div>{{title}}
    <div>{{showHrm()}}
    <div>{{showMyName('Hello HRM')}}
    <button v-on:click="changeMyName">{{title}}
```

```
</div>
<script type="text/javascript">
  var model = {title : '<h2>SSAFY</h2>'};
  var first = new Vue({
        el : '#first',
        data : model,
        methods: {
            showHrm : function() {
                return " SSAFY HRM    "+new Date();
            },
            showMyName : function(name, $event) {
                 return name+' '+new Date();
            },
            changeMyName : function() {
             this.title='SSAFY HRM SSAFY HRM';
            }
       }
 });
  Vue.config.devtools = true;   // devtools에 보인다. 디버그용
</script>
...
```

실행 결과는 다음과 같습니다.



마지막 라인의 버튼 클릭하면 이벤트가 적용되어 title 의 값이 모두 변경됩니다.



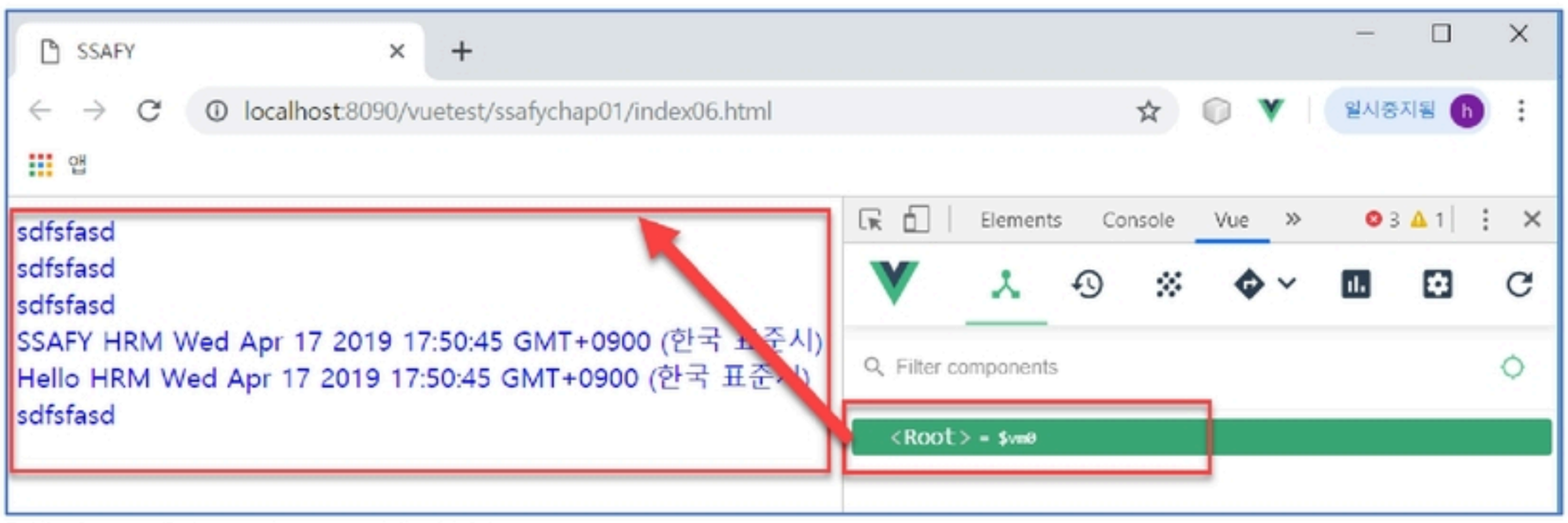
comment-버튼으로 변경되었기 때문에 사진과 글을 변경

2. Vue인스턴스

2.1 Vue 인스턴스 생성

2.1.1 Root Vue 인스턴스

자바에서 객체를 생성해야 사용할 수 있듯이 뷰 모델을 준비하기 위해서는 Vue 인스턴스를 생성해야 합니다. new Vue({})는 [그림 2.1]과 같이 Root 인 Vue 인스턴스를 만듭니다. Vue 인스턴스인 뷰 모델(View Model)에 el 을 통해서 뷰(View)의 관찰대상(관찰범위-도메인)과 data 을 이용하여 Model 을 선언합니다.
이 과정을 통해서 뷰 모델은 View 와 Model 을 참조할 수 있습니다. Model 은 el 에서 선언한 태그내에서만 사용할 수 있기 때문에 "**도메인 특화 데이터**"라 합니다. el 은 DOM 엘리먼트의 CSS 셀렉터를 의미합니다.



[그림 2.1] Root Vue 인스턴스

Vue 앱은 new Vue 명령을 통해 만들어진 Root Vue 인스턴스로 구성되며, 선택적으로 중첩이 가능하고 재사용 가능한 컴포넌트 트리로 구성됩니다. 예를 들어, Todo 앱의 컴포넌트 트리는 다음과 같습니다. (참고: https://kr.vuejs.org/v2/examples/todomvc.html)

```
Root Instance
└── TodoList
    ├── TodoItem
    │   ├── DeleteTodoButton
    │   └── EditTodoButton
    └── TodoListFooter
        ├── ClearTodosButton
        └── TodoListStatistics
```

Vue 인스턴스를 인스턴스화 할 때는 데이터(data), 템플릿(template), 마운트할 엘리먼트(el), 메소드(methods), 라이프사이클 콜백(후크) 등의 옵션을 포함할 수 있는 options 객체를 전달해야합니다. 명령어 형식은
new Vue({ options 객체 })

입니다.

먼저 간단하게 기본적인 Options 객체인 el 이 있습니다. el 옵션은 Vue 인스턴스에게 마운트 할 DOM 엘리먼트를 알려주는 역할을 합니다. 입력값은 CSS 셀렉터 문자열이나 실제 HTML 엘리먼트를 넣어줄 수 있습니다.

el : '#first' 는 Vue 인스턴스에게 id 가 first 인 DOM 엘리먼트에 마운트를 하라고 알려줍니다. Vue 인스턴스는 아래 소스와 같이 el, data, methods 등으로 구성됩니다.

Vue 생성자인 Vue( )에 {el:#id, data:model, methods:{ }} JSON 형식으로 옵션(option) 객체를 입력해 줌으로써 인스턴스를 생성할 수 있습니다.

화면(View)에서 {{ showHrm() }}와 같이 함수를 호출할 수 있습니다. 함수에 아규먼트가 있다면 {{ showMyName('Hello HRM') }}와 같이 아규먼트를 대입할 수 있습니다.

또한 이벤트를 이용하여 호출할 수도 있습니다. v-on:click = '함수이름'의 디렉티브 형식을 많이 사용합니다. **v-on:click = '함수이름'** 을 **@click = '함수이름'** 로 사용할 수 있습니다.

[소스-index01.html]

```
<div id='first'>
    <div v-html="title"></div>
    <div v-text="title"></div>
    <div>{{title}}</div>
    <div>{{showHrm()}}</div>
    <div>{{showMyName('Hello HRM')}}</div>
    <div v-on:click="changeMyName">{{title}}</div>
    <div @click="changeMyName">{{title | toH3 }}</div>
</div>
<script type="text/javascript">
  var model = {title : '<h2>SSAFY</h2>'};
  var first = new Vue({
        el : '#first',
        data : model,
        methods: {
            showHrm : function() {
                return " SSAFY HRM   "+new Date();
            },
            showMyName : function(name, $event) {
                 return name+' '+new Date();
            },
```

```
            changeMyName : function() {
             this.title='SSAFY HRM SSAFY HRM';
            }
        }
    });
  Vue.config.devtools = true; // devtools에 보인다.
</script>
```

2.2 Vue 인스턴스 옵션

아래는 Vue의 기본 옵션들입니다. 자바에서 속성들과 메서드로 보면 이해하기 쉽습니다. 하나 하나가 많은 내용이며 뒤에서 다루게 됩니다.
el 옵션은 Vue 인스턴스에게 마운트 할 DOM 엘리먼트를 알려주는 역할을 합니다. 입력 값은 CSS 셀렉터 문자열이나 실제 HTML 엘리먼트를 넣어줄 수 있습니다. 인스턴스 생성시에 el 옵션이 입력되어 있으면, 즉시 인스턴스의 마운트가 시작됩니다. 그렇지 않은 경우에는, 인스턴스이름.$mount() 명령을 호출해서 직접 마운트를 시켜주어야 합니다. 명령어의 입력에는 HTML 엘리먼트를 넣어줍니다. 인스턴스가 마운트 된 후에는, 마운트 된 엘리먼트를 **인스턴스이름.$el**을 사용해서 조작할 수 있습니다.

template (string)은 Vue 인스턴스의 화면 표시(마크업)로 사용할 문자열 템플릿입니다. 이 template이 인스턴스가 마운트되어있는 엘리먼트의 내용을 대체하게 됩니다. 원래 엘리먼트에 존재하던 내용은 무시됩니다. 만약 template의 입력한 문자열이 #로 시작하면 querySelector처럼 동작하며 그렇게 찾아낸 엘리먼트의 내용을 템플릿의 내용으로써 사용합니다.

data (Object | Function)는 화면에 출력하기 위한 data들을 정해주는 option 입니다. 입력 값은 사용할 data 객체를 반환하는 함수형이거나 사용할 data 객체 그 자체입니다. Vue에서 이 입력 값들을 getter / setter로 변환하여 반응형으로 사용할 수 있도록 만듭니다.

props (Array<string> | Object)는 부모 컴포넌트로부터 전달 받은 property들의 배열 혹은 객체입니다.
filters ([key: string]: Function)는 Vue 인스턴스에서 사용되는 필터입니다.
methods ([key: string]: Function)는 Vue 인스턴스에서 사용되는 method입니다.
computed ([key: string]: Function | { get: Function, set: Function }) 자동으로 this 컨텍스트를 Vue 인스턴스에 바인딩 합니다. 다시 말해 this. 로 Vue인스턴스의 속성과 메서드에 접근할 수 있습니다.
watch ([key: string]: string | Function | Object | Array)는 감시 속성으로 Vue 인스턴스의 데이터가 변경되는 시점에 자동 실행됩니다.

```
var 인스턴스이름= new Vue({
    el: '#app',
    template: '<div>HTML 태그를 이용하여 화면- 화면 템플릿</div>',
    data: {},
    props: [],
    methods: {},
    filters: {}
```

```
    computed: {},
    watch: {}
});
```

2.2.1 Vue 인스턴스의 filter

출력(화면에 보이기-텍스트 형식화)할 때 사용하는 {{ }}내에 문자열을 위해 |을 사용할 수 있습니다. 출력할 때 |의 왼쪽 data를 가공해서 목적에 알맞은 문자열로 변환 혹은 원하는 형식으로 변환하는데 사용합니다.

{{ title | toH3 }}는 뷰 모델의 data인 title을 toH3의 아규먼트로 입력하여 가공한 후 문자열로 출력하라는 의미입니다. 뷰 모델의 data인 title이 뷰 모델의 filter에 선언한 toH3(value)의 아규먼트 value로 toH3에 입력됩니다.

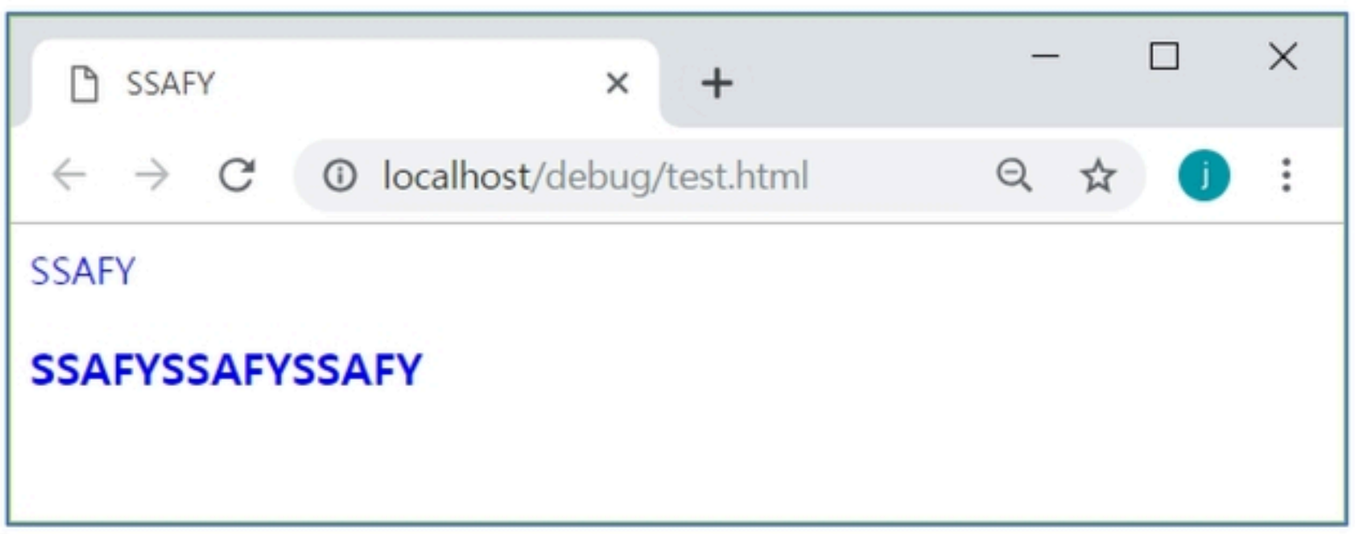
[소스-index02.html]에서 {{ title | toH3 }}은 toH3(title) 형식으로 실행되어 실행된 후 결과를 문자열로 출력합니다.

[소스-index02.html]

```
<div id='first'>
    <div v-html="title"></div>
    <div><h3>{{ title | toH3 }}</h3></div>
</div>
<script type="text/javascript">
  var model = {title : 'SSAFY'};
  var first = new Vue({
        el : '#first',
        data : model,
        filters: {
              toH3 (value) {
                    var s="";
                    for(i=0; i<3; i++){
                          s+=value;
```

```
                    }
                    return s;
                }
        }
    });
  Vue.config.devtools = true;   //devtools 디버거용
</script>
```

실행결과는 다음과 같습니다.



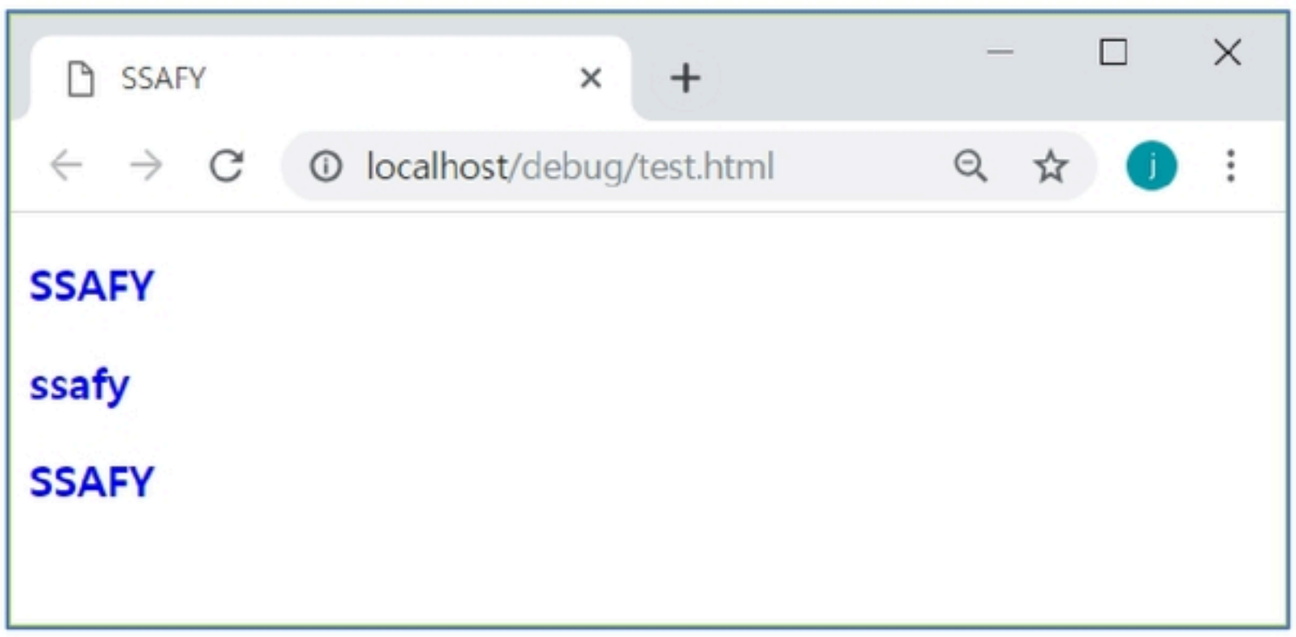
{{ title | toLower }}는 toLower(title) 필터를 실행시킵니다.
같은 방법으로 {{ title | toUpper }}는 toUpper(title)를 실행시킵니다.
value.toLowerCase()는 자바스크립트의 String의 문자열 처리 함수입니다. 모든 문자를 소문자로 만듭니다.
value.toUpperCase()도 같은 String 문자열 처리 함수이며, 모든 문자를 대문자로 만듭니다.

[소스-index03.html]

```
...
  <script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id='first'>
    <div><h3>{{title }}</h3></div>
    <div><h3>{{title | toLower }}</h3></div>
    <div><h3>{{title | toUpper }}</h3></div>
  </div>
```

```
<script type="text/javascript">
  var model = {title : 'SSAFY'};
  var first = new Vue({
    el : '#first',
    data : model,
    filters: {
      toUpper (value) {
        return value.toUpperCase();
      },
      toLower (value) {
        return value.toLowerCase();
      }
    }
  });
  Vue.config.devtools = true;
</script>
...
```

실행 결과는 다음과 같습니다.



2.2.2 Vue 인스턴스의 computed 옵션

computed 옵션은 뷰 모델의 data에서 선언한 모델 값을 원하는 작업을 거친 후 인스턴스에 저장해서 다른 곳에서 참조하고 싶을 때 사용합니다. 인스턴스 내에서는 data 객체와 computed 객체를 참조할 수 있는데, this(Vue 인스턴스 자신)를 거쳐서 참조해야 합니다.

computed 옵션은 캐싱 기능을 가지고 있는데, computed 옵션이 의존하고있는 데이터가 수정되기 전까지는 캐시 해두는 기능입니다. 같은 기능을 메서드로 구현해도 결과는 같습니다. 하지만 메서드는 캐싱 기능이 없어서 같은 내용을 실행해도 매번 계산을 다시 해야합니다. computed는 같은 내용을 실행할 경우 이전에 캐시해둔 값을 불러와서 재사용할 수 있습니다. 이런 특징 때문에 계산한 결과를 재사용하는 경우는 computed 옵션을 사용하는 것이 좋습니다.

method처럼 모델 값이 아니어도 사용가능하기 때문에 method와 결과만으로는 동일합니다. 모델 값의 문자열을 가공해서 원하는 문자열로 변환하는 filter와 모델 값을 이용하여 대소비교, 길이, 원하는 문자 찾기는 computed, 모델 값이거나 아니어도 사용할 수 있는 메서드 등이 있습니다.

[소스-index04.html]에서 v-if='hasResult'는 if(hasResult())처럼 true, false에 대하여 div 태그를 보이거나 안 보이도록 합니다. computed는 v-if='hasResult'와 같이 메서드가 아닌 프로퍼티형식을 사용합니다.

true일 때는 v-show와 v-if는 div 태그를 보여주기 때문에 결과가 동일합니다.
그러나 **false(this.title.length>10)일 때는 v-show는 display : 'none'이 되어 div 태그가 보이지 않고 v-if는 <--주석 -->는 주석이 되어 태그가 반영이 되지 않습니다.**

v-if 와 v-show 는 언제 사용하여야 할까요?

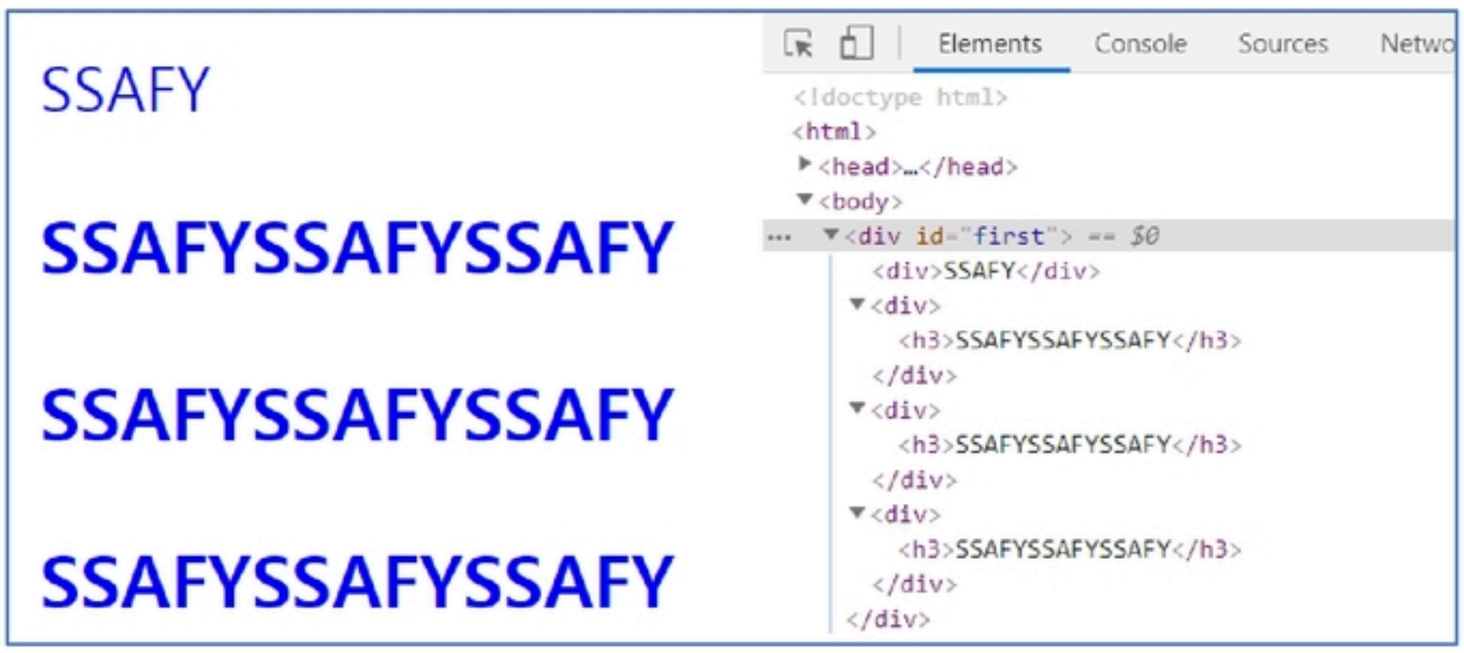
v-show는 조건의 여부와 관계없이 무조건 렌더링 되기 때문에 초기 렌더링 비용이 높습니다. 반면에 v-if는 조건에 따라 초기 렌더링 여부가 달라지므로 초기비용이 낮지만 토글 될 때마다 렌더링이 필요하므로 토글 비용이 높습니다. 따라서 조건이 자주 바뀐다면 v-show를, 조건이 자주 바뀌지 않으면 v-if를 권장합니다.

[소스-index04.html]

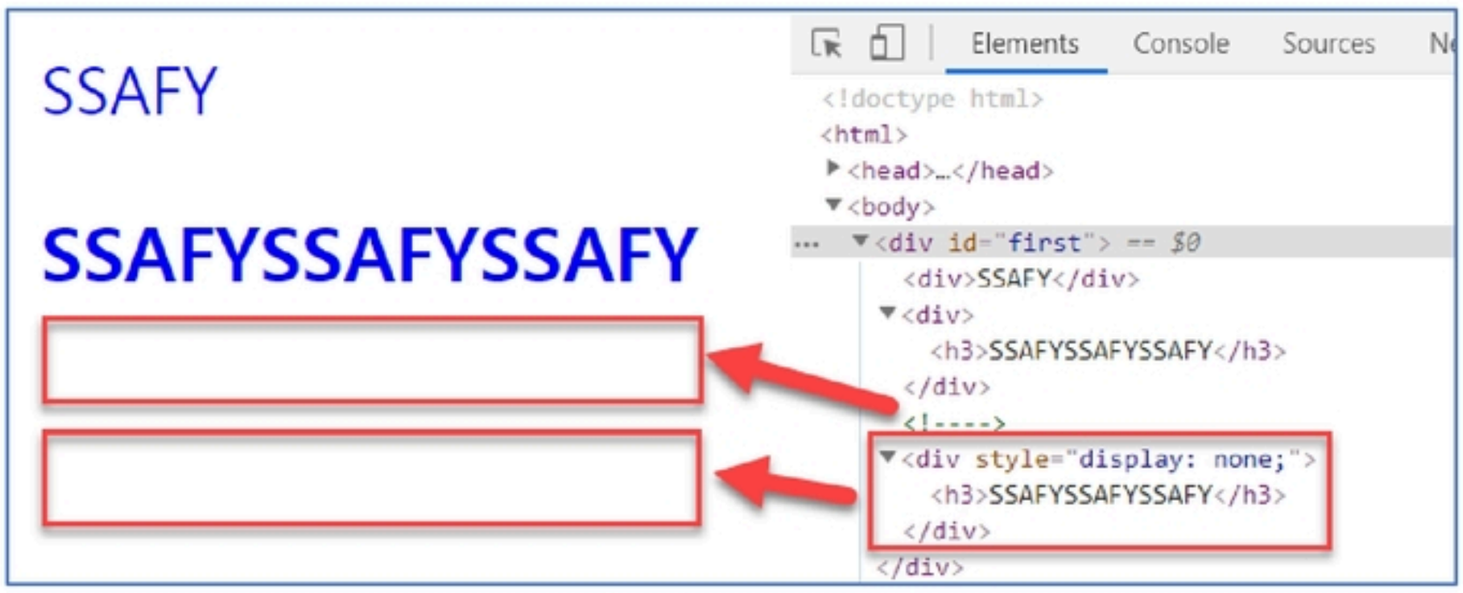
```
<div id='first'>
  <div v-html="title"></div>
  <div><h3>{{title | toH3 }}</h3></div>
  <div v-if='hasResult'><h3>{{title | toH3 }}</h3></div>
```

```
  <!--  false면  주석처리됨  -->
  <div v-show='hasResult'><h3>{{title | toH3 }}</h3></div>
  <!--  flase면  display:none이됨  -->
</div>
<script type="text/javascript">
  var model = {title : 'SSAFY'};
  var first = new Vue({
    el : '#first',
    data : model,
    filters: {
      toH3 (value) {
        var s="";
         for(i=0;i<3; i++){  s+=value;  }
         return s;
      }
    },
    computed: {
      hasResult: function () {
        return this.title.length>10;
      }
    }
  });
  Vue.config.devtools = true;
</script>
```

실행 결과는 다음과 같습니다.



[그림 2.2] v-if 와 v-show 가 true 일때



[그림 2.3] v-if 와 v-show 가 false 일때

2.2.3 Vue 인스턴스 data 속성 이용

Vue 인스턴스를 생성한 후 속성을 접근하려면 this를 사용합니다.

그러나 외부에서 속성에 접근하려면 인스턴스명.$data.속성을 이용합니다. 인스턴스명.속성으로도 사용 가능 합니다.

인스턴스명.메서드()를 호출하여 사용할 수 있습니다.

예제 [소스-index05.html] 에서 first.$data.title는 first.title과 동일하며 first Vue의 data 속성인 title 입니다.

자바에서 자신의 내부 속성(필드)를 사용할 때는 this를 이용하여, 외부에서는 생성된 객체의 레퍼런스.을 이용하는 것과 동일한 원리입니다.

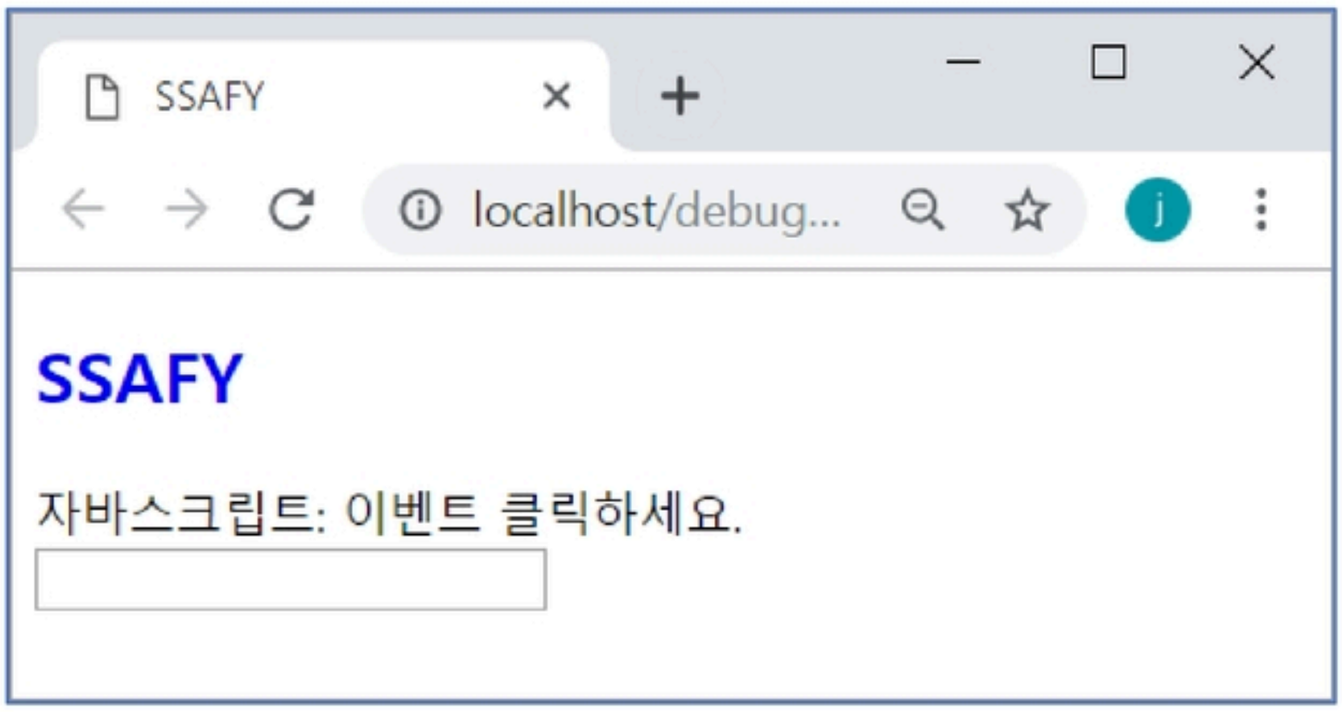
아규먼트로 받은 값을 first의 tittle에 대입할 수 있습니다.

[소스-index05.html]

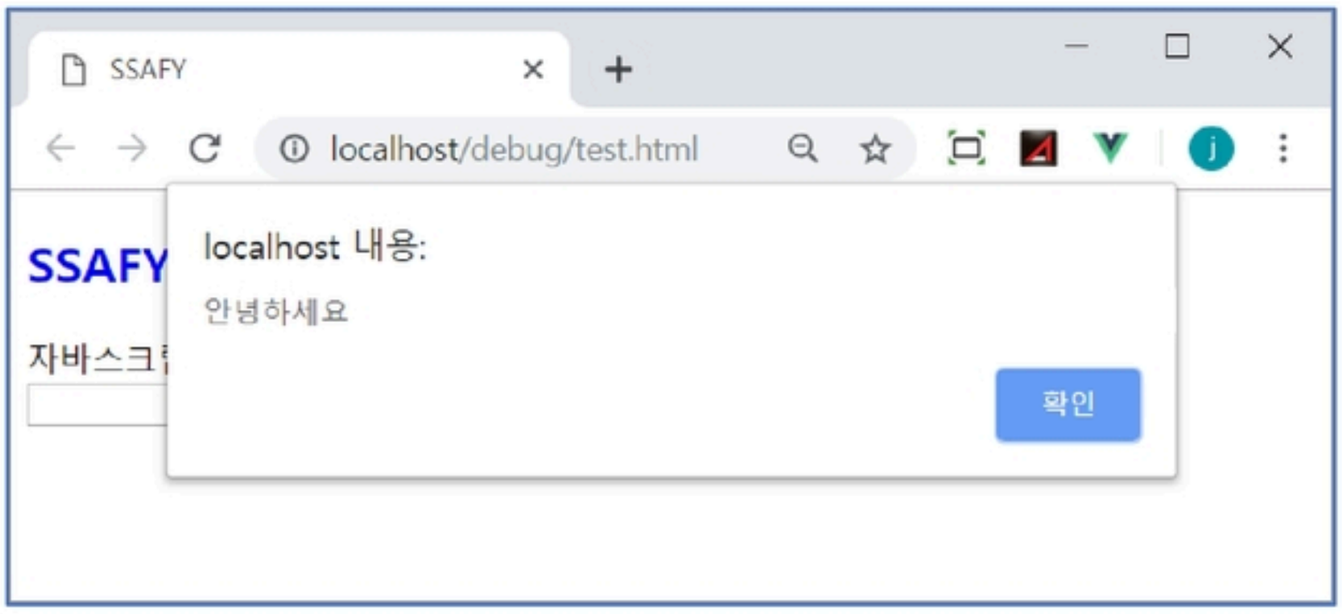
```
<body>
  <div id='first'>
    <div v-html="title"></div>
  </div>
  <div id='second'>
    <div onclick="hello('안녕하세요');">자바스크립트: 이벤트 클릭하세요.</div>
  </div>
  <input id='third' onclick="hello2(this);" />
  <script type="text/javascript">
    var model = {title : '<h2>SSAFY</h2>'};
    var first = new Vue({
      el : '#first',
      data : model
    });
```

```
    function hello(d){
      first.$data.title=d;
      alert(first.title);
    }
    function hello2(d){
      first.$data.title=d.value;
      if(d.value===document.getElementById('third').value){ //true
        alert(first.title);
      };
    }
    Vue.config.devtools = true;
  </script>
</body>
```

실행 결과는 다음과 같습니다.



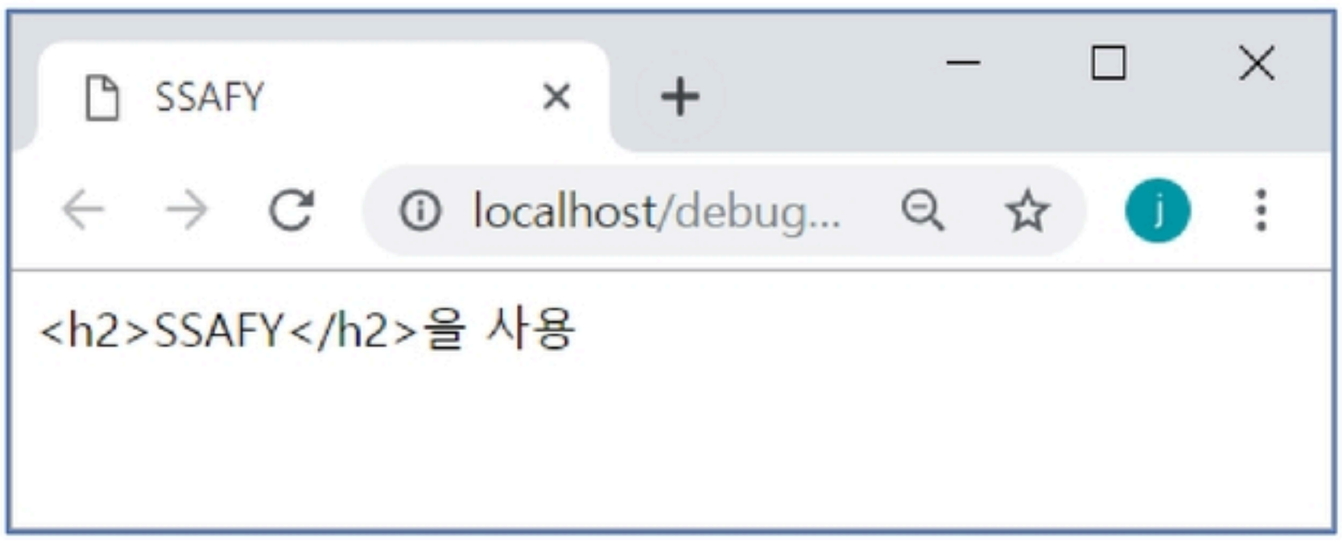
이벤트를 클릭하세요 라는 글자를 클릭하면



[소스-index06.html]

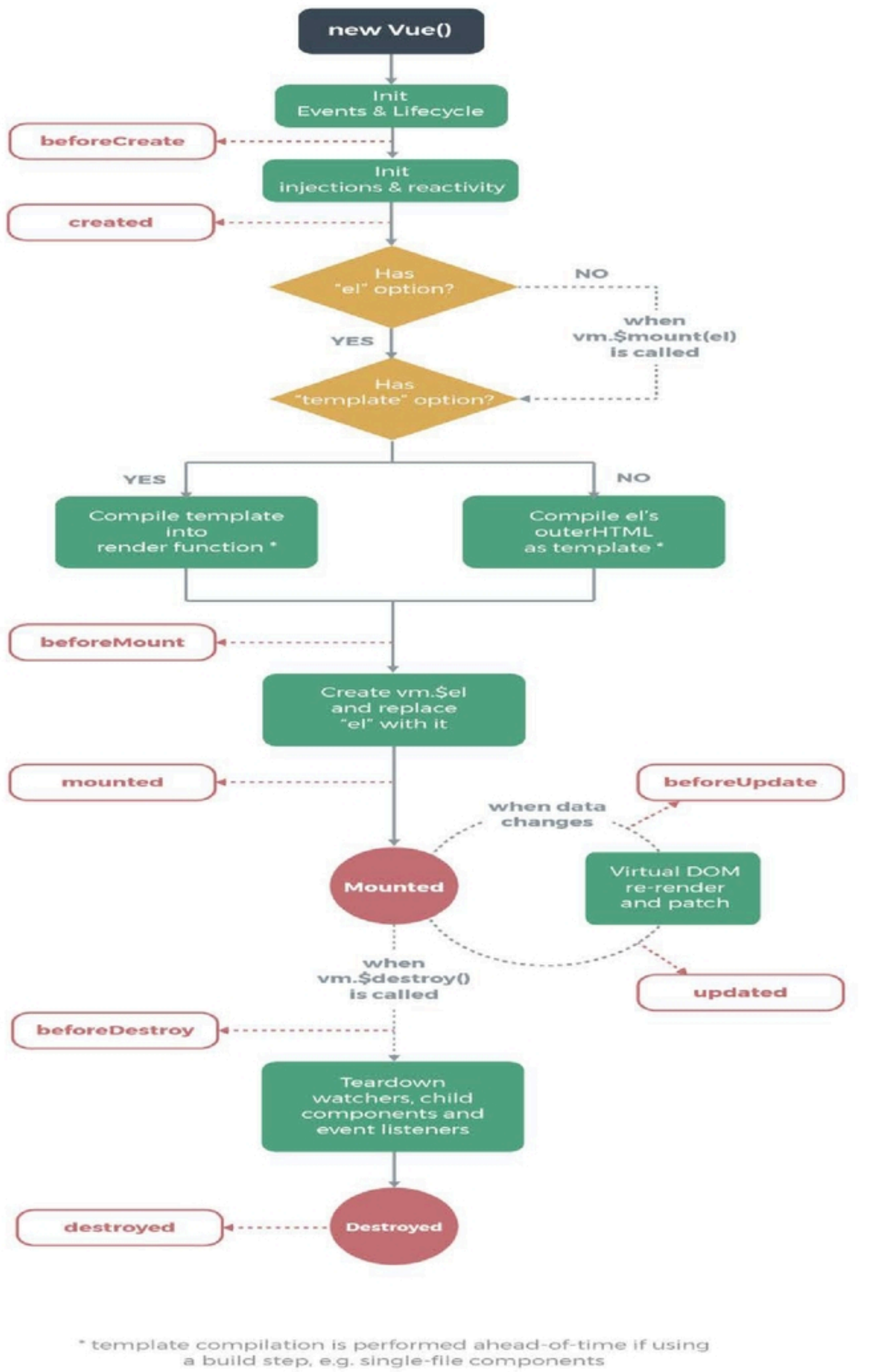
```
<div id='first'>
</div>
<div id='second'>
  <div>{{titleshow()}}</div>
</div>
<script type="text/javascript">
  var model = {title : '<h2>SSAFY</h2>'};
  var first = new Vue({
    el : '#first',
    data : model
  });
  var second = new Vue({
    el : '#second',
    methods: {
      titleshow : function() {
        return first.title+"을 사용";
      }
    }
  });
  Vue.config.devtools = true;
</script>
```

실행 결과는 다음과 같습니다.



2.2.4 Vue 인스턴스 라이프 사이클

서블릿의 라이프 사이클과 안드로이드의 라이프 사이클처럼 Vue도 라이프 사이클이 있습니다.
Vue도 같은 원리로 인스턴스가 생성(create)되고 옵션들이 준비(mount)된 후 수정(upate)되고 파괴되는(destroy) 과정을 밟습니다. 이 각 과정에 대하여 전후에 정해진 메서드들이 자동으로 실행되는데 이를 훅(후크-라이프 사이클 과정에 자동으로 걸려 실행됨)이라 한다. 아래 도표는 라이프 사이클이 발생하는 과정입니다.



[그림 2.5] 인스턴스의 라이프 사이클(https://kr.vuejs.org/v2/guide/instance.html 발췌)

Vue 인스턴스가 생성되기 전/후에 beforeCreate, created 가 호출됩니다.
Vue 인스턴스가 옵션들을 받아서 마운트되기 전/후 beforeMount, mounted 가 실행됩니다.
Vue 인스턴스의 data 가 변경되어 다시 렌더링하기 전/후 beforeUpdate, updated 가 실행됩니다.
Vue 인스턴스가 파괴되기 전/후에 beforeDestory, destory 가 실행됩니다.

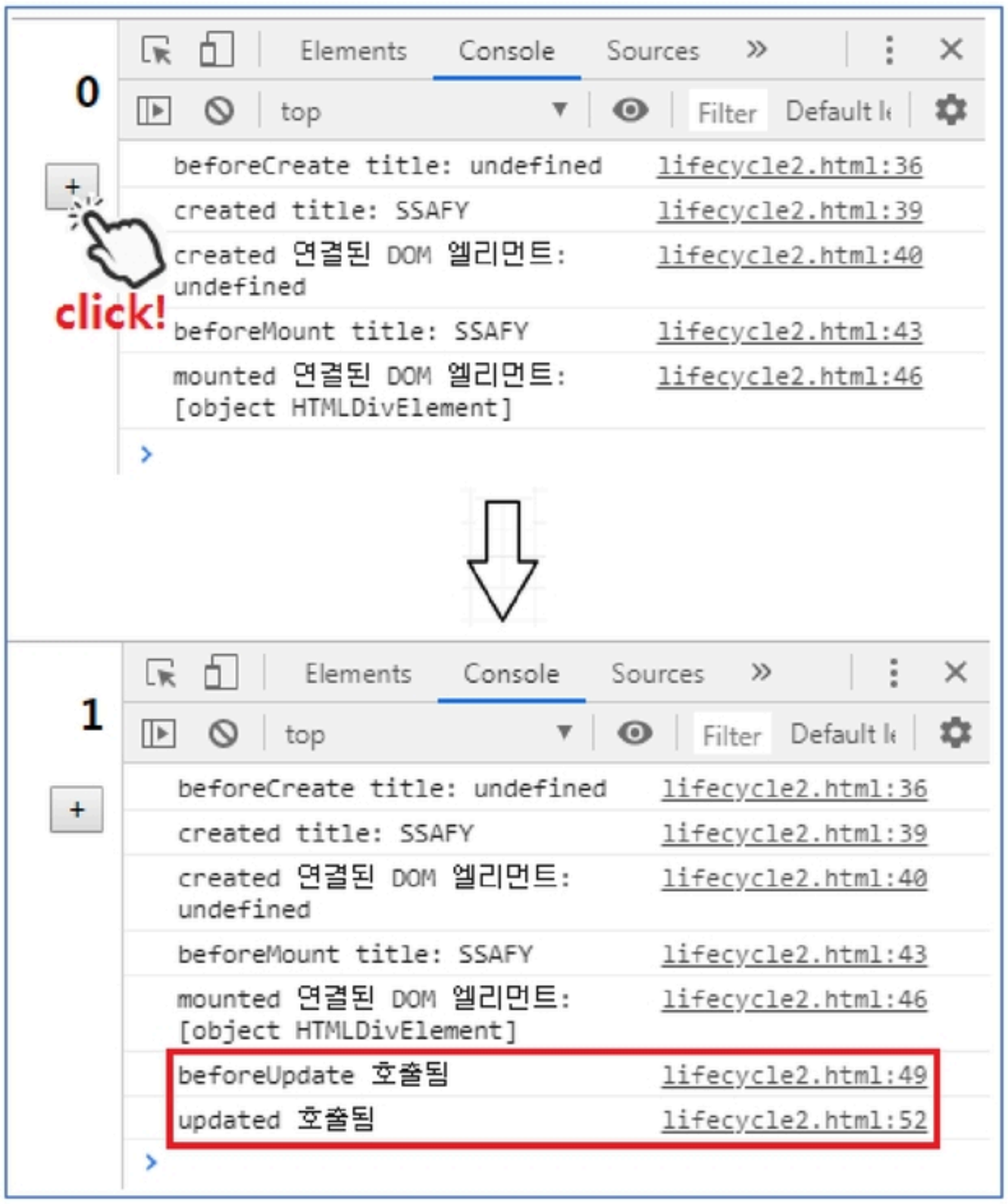
이런 메서드들은 Vue 인스턴스의 생성과 소멸에 연결(후크)되어 Vue 인스턴스의 상태에 따라 자동으로 실행되는 콜백 메서드입니다.

[소스-index07.html]

```
<div id='first'>
  <h3 v-html="num"></h3>
  <button @click="plus">+</button>
</div>
<script type="text/javascript">
  var model = {
    title : 'SSAFY',
    num : 0
  };
  var first = new Vue({
    el : '#first',
    data : model,
    beforeCreate: function () {
      console.log('beforeCreate title: ' + this.title);
    },
    created: function () {
      console.log('created title: ' + this.title);
      console.log('created 연결된 DOM 엘리먼트: ' + this.$el);
    },
    beforeMount: function () {
      console.log('beforeMount title: ' + this.title);
    },
    mounted: function () {
      console.log('mounted title: ' + this.title);
      console.log('created 연결된 DOM 엘리먼트: ' + this.$el);
    },
```

```
    beforeUpdate: function () {
      console.log('beforeUpdate 호출됨');
    },
    updated: function () {
      console.log('updated 호출됨');
    }
    beforeDestory: function () {
      console.log('beforeDestory title: ' + this.title);
    },
    destory: function () {
      console.log('destory title: ' + this.title);
    }
  });
  Vue.config.devtools = true;
</script>
```

Vue 인스턴스가 생성되기 전에 this.title 은 정의가 안 된 상태가 되어 사용할 수 없습니다.
Vue 인스턴스가 마운트되기 전에는 인스턴스와 DOM 의 엘리먼트가 연결되지 않아서 사용할 수 없습니다.
+버튼을 클릭하여 인스턴스의 data 가 업데이트 되었을 때 리-렌더링 전후로 beforeUpdate 와 updated 가 호출됩니다.



[그림 2.5] 라이프 사이클 후크 메서드

2.2.5 Vue 인스턴스와 컴포넌트

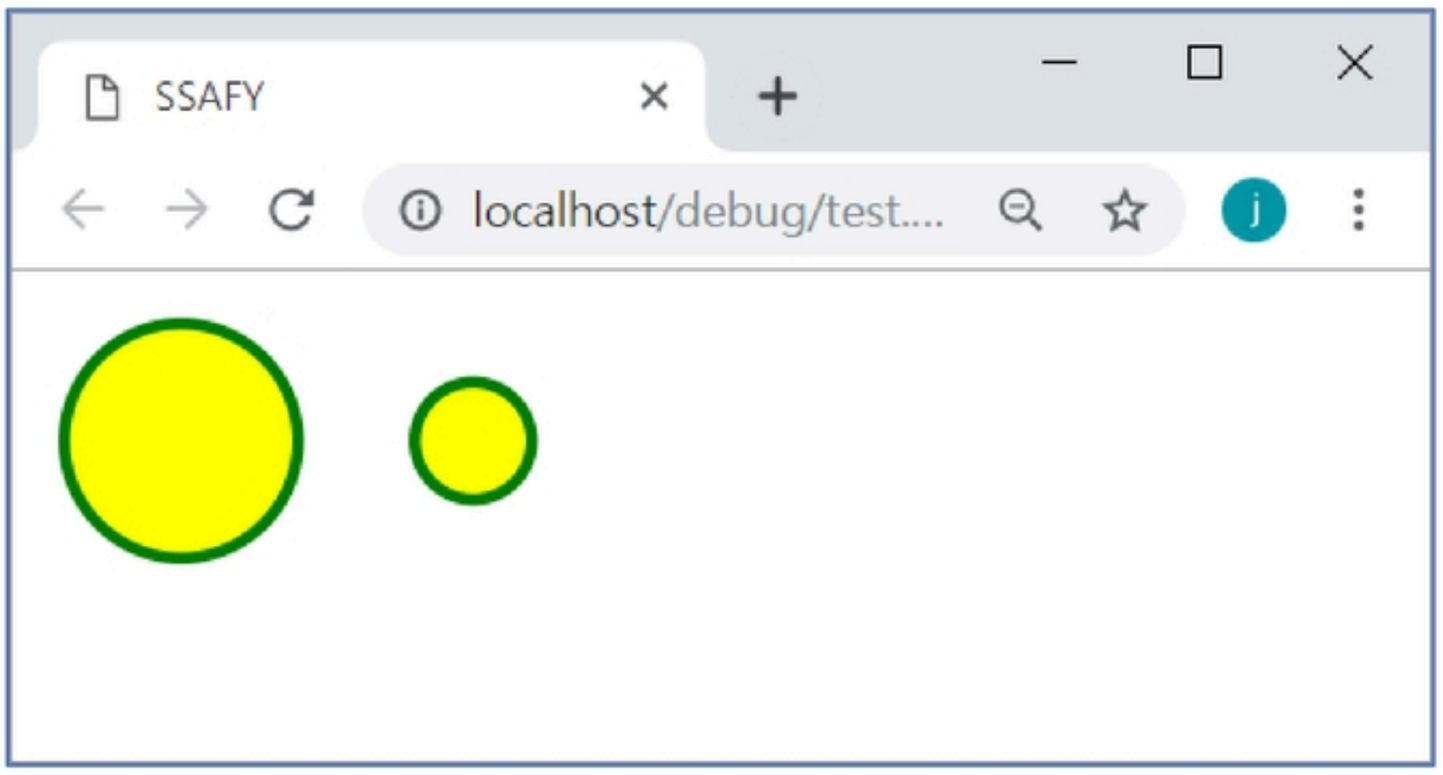
Vue 인스턴스에 지역(내부) 컴포넌트를 선언할 수 있습니다.
원하는 화면(View)이 일정하지만 데이터만 바꾸어서 보이고 싶을 때 컴포넌트를 사용합니다.
예를 들어 원을 그리려고 하면 원을 그리는 요소들을 동일하지만 위치, 반지름의 정보에 따라 보여 줄 수 있습니다. 컴포넌트는 화면을 보이는 템플릿과 템플릿과 입력되는 속성을 바인딩하는 부분이 필요합니다.

[소스-index08.html]에서 HTML 에서 <my-svg ccx='100'></my-svg>는 정의한 컴포넌트를 실행합니다. 이때 속성 ccx 는 template:에 속성으로 설정한 :cx=ccx 에 대입되는데 이런 속성을 바인딩 해주는 것이 props: ['ccx']입니다. :cx 는 cx 속성을 의미합니다. props 는 뒤에 다루는 내용으로 부모 컴포넌트에서 자식 컴포넌트로 값을 전달할 수 있습니다. 여기서는 first 부모에서 <my-svg> 자식으로, x,y 좌표와 반지름 r 을 전달할 때 props: ['ccx','ccy','cr']와 같은 방법을 사용합니다.

[소스-index08.html]

```
<div id='first'>
  <svg width="200" height="100">
      <my-svg ccx="50" ccy="50" cr="40"></my-svg>
      <my-svg ccx="150" ccy="50" cr="20"></my-svg>
  </svg>
</div>
<script type="text/javascript">
  var first = new Vue({
       el : '#first',
       components:{
            'my-svg': {
             props: ['ccx','ccy','cr'],
             template: '<circle :cx="ccx" :cy="ccy" :r="cr" stroke="green" stroke-width="4" fill="yellow" />'
            }
       }
   });
  Vue.config.devtools = true;
</script>
```

실행 결과는 다음과 같습니다.



***참고 SVG**

[https://www.w3schools.com/html/html5_svg.asp]

```
<!DOCTYPE html>

<html>

<body>

<svg width="100" height="100">

  <circle cx="50" cy="50" r="40"   stroke="green" stroke-width="4" fill="yellow" />

</svg>

</body>
</html>
```

원의 x 좌표(cx), 원의 y 좌표(cy), 원의 반지름(r)의 원을 그립니다.

3. Vue 디렉티브

Vue는 디렉티브라는 기능을 제공합니다. 디렉티브는 HTML의 속성 값을 활용해서 만든 특수한 기능입니다. HTML 태그 안에 “v-” 로 시작하는 속성을 넣어줌으로써 Vue에서 제공하는 기능을 사용할 수 있도록 해줍니다.

3.1 기본 디렉티브

3.1.1 HTML 리스트 데이터와 폼 데이터

Vue 에서 뷰 모델에 선언한 data 와 뷰에서 화면출력부분을 바인딩할 때, HTML 리스트 데이터인가 폼 태그인가에 따라 바인딩하는 방법이 달라질 수 있습니다.
리스트는 단방향(한방향)으로 폼은 단방향, 양방향으로 바인딩 할 수 있습니다.

HTML 리스트 태그(텍스트 태그)는 아래와 같이 img, table, ol 있으며, span, div 등도 많이 사용합니다.

Image

```
<img src="my.jpg" alt="me" width="500" height="333">
```

table, tr, td

```
<table style="width:100%">
  <tr>
    <td>이동관</td>
    <td>대전</td>
    <td>23</td>
  </tr>
</table>
```

ol, ul, li

```
<ol>
  <li>땅콩</li>
  <li>연어</li>
  <li>담배</li>
</ol>
```

HTML 폼 태그는 〈form〉〈/form〉 태그 내부에 입력하여 〈form〉 태그 안에 입력 받은 데이터(파라메터)를 서버로 보낼 때 사용합니다. 〈input type=’checkbox’〉와 같이 input 의 type 속성을 이용합니다. 이런 폼 데이터 계열은 value 속성을 이용하여 값을 입력 받고 전달합니다.

서버로 데이터를 보낼 때는 name : value 형식으로 파라메터로 전달합니다.

value 의 값을 변경하면 바인딩 된 화면에 값을 변경된 값이 반영되는 방식입니다.

```
<form>
  <input type="button" name="button"><input type="date" name="date">
  <input type="time" name="time">
  <input type="email" name="email">
  <input type="hidden" name="hidden">
  <input type="number" name="number">
  <input type="password" name="password">
  <input type="text" name="text">
</form>
```

textarea

```
<textarea name="message" rows="10" cols="30">
  알레르기 성분이 있습니다.
</textarea>
```

checkbox

```
<form>
  <input type="checkbox" name="box" value="대두" /><label>대두</label>
  <input type="checkbox" name="box" value="땅콩" /><label>땅콩</label>
  <input type="checkbox" name="box" value="우유" /><label>우유</label>
</form>
```

radio

```
<form>
  <input type="radio" name="box" value="대두"/>대두
  <input type="radio" name="box" value="땅콩" />땅콩
  <input type="radio" name="box" value="우유" />우유
</form>
```

Select

```
<select name="allergen" size="3">
  <option value="대두">대두</option>
```

```
  <option value="땅콩">땅콩</option>
  <option value="우유">우유</option>
</select>
```

3.1.2 v-if , v-else-if, v-else 디렉티브

v-if 는 조건 디렉티브로 참인 태그를 출력합니다. 그래서 조건부 렌더링이라고도 합니다.
v-if 만 있다면 거짓일 경우 <!-- -->가 됩니다.
또한, v-else-if 디렉티브를 사용해 v-if 조건을 만족하지 않을 경우 새로운 조건을 통해 해당태그의 출력여부를 결정할 수도 있습니다. 마지막으로 v-else 디렉티브를 이용하여 v-if, v-else-if 모두 만족하지 않는 경우에 해당태그를 출력하게 만들 수 있습니다.

[소스-index01_1.html]에서 input 태그의 value 값에 모델링 되어 있는 뷰의 data 의 속성 val 이 문자 'A'일 경우 v-if 에 해당하는 태그를, 'B'일 경우 v-else-if 에 해당하는 태그를, 그 외에 모든 경우는 v-else 에 해당하는 태그를 렌더링합니다.

[소스-index01_1.html]

```
<div id='first'>
  <div v-html = "title"></div>
  <input v-model="val">
  <div v-if="val==='A'"><h3>v-if 실행</h3></div>
  <div v-else-if="val==='B'"><h3>v-else-if 실행</h3></div>
  <div v-else><h3>v-else 실행</h3></div>
</div>
<script type="text/javascript">
  var model = {
    title : 'SSAFY',
    val : 'A'
  };
  var first = new Vue({
    el : '#first',
    data : model,
  });
```

```
  Vue.config.devtools = true;
</script>
```

위 코드의 실행 결과입니다.

[소스-index01_2.html]는 여러 태그의 출력여부를 특정 조건에 따라 한꺼번에 결정하고 싶을 경우엔 <template></template>태그에 조건 디렉티브를 사용하여 구현할 수 있습니다.
(예시 : <template v-if>…</template>)

[소스-index01_2.html]

```
<div id='first'>
  <div v-html = "title"></div>
  <input v-model="val">
  <template v-if="val=='A'">
    <div>val의</div>
    <div>값이</div>
    <div>A인 경우입니다.</div>
  </template>
  <template v-else>
    <div>val의</div>
    <div>값이</div>
    <div>A가 아닌 경우입니다.</div>
  </template>
</div>
<script type="text/javascript">
  var model = {
    title : 'SSAFY',
    val : 'A'
  };
  var first = new Vue({
    el : '#first',
```

```
    data : model,
  });
  Vue.config.devtools = true;
</script>
```

실행 결과는 다음과 같습니다.

3.1.3 v-for 디렉티브

v-for 는 순환 디렉티브로 프로그래밍의 for 와 같이 순환 또는 반복적인 작업을 합니다. 그래서 리스트 렌더링이라고도 합니다.

배열처럼 1 개 이상인 값을 출력하는데 사용합니다. v-for 는 자바의 향상된 for 처럼 사용합니다.

v-for='배열 요소(이름은 마음대로 정합니다.) in 배열' 형태로 사용합니다. 배열의 순서값인 인덱스를 사용하고 싶으면, v-for='(배열 요소, 인덱스) in 배열'형태로 사용합니다.

[소스-index02.html]

```
<div id="app">
  <h2 v-for="(item,index) in items">{{ index+"번째 물품 : "+item}}</h2>
</div>
<script type="text/javascript>
  ...
  new Vue({
    el: "#app",
    data: {
      items: ["연필","펜","지우개"]
    }
  })
  ...
</script>
```

결과는 다음과 같습니다.

0번째 물품 : 연필

1번째 물품 : 펜

2번째 물품 : 지우개

v-for 는 배열뿐만 아니라 객체에도 사용이 가능합니다. 객체에 사용할 경우 3 가지의 방법이 있습니다.
v-for='value in 객체' 형태로 사용하는 경우 각 객체 요소의 value 값만을 가져옵니다.
v-for='(value, name) in 객체' 형태로 사용하는 경우 각 객체 요소의 value 값과 key 값을 가져옵니다.
v-for='(value, name, index) in 객체' 형태로 사용하는 경우 각 객체 요소의 value 값과 key 값, 순서값을 가져옵니다.

[소스-index03.html]

```
<div id="app">
  <h2 v-for="(value, name, index) in items">{{ index+"번째 물품 : "+name+", 가격 :
"+value}}</h2>
</div>
<script type="text/javascript>
  ...
  new Vue({
    el: "#app",
      data: {
        items: {
          pencil: 500,
          pen: 1000,
          eraser: 300
        }
      }
  })
  ...
</script>
```

결과는 다음과 같습니다.

0번째 물품 : pencil, 가격 : 500

1번째 물품 : pen, 가격 : 1000

2번째 물품 : eraser, 가격 : 300

아래 소스는 다음과 같은 구조를 갖습니다. 〈template v-for〉〈/template〉을 사용하면 태그 안의 내용만 출력되고 〈template〉 태그는 남지 않습니다.

```
<template   v-for="flag in flags">
    <label><span class="label" v-html="flag.name"></span></label>
    <input type="text" v-bind:value="flag.korname">
    <img v-bind:src="flag.flag"/><br/>
</template>
```

우선 뷰 모델의 data 의 모델에 JSON 형식으로 선언한 데이터를 보면 다음과 같습니다.

{flags:[{name:' ', korname:' ', flag:' '}]}
flags 라는 키로 배열([ ])을 얻을 수 있습니다. 그리고 배열의 한개 엘리먼트(요소)는 다시 {name:' ', korname:' ', flag:' '}로 구성되고 name, korename, flag 로 해당 값들을 얻을 수 있는 구조입니다.
이것을 출력하기 위해 〈태그 v-for='반복되는배열값 in 배열'〉에 넣습니다. 그러면 〈template v-for="flag in flags"〉가 됩니다.
배열 flags 에서 한 개의 flag 엘리먼트는 {name:' ', korname:' ', flag:' '}을 의미하기 때문에 flag.name, flag.korname, flag.flag 로 원하는 값을 얻을 수 있습니다.
〈template〉〈/template〉을 사용하면 〈template〉 태그는 남지 않는다는 것도 중요합니다.

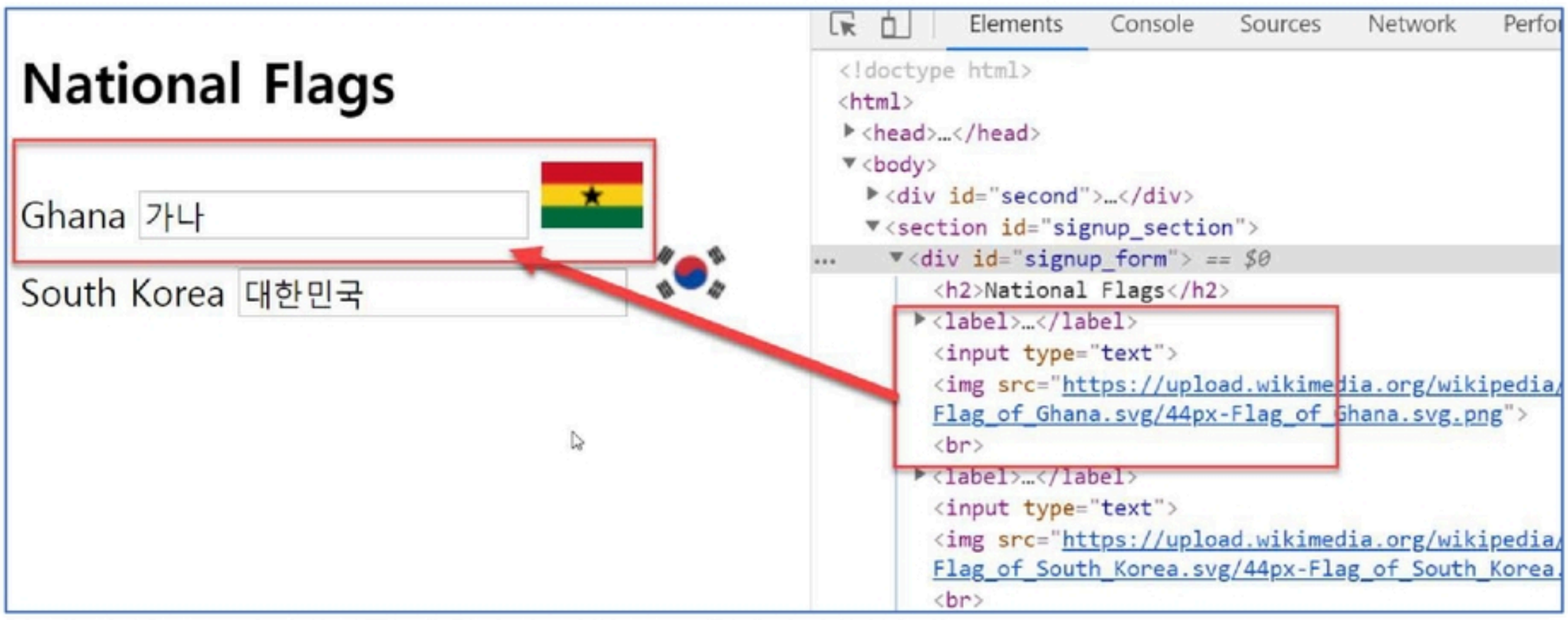
〈span〉〈/span〉 엘리먼트 값을 출력하기 위해 v-html 디렉티브를 사용합니다. 그래서 〈span v-html="flag.name"〉〈/span〉을 사용합니다.

〈input type='text'/〉에 값(value)를 입력하기 위해서는 v-bind:value 혹은 v-bind 를 빼고 :value 를 사용합니다. Input 은 단방향 바인딩 bind 와 양방향 바인딩 v-model 을 사용할 수 있지만 값만 출력하기 위해서 bind 를 사용합니다. v-bind 는 속성 값에 바인딩합니다. 그래서 〈input type='text' v-bind:value='flag.korname'/〉을 사용합니다.

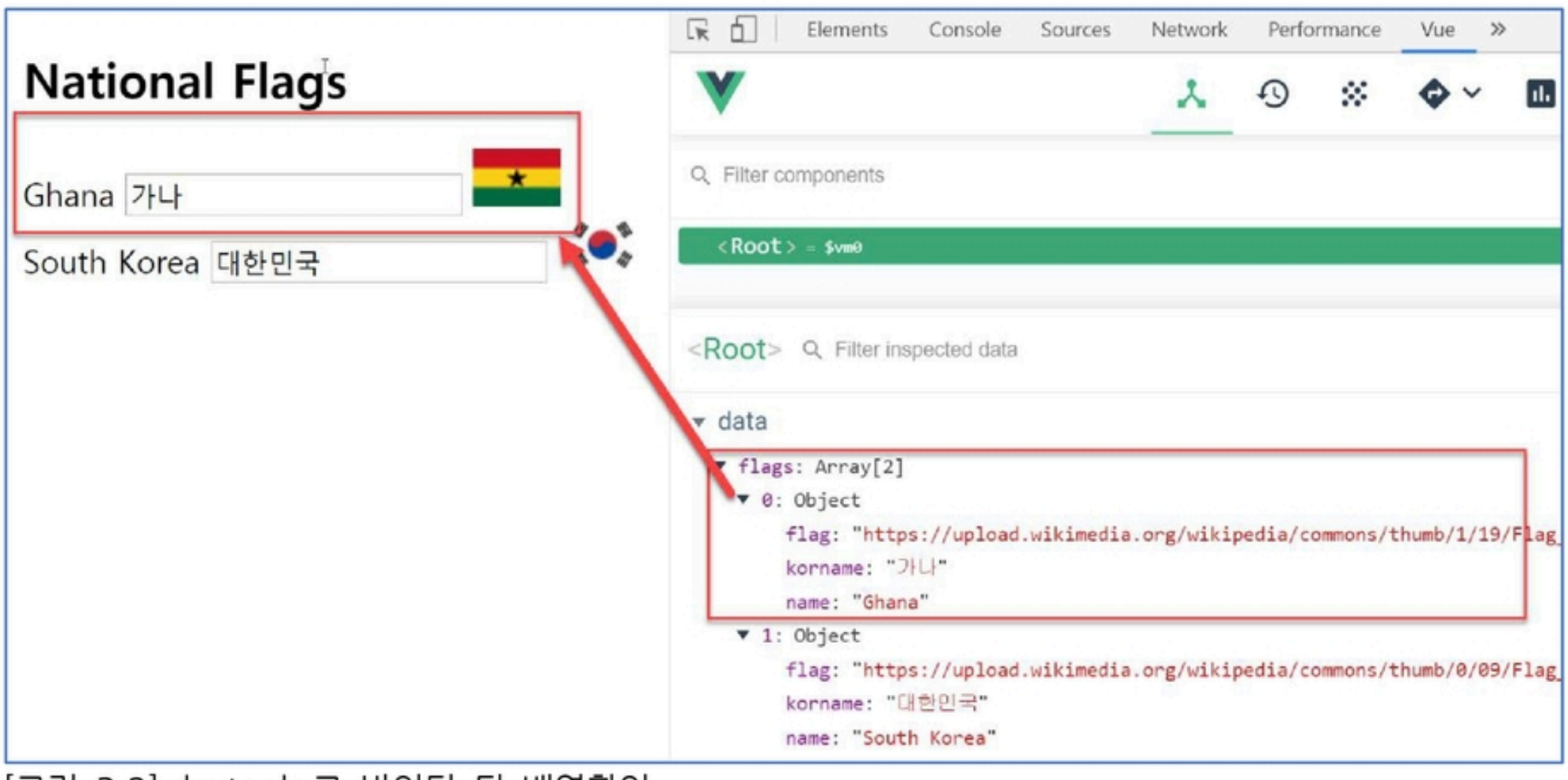
〈img/〉의 src 속성에 이미지 경로를 바인딩하기 위해서 〈img v-bind:src= "flag.flag"/〉를 사용합니다.

[소스-index04.html]

```
<body>
  <section id="signup_section">
    <div id='signup_form'>
      <h2>National Flags</h2>
      <template  v-for="flag in flags">
        <label><span class="label" v-html="flag.name"></span></label>
        <input type="text" v-bind:value="flag.korname">
        <img v-bind:src="flag.flag"/><br/>
      </template>
    </div>
  </section>
  <script type="text/javascript">
    var model = {
      flags:[
        {name:"Ghana",korname:"가나", flag:"44px-Flag_of_Ghana.svg.png"},
        {name:"South Korea",korname:"대한민국", flag:"44px-Flag_of_South_Korea.svg.png"}
      ]
    };
    var signup_section = new Vue({
      el : '#signup_section',
      data : model
    });
    Vue.config.devtools = true;
  </script>
</body>
```



[그림 3.1] for 디렉티브를 이용하여 반복으로 출력하기(렌더링)



[그림 3.2] devtools 로 바인딩 된 배열확인

〈tr〉〈/tr〉는 반복을 한 후 남기고 싶기 때문에 〈template〉을 사용하지 않고 〈tr v-for〉를 사용합니다.
〈tr v-for="flag in flags"〉는 flags 배열에서 한 개의 엘리먼트를 flag 라고 한 것입니다.
〈tr〉〈/tr〉에 세 개의 〈td〉〈/td〉를 사용합니다. 〈td〉〈/td〉 엘리먼트에 값을 출력하기 위해서 v-html 을 사용하고 〈img〉의 속성 src 에 바인딩하기 위해 v-bind:src 를 사용합니다.

v-bind:src 는 :src 로 사용할 수 있습니다.

[소스-index05.html]

```
<section id="signup_section">
  <h2>National Flags</h2>
  <table id='tid' border='1px'>
    <tr><td>나라이름</td><td>나라이름</td><td>국기</td></tr>
    <tr v-for="flag in flags">
      <td v-html="flag.name"></td>
      <td v-html="flag.korname"></td>
      <td><img v-bind:src="flag.flag"/></td>
    </tr>
  </table>
</section>
```

[그림 3.3] devtools 로 바인딩 된 배열 확인

3.1.4 v-pre 디렉티브

v-pre 디렉티브는 해당 태그와 자식태그들이 Vue 를 사용하지 않는다고 알려주는 디렉티브 입니다. Vue 를 사용하지 않는 코드들을 Vue 에서 컴파일 하지 않고 skip 하기 때문에 그 코드의 내용이 길수록 컴파일 성능 향상에 도움을 줍니다. 컴파일 하지 않기 때문에 콧수염 표현식도 문자열 그대로 출력이 됩니다.

[소스-index06_1.html]

```
<div id="app">
    <h2 v-pre>{{ 그대로 출력해주세요 }}</h2>
</div>

<script type="text/javascript">
    new Vue({
        el: "#app",
        data: {}
    })
    Vue.config.devtools = true;
</script>
```

결과는 다음과 같습니다.

{{ 그대로 출력해주세요 }}

3.1.5 v-cloak 디렉티브

v-cloak 디렉티브는 자바스크립트의 로딩이 끝나기 전까지 해당 엘리먼트에 남아서 display 되지 않도록 가려주는 역할을 하는 디렉티브 입니다. 웹 페이지가 열릴 때 처음에 자바스크립트가 완전히 실행되기 전에는 원하는 출력값이 아닌 다른 값이 출력될 수 있습니다. 예를 들어 콧수염 표현식이 Vue 가 컴파일 해주기 전에 먼저 출력이 되어서 문자열 그대로 출력되는 경우가 있습니다. v-cloak 디렉티브를 사용해주면 이런 현상을 방지할 수 있습니다.
자바스크립트 실행 전 출력되는 경우

{{ title }}

v-cloak 디렉티브를 사용할 때는 display 를 조작해주기 위해 CSS 설정도 같이 해주어야 합니다.

[소스-index06_2.html]

```
<style>
    [v-cloak] { display: none }
</style>
...
<div id="app">
    <h2 v-cloak>{{ title }}</h2>
</div>

<script type="text/javascript">
    new Vue({
        el: "#app",
        data: {
            title: "안녕하세요"
        }
    })
    Vue.config.devtools = true;
</script>
```

이제 자바스크립트가 완전히 실행되기 전까지는 display 속성이 none 으로 지정되어 출력이 되지 않습니다.

3.1.6 v-once 디렉티브

v-once 디렉티브는 해당 엘리먼트나 컴포넌트를 단 한번만 렌더링하게 해줍니다. 그 뒤로는 렌더링 할 때마다 정적인 엘리먼트로 처리되기 때문에 렌더링을 하지 않고 건너뛰게 됩니다. 이를 이용해서 업데이트 성능을 최적화 할 수도 있습니다.
v-once 디렉티브를 이용해서 카운터의 초기 값과 현재 값을 출력하는 페이지를 작성해보겠습니다.

[소스-index06_3.html]

```
<div id="app">
    <h2 v-once>{{ "초기값 : "+count }}</h2>
    <h2>{{ "현재값 : "+count }}</h2>
    <button @click="count++">+</button>
</div>

<script type="text/javascript">
    new Vue({
        el: "#app",
        data: {
            count: 1
        }
    })
    Vue.config.devtools = true;
</script>
```

결과는 다음과 같습니다.

초기값 : 1

현재값 : 7

+

3.2 바인딩

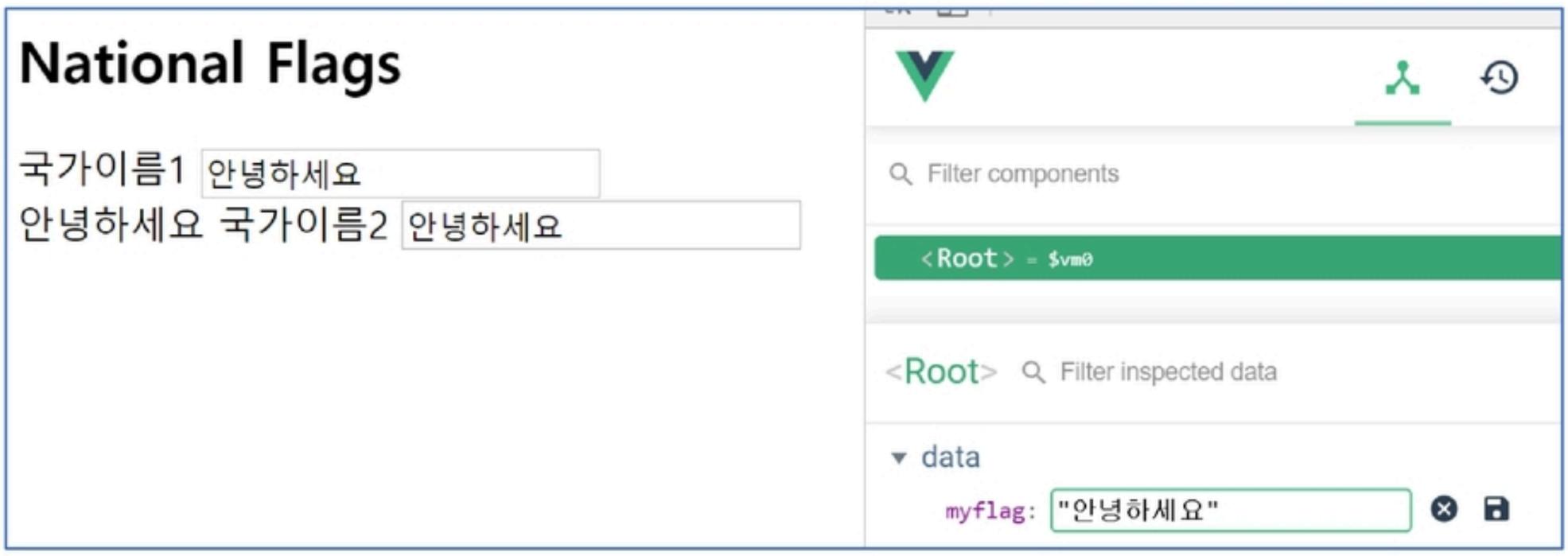
3.2.1 v-bind, v-model 디렉티브

v-bind 는 단방향 바인드로 데이터를 화면에 출력하는데 사용합니다. 리스트 데이터는 데이터를 입력 받는 value 속성이 없기 때문에 v-bind 를 사용합니다. 그러나 폼 데이터(input, textarea, selecto)는 value 속성으로 값을 입력 받을 수 있습니다. 폼 데이터에 있는 데이터를 변경하거나 입력 한다면 입력 받거나 변경된 값을 모델(데이터)에 적용해 다른 엘리먼트에 반영시킬 수 있습니다. 이처럼 폼 데이터는 양방향으로 데이터의 변경된 값을 반영할 수 있습니다.

양방향 데이터 바인딩을 위해서 v-model 디렉티브를 사용합니다. [소스-index7.html]에서
국가이름 1 과 국가이름 2 는 input 폼 데이터로 v-model 로 양방향 바인딩하였고, span 은 입력 받지 않고 출력한 하기 위해 v-html 로 바인딩 했습니다. 국가이름 1 과 국가이름 2 에서 input 속성의 value 값을 변경하면 서로 변경된 값을 반영시킬 수 있습니다.

[소스-index7.html]

```
<section id="signup_section">
  <div id='signup_form'>
    <h2>National Flags</h2>
    국가이름1 <input type="text"   v-model='myflag'>
  </div>
  <label><span class="label" v-html="myflag"></span></label>
  국가이름2 <input type="text"   v-model='myflag'>
</section>
<script type="text/javascript">
  var model = {myflag:''};
  var signup_section = new Vue({
    el : '#signup_section',
    data : model
  });
  Vue.config.devtools = true;
</script>
```



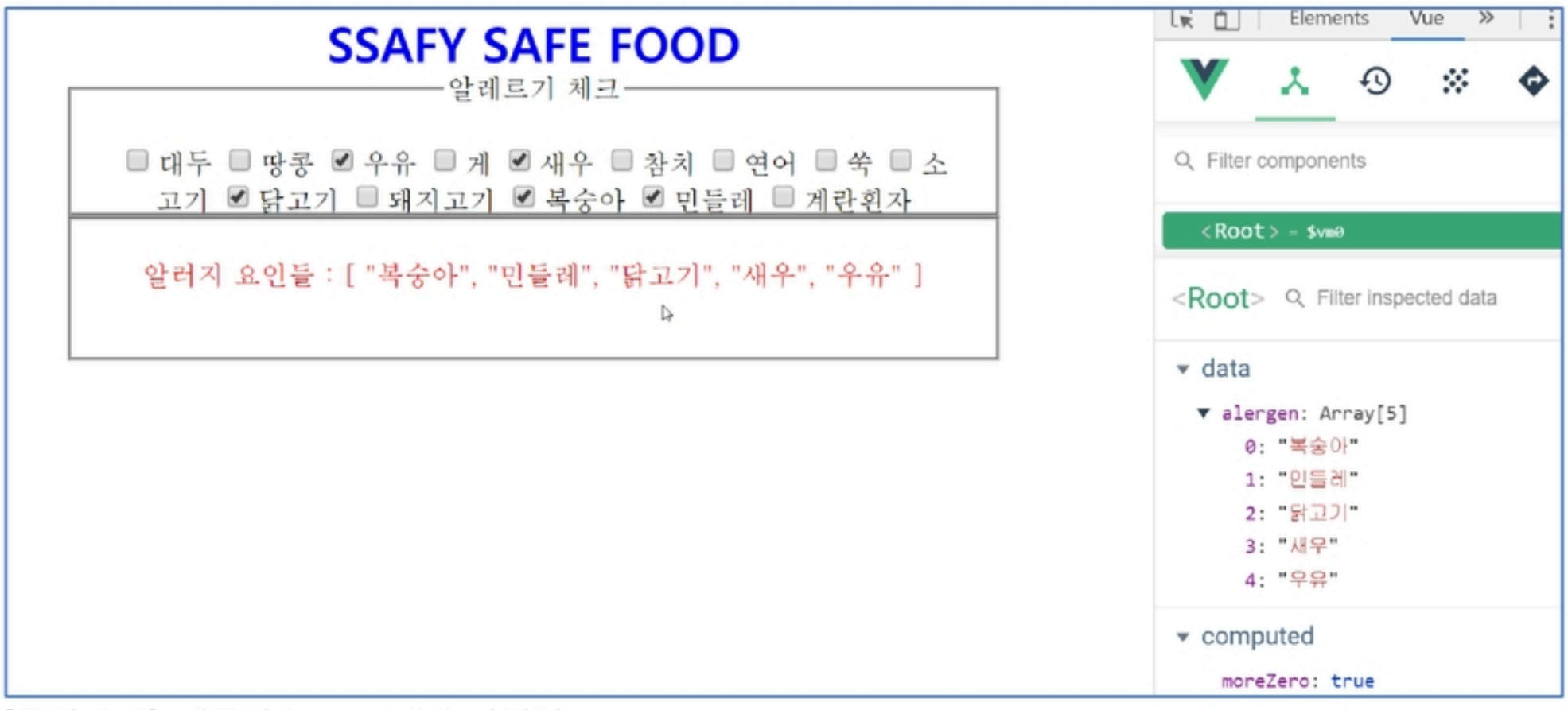
[그림 3.4] devtools 에서 v-model 테스트

[소스-index08.html]는 체크박스에서 선택한 알레르기 요인들을 모델의 allergen 에 저장하고, 알레르기 요인들의 변경이 반영되도록 v-model='allergen'으로 바인딩하였습니다.
선택한 알레르기 원인이 없다면 checkboxArea2 필드셋이 보이지 않도록 computed 옵션에 moreZero 를 선언하였습니다.

[소스-index08.html]

```
<div id='app'>
  <fieldset id="checkboxArea1">
    <legend>알레르기 체크</legend>
    <input type="checkbox" name="box" value="대두" v-model="alergen"/>
    <label>대두</label>
    <input type="checkbox" name="box" value="땅콩" v-model="alergen"/>
    <label>땅콩</label>
    <input type="checkbox" name="box" value="우유" v-model="alergen"/>
    <label>우유</label>
  </fieldset>
  <fieldset id="checkboxArea2" v-if='moreZero'>
    알러지 요인들 : <span v-html="alergen"></span>
  </fieldset>
</div>
<script type="text/javascript">
  var model = {
```

```
    alergen : []
  };
  var allergenes1 = new Vue({
    el : '#app',
    data : model,
    computed: {
      moreZero: function () {
        return this.alergen.length > 0
      }
    }
  });
  Vue.config.devtools = true;
</script>
```



[그림 3.5] 체크박스 v-model 바인딩

[소스-index09.html]는 체크박스에서 선택한 알레르기 요인을 모델의 allergen 에 저장하고, 알레르기 요인의 변경이 반영되도록 v-model='allergen'으로 바인딩하였습니다.
선택한 알레르기 원인이 없다면 checkboxArea2 필드셋이 보이지 않도록 computed 옵션에 moreZero 를 선언하였습니다.

결국 [소스-index08.html]과 동일하지만 v-for 를 이용하여 많은 반복 태그들을 제거하였습니다.

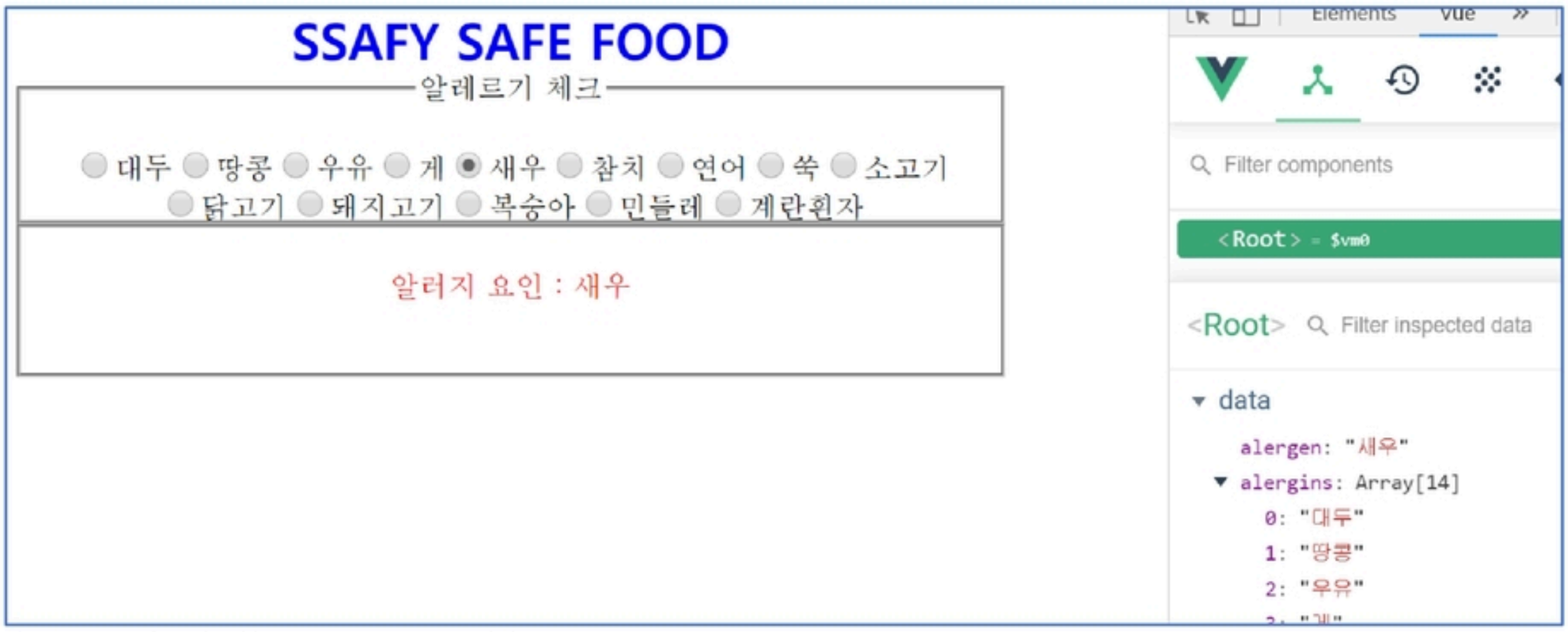
[소스-index09.html]

```
<div id='allergenes1'>
  <fieldset id="checkboxArea1">
    <legend>알레르기 체크</legend>
    <template   v-for="alergin in alergins">
      <input type="checkbox" name="box" v-bind:value="alergin" v-model="alergen" />
      <label>{{alergin}}</label>
    </template>
  </fieldset>
  <fieldset id="checkboxArea2" v-if='moreZero'>
    알러지 요인들 : <span v-html="alergen"></span>
  </fieldset>
</div>
<script type="text/javascript">
  var model = {
    allergen : [],
    alergins :  ["대두","땅콩","우유","게","새우","참치","연어","쑥","소고기","닭고기","돼지고기","복숭
아","민들레","계란흰자"]
  };
  var allergenes1 = new Vue({
    el : '#allergenes1',
    data : model,
    computed: {
      moreZero: function () {
        return this.alergen.length > 0
      }
    }
  });
</script>
```

[소스-index10.html]는 소스 [소스-index08.html]의 checkbox를 라디오로 변경한 것입니다.
알레르기 요인 중 한 가지만을 선택할 수 있습니다.

[소스-index10.html]

```
<div id='allergenes1'>
  <fieldset id="checkboxArea1">
  <legend>알레르기 체크</legend>
  <input type="radio" name="box" value="대두" v-model="alergen"/><label>대두</label>
  <input type="radio" name="box" value="땅콩" v-model="alergen"/><label>땅콩</label>
  <input type="radio" name="box" value="우유" v-model="alergen"/><label>우유</label>
  </fieldset>
  <fieldset id="checkboxArea2" v-if='moreZero'>
    알러지 요인들 : <span v-html="alergen"></span>
  </fieldset>
</div>
```



[그림 3.6] 라디오 v-model 바인딩

[소스-index11.html]는 소스 [소스-index10.html]의 v-for 디렉티브를 이용하여 라디오를 만든 것입니다. 알레르기 요인 중 한 가지만을 선택할 수 있습니다.

[소스-index11.html]

```
<div id='allergenes1'>
  <fieldset id="checkboxArea1">
    <legend>알레르기 체크</legend>
    <template   v-for="alergin in alergins">
      <input type="radio" name="box" v-bind:value="alergin" v-model="alergen" />
      <label>{{alergin}}</label>
    </template>
  </fieldset>
  <fieldset id="checkboxArea2" v-if='moreZero'>
    알레르기 요인 : <span v-html="alergen"></span>
  </fieldset>
</div>
```

3.3 사용자 정의 디렉티브

3.3.1 사용자 정의 디렉티브

Vue.directive 를 이용하여 사용자 디렉티브를 만들 수 있습니다.
Vue.directive('사용자 정의 디렉티브 이름', bind(elm, binding, nodename){})으로 구성합니다.
elm : 디렉티브를 적용할 엘리먼트, **binding** : 아규먼트 관련 데이터를 가지고있는 자바스크립트 object, **nodename** : 디렉티브를 감싸는 태그 이름
bind 는 디렉티브가 엘리먼트에 처음 바인드 됐을 때 한번만 실행됩니다. CSS 스타일의 값을 변경하고 싶다면 elm.style.속성='값'을 이용합니다. 사용자 디렉티브에 "v-"붙여서 사용합니다.

binding 의 구성요소는 다음과 같습니다.

```
binding = {
   name: "v-를 제외한 디렉티브의 이름",
   value: "디렉티브에 전달된 값 ex) v-custom="1+1"의 경우 2 가 들어옵니다.",
   expression: "디렉티브에 전달된 값의 string 형태 ex) v-custom="1+1"의 경우
"1+1"이 들어옵니다."
}
```

[소스-index12.html]

```
<div id = "mydirective">
   <div v-mystyle>월요일</div>
   <div v-mystyle>화요일</div>
</div>
<script type = "text/javascript">
   Vue.directive("mystyle",{
      bind(elm,binding, nodename) {
         elm.style.color = "blue";
         elm.style.fontSize = "20px";
         elm.style.border = "1px";
         elm.style.borderStyle = "dotted";
         elm.style.textAlign = 'left';
      }
   });
```

```
  var mydirective = new Vue({
    el: '#mydirective'
  });
</script>
```

Vue.directive를 이용하여 사용자 디렉티브를 만들 수 있습니다. 디렉티브에서 값을 입력 받게 하려면 디렉티브에 JSON 형식의 값을 대입합니다.

v-mystyle="{color:'blue', fontSize:'20px', border:'1px', borderStyle:'dotted'}"는 mystyle 디렉티브에 입력 받은 값을 대입합니다.
bind(elm,binding, nodename)에서 입력 받은 값을 대입할 때는 binding.value.키 값을 이용합니다.

[소스-index13.html]

```
<div id = "mydirective" >
  <div v-mystyle="{color:'blue', fontSize:'20px', border:'1px', borderStyle:'dotted'}">
    토요일
  </div>
  <div v-mystyle="{color:'red', fontSize:'20px', border:'1px', borderStyle:'solid'}">
    일요일
  </div>
</div>
<script type = "text/javascript">
  Vue.directive("mystyle",{
    bind(elm,binding, nodename) {
      elm.style.color = binding.value.color;
      elm.style.fontSize = binding.value.fontSize;
      elm.style.border = binding.value.border;
      elm.style.borderStyle =binding.value.borderStyle;
      elm.style.textAlign = 'center';
      //console.log(nodename.tag);
      //console.log(nodename.elm);
    }
  });
  var mydirective = new Vue({
    el: '#mydirective'
```

```
  });
</script>
```

[그림 3.7] 사용자 정의 디렉티브

4. Vue 와 CSS

4.1 CSS 기본

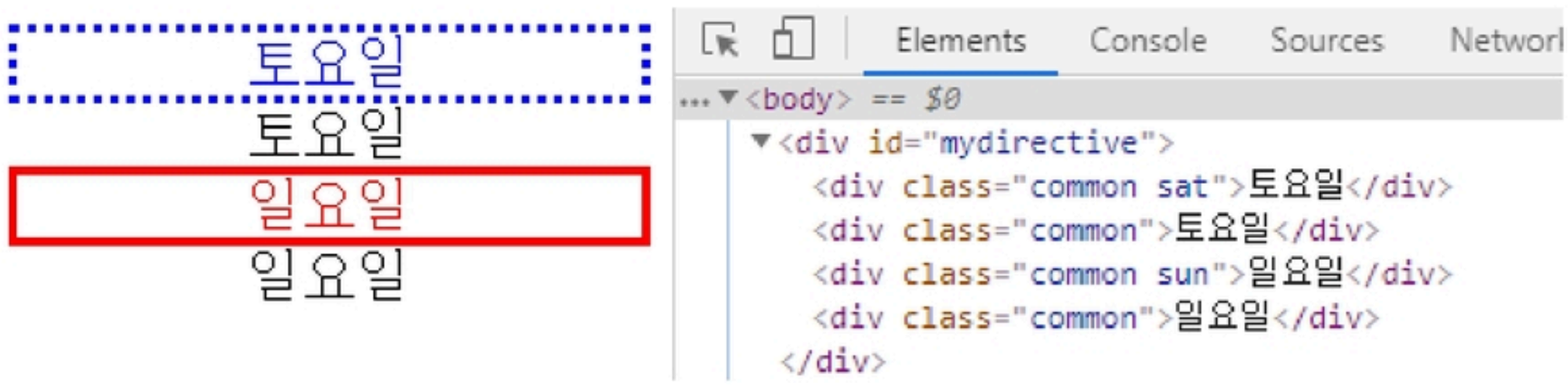
4.1.1 HTML 리스트 데이터와 폼 데이터

Vue 에서 CSS 스타일을 주는 방법은 크게 두 가지가 있습니다.
CSS 에서 사용하는 방법과 v-bind:class="{클래스이름: 보일지안보일지}"방법이 있습니다. 보일지안보일지에 값에 따라 클래스가 실행될지 안될지 결정됩니다. 또한 CSS 의 class 와 v-bind:class 는 같이 사용할 수 있습니다.
예제에서 isSat 이 true 면 sat 클래스가 , isSun 이 true 면 sun 클래스가 적용됩니다.

[소스-index01.html]

```
<style>
  .common{margin:auto;   text-align:center;
    font-family: serif; font-size: 20px;}
  .sat{color: blue;border-style: dotted;}
  .sun{color: red; border-style:solid;}
</style>
<script src="https://unpkg.com/vue"></script>
...
...
<div id = "mydirective" >
  <div class="common" v-bind:class="{sat: isSat }">토요일</div>
  <div class="common" v-bind:class="{sat: !isSat }">토요일 아님</div>
  <div class="common" v-bind:class="{sun: isSun }">일요일</div>
  <div class="common" v-bind:class="{sat: !isSun }">일요일 아님</div>
</div>
<script type = "text/javascript">
  var model = {isSat: true, isSun:true};
  var mydirective = new Vue({
    el: '#mydirective',
    data : model
  });
```

```
</script>
```



[그림 4.1] class 와 v-bind:class 를 같이 사용하는 경우

삼항 연산자를 사용할 수도 있습니다. isSat 이 true 이면 sat 클래스, 아니면 sun class 가 바인딩 됩니다.

배열 형태 []도 사용할 수 있습니다. 배열의 각 요소가 차례대로 class 에 적용됩니다. 배열의 요소가 여러 개면 class 도 여러 개가 적용됩니다.
index02.html 에서 배열의 첫번째 요소{ sun : isSun }는 위에 사용했던 방법에 따른 class 적용 방법이고 다음 요소인 sat 문자열은 그대로 class 에 추가 적용됩니다.

[소스-index02.html]

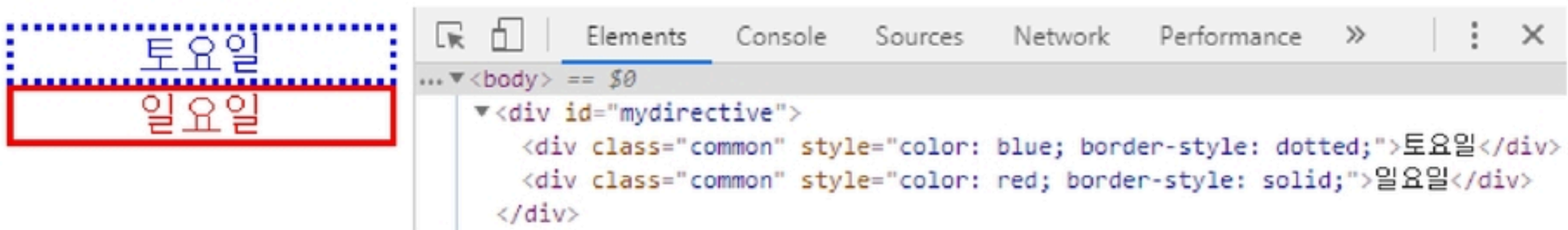
```
<div id = "mydirective" >
  <div class='common' v-bind:class="isSat?'sat':'sun'">토요일</div>
  <div class='common' v-bind:class="[{sun:isSun} ,'sat']">일요일</div>
</div>
<script type = "text/javascript">
  var model = {isSat: true, isSun:true};
  var mydirective = new Vue({
    el: '#mydirective',
    data : model
  });
</script>
```

v-bind:style 을 사용해서 직접 적용할 CSS 를 지정해줄 수도 있습니다. 디렉티브의 입력 값으로 자바스크립트 객체나 배열도 사용할 수 있으며, 지정해줄 style 은 camelcase 로 입력해주어야 합니다.
Ex) border-style → borderStyle

입력 값 안에서 삼항연산자도 사용할 수 있습니다.

[소스-index03.html]

```
<div id="mydirective">
  <div class='common' v-bind:style="{ color: 'blue', borderStyle: 'dotted'}">
    토요일
  </div>
  <div class='common' v-bind:style="isSun? {color: (isRed?'red':''), borderStyle: 'solid'}:''">
    일요일
  </div>
</div>
<script type="text/javascript">
  var model = {
    isSun: true,
    isRed: true
  }
  var mydirective = new Vue({
    el: '#mydirective',
    data: model
  });
  Vue.config.devtools = true;   //devtools 디버거용
</script>
```



5. Rendering

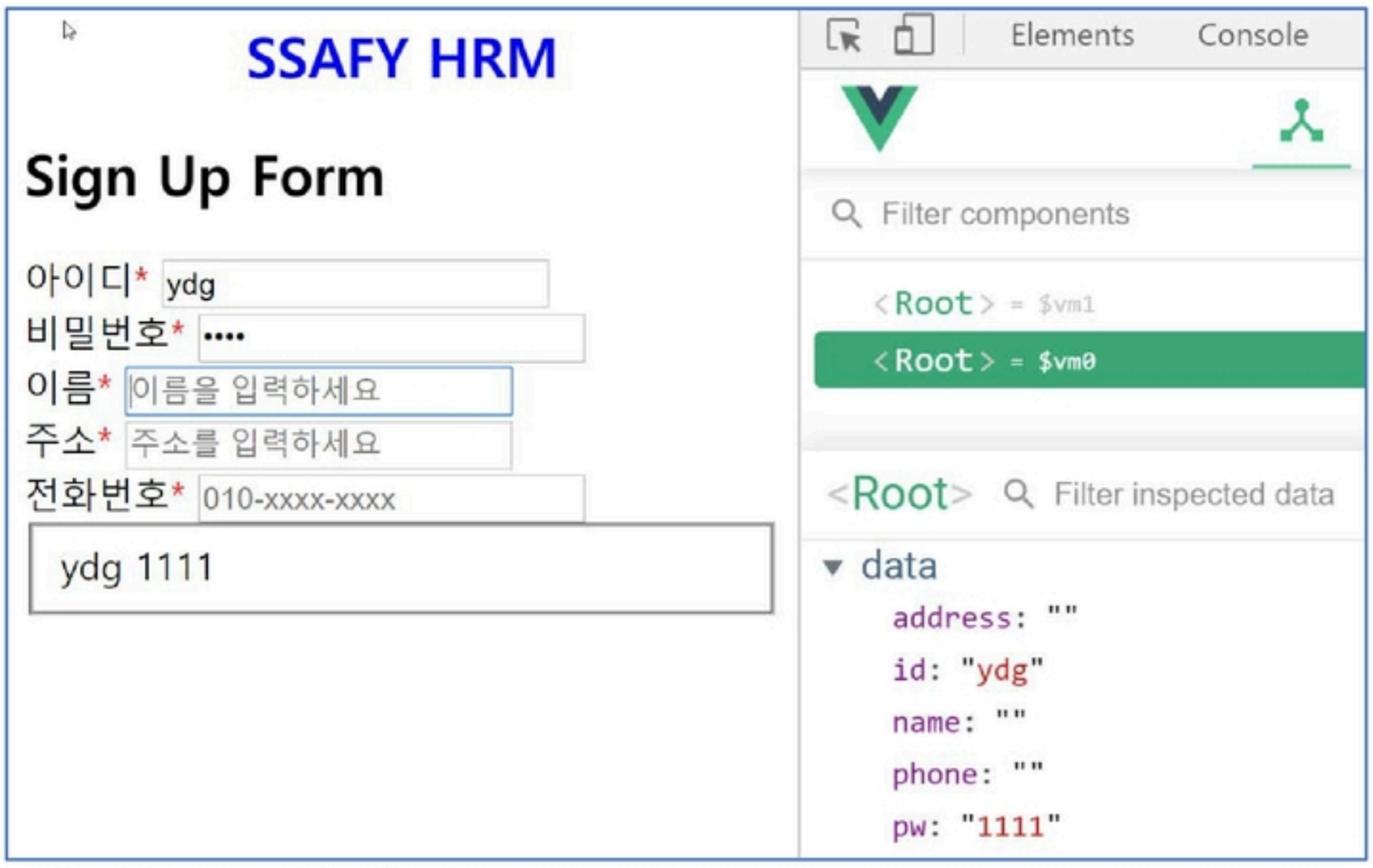
5.1 HTML 리스트 데이터와 폼 데이터 바인딩

5.1.1 바인딩

렌더링은 모델(데이터)가 있는 화면(뷰)를 화면으로 보이는 작업입니다. 바인딩은 화면(뷰)에 모델(데이터) 값을 자동으로 연결시키는 것이며 뷰 모델이 데이터 흐름과 화면을 제어합니다.

Vue 에서 뷰 모델에 선언한 data 와 뷰에서 화면출력부분을 바인딩 할 때, HTML 리스트 데이터인가, 폼 태그 인가에 따라 바인딩 하는 방법이 달라질 수 있습니다. 리스트는 단 방향(한 방향)으로 폼은 단방향, 양방향으로 바인딩 할 수 있습니다.

[소스-index01.html] 예제에서 v-model 디렉티브는 뷰 모델의 data 와 양방향 바인딩 되어 있고, v-html 디렉티브는 단방향 바인딩 되었습니다. id, pw, name, address, phone 폼 데이터는 v-model, 〈span〉 태그로 출력하는 부분은 v-html 로 바인딩되어 있습니다.



[그림 5.1] v-model,v-html 을 이용한 바인딩

중요한 것은 뷰 모델의 data에 선언한 data : {id : '',pw : '',name : '',address : '',phone : ''}입니다. 바인딩 할 속성들을 선언하고 초기화를 해야 합니다.

[소스-index01.html]

```
<section id="signup_section">
  <div id='signup_form'>
    <h2>Sign Up Form</h2>
    <form id="_formhrm" action="./main.do" method="post">
      <label><span class="label">아이디<span class='red'>*</span></span></label>
      <input id="_signId" name="id" type="text" v-model="id" placeholder="아이디를
입력하세요"/><br/>
      <label><span class="label">비밀번호<span class='red'>*</span></span></label>
      <input id="_signPw" name="pw" type="password" v-model="pw" placeholder="패스워드를
입력하세요"/><br/>
      <label><span class="label">이름<span class='red'>*</span></span></label>
      <input id="_signName" name="name" type="text" v-model="name" placeholder="이름을
입력하세요"/><br/>
      <label><span   class="label">전화번호<span class='red'>*</span></span></label>
      <input id="_signPhone" name="phone" type="tel" v-model="phone" placeholder="010-
xxxx-xxxx"/><br/>
    </form>
  </div>
  <div id='signup_form2'>
    <fieldset>
      <span v-html="id"></span>
      <span v-html="pw"></span>
      <span v-html="name"></span>
      <span v-html="address"></span>
      <span v-html="phone"></span>
    </fieldset>
  </div>
</section>
<script type="text/javascript">
  var signup_section = new Vue({
    el : '#signup_section',
    data : {
      id : '',
      pw : '',
      name : '',
```

```
        address : '',
        phone : '',
    }
})
var second = new Vue({
    el: '#second',
    data: {title: 'SSAFY HRM'}
    });
</script>
```

5.2 HTML 리스트 데이터와 폼 데이터 렌더링

5.2.1 렌더링

렌더링이란 계층적(부모 자식 관계가 있음)이면서 논리적인 문서의 표현식을 그래픽 표현식으로 변형시키는 과정입니다.

렌더링은 계층적으로 되는데 DOM 과 박스모델인 스타일을 기반으로 레이아웃이 준비되고 엘리먼트(요소-태그)을 찾아서 화면에 보입니다. 그러므로 브라우저는 뷰 디렉티브를 만나면 모델(데이터)을 반영해서 결과화면을 보입니다.

바인딩의 좋은 예시를 위해서는 여러 개의 data 가 필요하니 JSON 파일을 읽어서 처리하겠습니다. 우선 JSON 파일의 정보를 가져오기 위해 axios 모듈을 사용하겠습니다. 이를 위해 스크립트에 axios 라이브러리를 포함시킵니다.

<script src="https://unpkg.com/vue"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.18.0/axios.js"></script> axios 는 JSON 을 읽어오는데 라이프 사이클을 고려해야 합니다.

인스턴스를 준비하고 자원들을 모두 마운트하면 mounted 콜백 메서드가 자동 호출된다는 것을 앞에서 학습하였습니다. mounted 라이프 사이클의 메서드에서 axios 를 통해 JSON 데이터를 얻는데 실패하거나, 읽고 있는 중이거나, 읽는 것을 끝냈는 지를 확인할 필요가 있습니다. 따라서, JSON 을 읽는 데 성공했을 경우 info 에 읽어온 데이터를 담고 loading 을 false 로 바꿔줍니다. 이 경우엔 info 에 읽어온 데이터를 화면에 출력합니다. 반대로 실패했을 경우 errored 에 true 로 저장한 뒤 loading 을 false 로 바꿔줍니다. 이 경우엔 에러 메시지를 화면에 출력합니다.

이 때 기존에 사용하던 data 의 형식과 달리 메서드의 형태로 구현을 했습니다. 따라서 return 을 통해 data 에 접근할 수 있도록 해주었습니다. 이런 data function 은 동적 데이터 변화를 반영합니다.

Info 는 배열([])이므로 v-for 를 이용하여 렌더링합니다. JSON 파일의 데이터를 살펴보면 아래와 같습니다. v-for 를 이용하여 한 제품에 대한 ".키값"으로 밸류를 얻습니다.

[소스 - food.json]

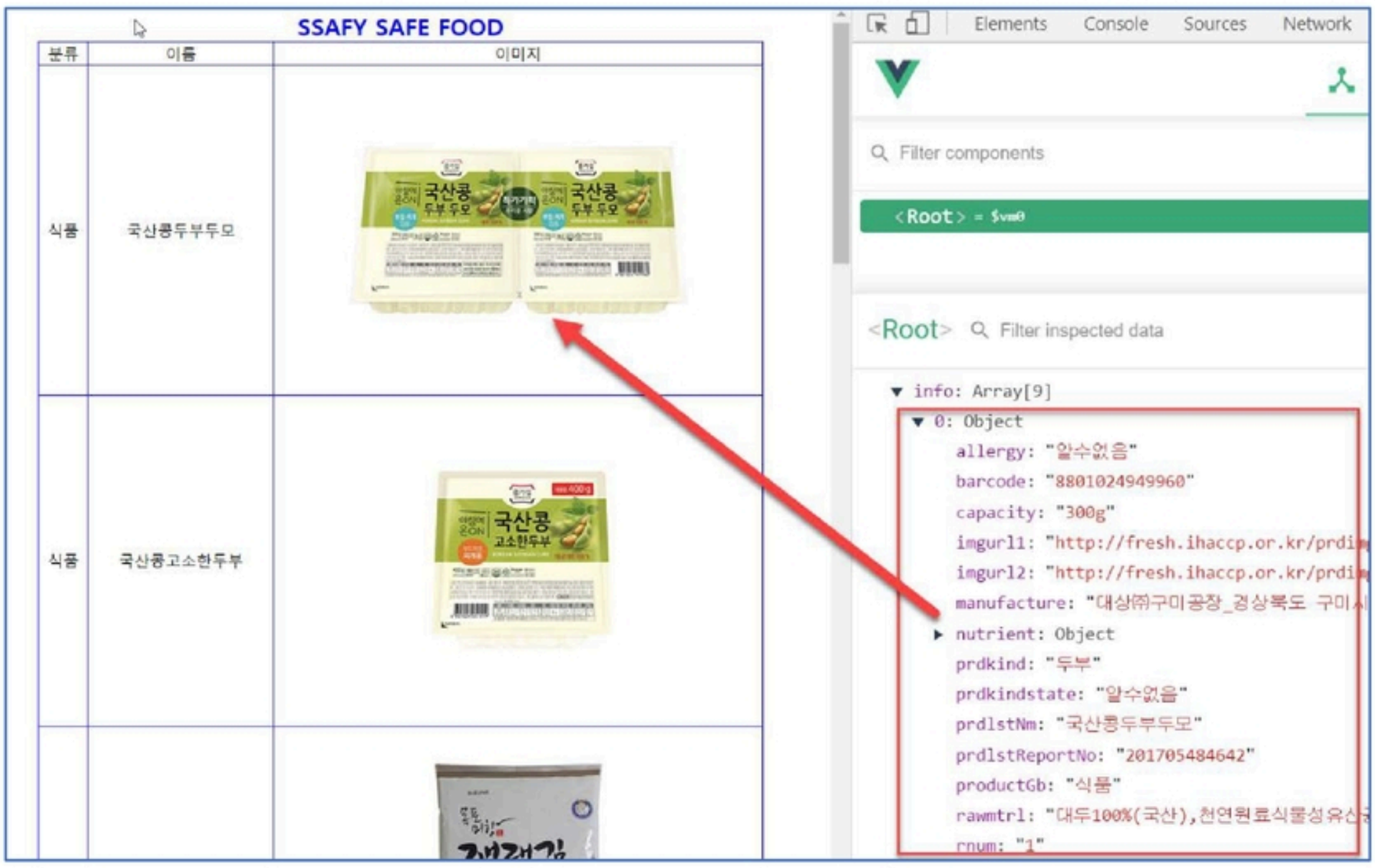
```
{
  "rnum": "1",
  "prdlstReportNo": "201705484642",
  "productGb": "식품",
  "prdlstNm": "국산콩두부두모",
  "imgurl1": "201705484642-1.jpg"
}
```

[소스-index02.html]

```
<div id="app">
  <section v-if="errored">
    <p>Error In AJAX Transport </p>
  </section>
  <section v-else>
  <div v-if="loading">Loading...</div>
  <table id='tid' style='border: 1px'>
    <tr><td>분류</td><td>이름</td><td>이미지</td></tr>
    <div v-else>
      <tr   v-for="emp in info">
        <td v-html="emp.productGb"></td>
         <td v-html="emp.prdlstNm"></td>
         <td> <img v-bind:src="emp.imgurl1"/></td>
       </tr>
    </div>
    </table>
  </section>
</div>
<script type="text/javascript">
  new Vue({
    el: '#app',
```

```
    data () {
      return {
        info: null,
        loading: true,
        errored: false
      }
    },
    mounted () {
      axios
        .get('../food.json')
        .then(response => (this.info = response.data))
        .catch(error => {
          console.log(error);
          this.errored = true;
        })
        .finally(() => this.loading = false)
    }
  })
</script>
```

.then(function(response){ this.info=response.data})
와 .then((response)=>{ this.info=response.data})
또는 .then((response)=>(this.info=response.data))
또는 .then(response=>(this.info=response.data))는 모두 동일한 것 같지만 => 사용하지 않으면 지역변수만 적용되어 this.info 의 값이 변경되지 않습니다. 멤버인 this.info 에 영향(바인딩)을 주기위해서는 =>를 사용해야 합니다.

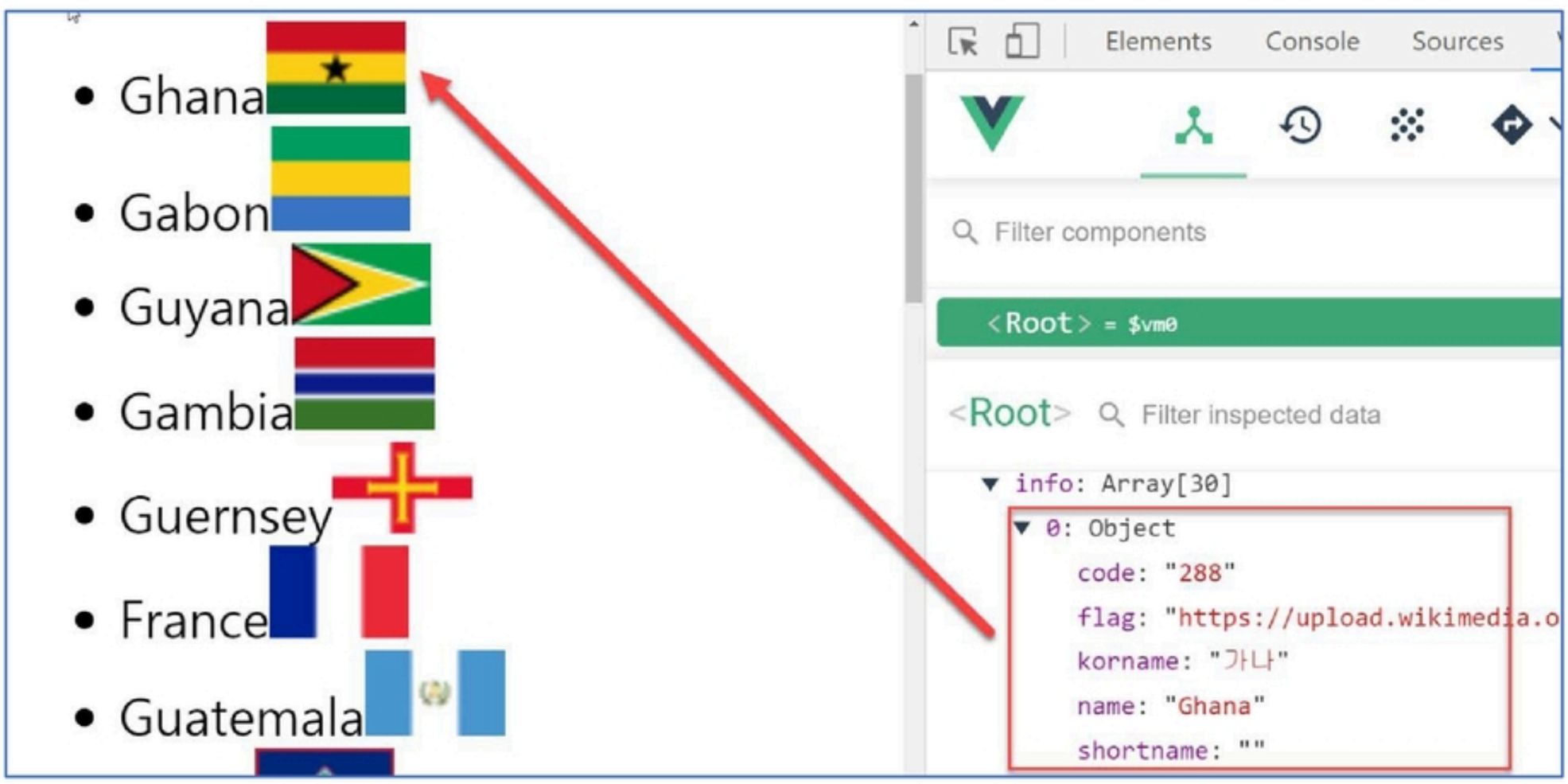


[그림 5.2] 바인딩 된 리스트 데이터를 렌더링하여 화면출력

[소스-index02.html]와 [소스-index03.html]의 차이는 테이블로 렌더링했는가 리스트로 렌더링했는가 하는 것입니다.
{{flag.name}}와 는 같습니다.

[소스-index03.html]

```
<div id="app">
  <section v-if="errored">
    <p>Error In AJAX Transport </p>
  </section>
  <section v-else>
    <div v-if="loading">Loading...</div>
    <ul>
      <li v-for="flag in info">
        <span v-html='flag.name'></span><img v-bind:src="flag.flag"/>
      </li>
    </ul>
  </section>
</div>
```

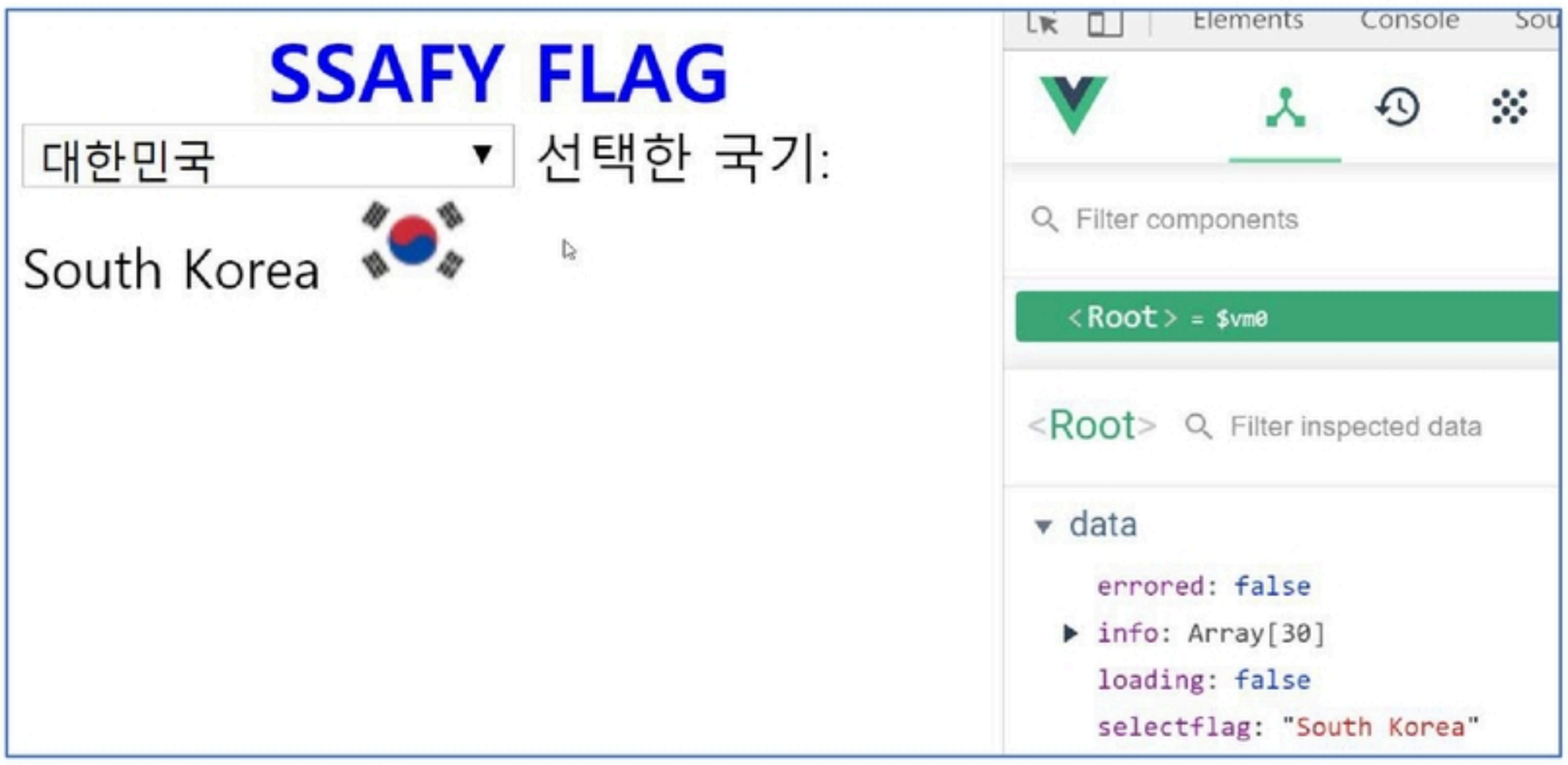


[그림 5.3] 바인딩 된 리스트 데이터를 렌더링하여 화면출력

[소스-index04.html]에서 selectflag 는 뷰 모델의 data 에 선언한 selectflag 과 양방향 바인딩을 했습니다. 그래서 selector 에서 한 개의 국기를 선택하면 선택한 국기의 이름을 <span v-html>에 출력할 수 있습니다. 또한 selector 에서 선택한 국기와 비교하고 선택한 국기의 이미지만 출력할 수 있습니다. <template>에서 전체의 국기와 선택한 국기가 같으면(flag.name===selectflag) 해당 국기를 화면에 보입니다.

[소스-index04.html]

```
<select v-model="selectflag">
  <option v-for="flag in info" v-bind:value="flag.name">
    {{flag.korname}}
  </option>
</select>
<span>선택한 국기: <span v-html='selectflag'></span></span>
<template v-for="flag in info" >
  <img v-show="flag.name==selectflag" v-bind:src="flag.flag"/>
</template>
<script type="text/javascript">
  new Vue({
    el: '#app',
    data () {
      return {
        info: null,
        loading: true,
        errored: false,
        selectflag:' '
      }
    },
  });
</script>
```



[그림 5.4] 셀렉터와 바인딩

info 는 배열([])입니다.
배열에 대한 key 는 index 로 {{key}}는 배열의 인덱스를 출력하고 {{value}}는 키에 대한 밸류인데 여기서는 하나의 국기 정보이므로 {국기정보}가 출력됩니다.

[소스-index05.html]

```
<ul>
  <li v-for="(value, key) in info">
    {{ key }} :: {{ value }}
  </li>
</ul>
```

6. 이벤트 핸들링

6.1 이벤트

6.1.1 자바스크립트 이벤트 핸들링

Vue 에서 자바스크립트 문법을 그대로 사용할 수 있습니다. 자바스트립트처럼 Vue 도 이벤트를 발생시킬 수 있으며 해당 이벤트를 처리할 수도 있습니다. 이벤트 디렉티브는 “v-on:이벤트종류” 를 사용합니다. onclick 은 v-on:click 또는 @click 을 사용합니다.

[소스-index01.html]

```
<div id='first'>
  <div v-html="title"></div>
  <div v-on:click="changeMyName">{{title}}</div>
  <div @click="changeMyName">{{title}}</div>
</div>
<script type="text/javascript">
  var model = {title : '<h2>SSAFY</h2>'};
  var first = new Vue({
    el : '#first',
    data : model,
    methods: {
      changeMyName : function() {
        this.title='<h2>SSAFY HRM SSAFY HRM</h2>';
      }
    }
  });
</script>
```

자바스크립트에서 〈form〉태그안에서 submit 을 했을 때 상황에 따라 submit 을 못하게 해야 하는데 그대로 실행될 때가 있습니다. 이 때는 〈form〉태그 밖으로 버튼을 만들어 자바 스트립트를 이용하여 submit 을 하면 이런 현상을 막을 수 있습니다.

비슷한 원리로 오작동을 막기 위해 이벤트 수식어를 붙일 수 있습니다. 실행된 이벤트가 화면을 다시 로딩하지 못하도록 @submit.prevent="메서드이름"을 사용합니다.
Submit 버튼을 누르면 @submit.prevent 에 등록된 메서드가 실행됩니다.
[소스-index02.html]에서는 addEmp()와 findEmp()가 실행됩니다.

addEmp()는 사원 목록에 새로운 사원을 추가시켜주는 메서드입니다. 폼 태그 데이터에 바인딩 된 값으로 문자열이 입력되지 않으면 아이디와 패스워드를 입력하라고 Alert 창을 띄워주고, 입력이 제대로 들어오면 그 정보를 모델(데이터)의 emp_list 에 저장한 후, 그 목록을 화면에 출력합니다.
input 폼 데이터 태그에 값을 입력하고 엔터키를 치면 submit 이 발생하여 addEmp()를 실행시킵니다.
findEmp()는 사원 번호를 사용해서 사원의 정보를 조회하는 메서드입니다. input 태그에 바인딩 된 값으로 input 태그에 값을 입력하고 엔터키를 치면 findEmp()를 실행시킵니다. 올바른 사원 번호가 입력되지 않으면 자바스크립트 DOM 을 이용하여 input 태그에 입력된 value(잘못된 값을 입력했을 때 보여주기 위한 것) 속성 값을 Alert 에 띄워줍니다. 올바른 사원 번호를 입력 받으면, 그 사원의 정보를 emp_list 에서 찾아서 출력해줍니다.

input 태그에서 v-on:keyup.enter 디렉티브를 사용하고 있는데 keyup 은 키보드 이벤트의 종류 중 하나이며 키보드를 누르고 땔 때 동작합니다. 여기서 enter 는 엔터키를 뜻하며 엔터키를 눌렀다 땔 때 키보드 이벤트가 발생하게 됩니다.

```
〈input v-on:keyup.enter="submit"〉
〈input @keyup.enter="submit"〉
```

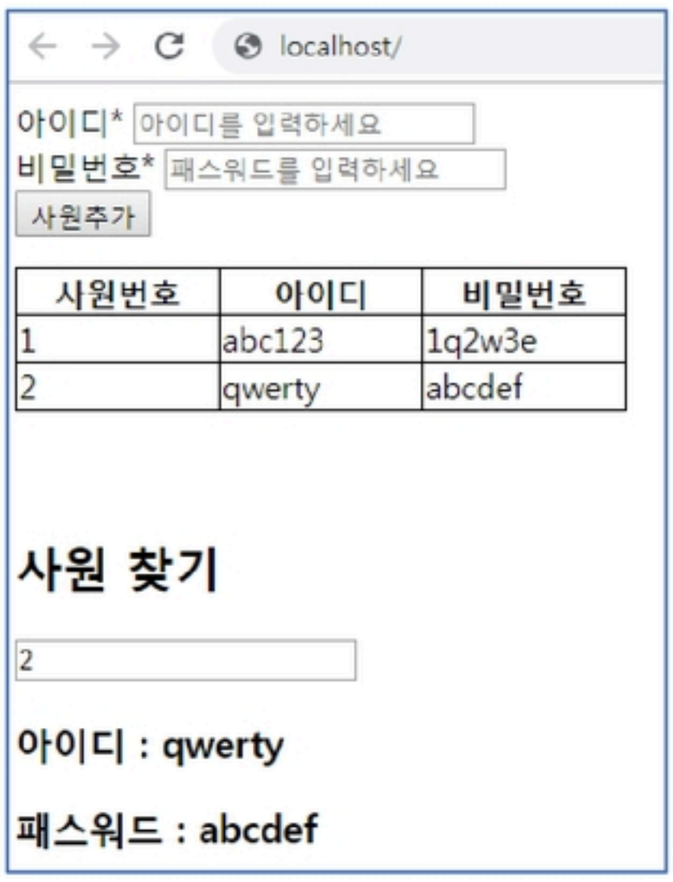
keyup 수식어 별칭의 전체 목록입니다.

- .enter
- .tab
- .delete(“delete”와 “backspace” 키 모두를 캡처합니다.)
- .esc
- .space
- .up
- .down
- .left
- .right

[소스-index02.html]

```
<div id="signup_section">
  <form action="" method="post" @submit.prevent="addEmp">
    아이디*
    <input id="_signId" name="id" type="text" v-model="id" placeholder="아이디를 입력하세요"
/><br />
    비밀번호*
    <input id="_signPw" name="pw" type="password" v-model="pw" placeholder="패스워드를
입력하세요" /><br />
    <button type="submit" name="button">사원추가</button>
  </form>
  <table v-bind:style="{ border:'1px solid black', marginTop: '15px', borderCollapse: 'collapse' }">
    <tr>
      <th v-bind:style="tablestyle">사원번호</th>
      <th v-bind:style="tablestyle">아이디</th>
      <th v-bind:style="tablestyle">비밀번호</th>
    </tr>
    <tbody v-if="emp_list[0]">
      <tr v-for="(e,i) in emp_list">
        <td v-bind:style="tablestyle">{{i+1}}</td>
        <td v-bind:style="tablestyle">{{e.id}}</td>
        <td v-bind:style="tablestyle">{{e.pw}}</td>
      </tr>
    </tbody>
  </table><br><br>
  <h2>사원 찾기</h2>
  <input type="number" @keyup.enter="findEmp">
  <h3 v-cloak>아이디 : {{emp.id}}</h3>
  <h3 v-cloak>패스워드 : {{emp.pw}}</h3>
</div>
<script type="text/javascript">
  var model = {
    tablestyle: {
      border: '1px solid black',
      width: '100px'
```

```
    },
    id: "",
    pw: "",
    emp_list: [],
    emp: { id: "", pw: "" }
  };
  var signup_section = new Vue({
    el: '#signup_section',
    data: model,
    methods: {
      addEmp() {
        if (this.id && this.pw) {
          this.emp_list.push({
            id: this.id,
            pw: this.pw
          });
          this.id = "";
          this.pw = "";
        } else alert("id나 pw를 제대로 입력하세요.")
      },
      findEmp(e) {
        if (this.emp_list[e.target.value - 1]) {
          this.emp.id = this.emp_list[e.target.value - 1].id;
          this.emp.pw = this.emp_list[e.target.value - 1].pw;
        } else alert(`존재하지 않는 사원 번호(${e.target.value})입니다.`)
      }
    }
  })
  Vue.config.devtools = true;
</script>
```



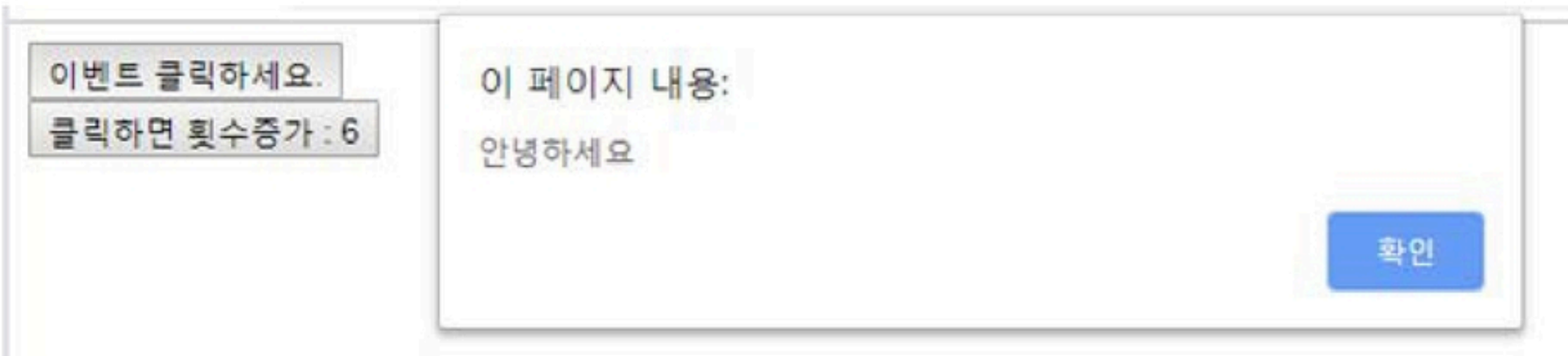
[소스-index03.html]

```
<div id='second'>
  <button @click="hello('안녕하세요')">자바스크립트: 이벤트 클릭하세요.</button>
  <button @click="clickcount+=1">클릭하면 회수증가 {{clickcount}}</button>
</div>
<script type="text/javascript">
  var model = {title : '<h2>SSAFY</h2>', clickcount:0};
  var first = new Vue({
    el : '#second',
    data : model,
    methods: {
      hello(element) {
        this.title=element;
         alert(this.title);
      }
    }
  });
</script>
```

button 태그를 클릭하면 hello()가 호출하고 hello()에 아규먼트로 '안녕하세요'가 입력됩니다.
이 값이 뷰 모델의 title 에 대입되고 tittle 값이 변경됩니다. 이 후 변경된 title 값이 alert 됩니다.
clickcount 는 메서드가 아니고 뷰 모델의 data 에 등록된 값입니다.

Button 을 클릭할 때마다 clickcount 의 값이 1 씩 증가합니다

실행 결과는 다음과 같습니다.



7. 폼 데이터 처리

7.1 리스트 데이터

7.1.1 리스트 데이터 바인딩

디렉티브에서 리스트 데이터와 폼 데이터의 차이를 학습했습니다.
리스트 데이터는 단방향으로 v-html 이나 v-bind 를 사용하고 폼 데이터는 양방향으로 모델뷰의 정의한 data 와 바인딩합니다.
v-model 은 폼데이터 값을 변경했을 때 변경된 값을 화면에 반영합니다.

아래와 같은 JSON 데이터를 읽어서 테이블로 만들겠습니다. 우선 테이블을 리스트 데이터로 v-bind 나 v-html 을 사용하게 됩니다. 배열 [] 안에 여러 개의 정보{}가 있기 때문에 v-for 디렉티브를 사용하여 반복하면서 {{}}나 v-hmtl 바인딩하여 테이블로 렌더링하겠습니다. 중요한 것은 JSON 이 키:밸류의 형식이라는 점입니다. JS 객체이므로 .으로 사용합니다.

[{"id":1,"name":"박구곤","mailid":"sky98","start_date":"1990-03-03","manager_id":0,
"title":"사장","dept_id":118,"salary":5000.0,"commission_pct":0.0}]

[소스-index01.html]는 v-for 디렉티브를 이용하여 테이블로 렌더링 했습니다.
<tr>이 반복되면서 남는 태그로 사용하기 때문에 <template>을 사용하지 않았습니다.
<template>은 실행 후 남지 않는 태그입니다. <tr>안에 <td></td>는 엘리먼트 값이므로 <td v-html></td>를 사용합니다. {{}}가능합니다.
아이디, 이름, 부서, 직책, 연봉을 출력합니다. 연봉은 소수점 두 자리까지 표시하기 위해 필터를 사용하였습니다.

[소스-index01.html]

```
<tr v-for="emp in info">
  <td v-html="emp.id"></td>
  <td v-html="emp.name"></td>
  <td v-html="emp.dept_id"></td>
```

```
  <td v-html="emp.title"></td>
  <td>{{ emp.salary | salarydecimal }}</td>
</tr>
</table>
</div>
```

```
<script type="text/javascript">
new Vue({
  el: '#app',
new Vue({
  el: '#app',
  data () {
    return {
      info: [],
      loading: true,
      errored: false
    }
  },
  filters: {
            salarydecimal (value) {
      return   value.toFixed(2)
    }


  },
```

SSAFY HRM LIST

사원추가 | 이름으로 찾기 | 아이디로 찾기 | 모든사원 보기

사원 아이디	사원명	부서	직책	연봉
1	박구곤	118	사장	5000.00
2	손명성	110	기획부장	3000.00
3	이은선	102	영업부장	3500.00
4	서의홍	101	총무부장	3000.00
5	김심선	118	인사부장	3200.00
6	전지영	110	과장	2400.00
7	김정미	111	과장	2400.00
8	신기웅	112	과장	2100.00

[그림 7.1] <td v-html> 디렉티브를 이용한 바인딩

7.2 폼 데이터

7.2.1 select 폼 데이터 바인딩

폼 데이터는 양방향으로 모델뷰에 정의한 data 와 바인딩합니다.
v-model 은 폼 데이터 값을 변경했을 때 변경된 값을 화면에 반영합니다.
아래 소스는 사원의 아이디와 이름을 JSON 형식의 배열([ ])을 얻은 후, option 에서 이름으로 아이디를 선택할 수 있도록 만듭니다. 〈option value='사원아이디'〉사원이름〈/option〉 형식이 반복되기 때문에 〈option v-for〉 디렉티브를 사용합니다. info 는 배열이므로 배열의 각 요소인 한 개의 JSON 객체를 emp 라고 지정하고 emp.id 로 아이디를 얻습니다. 같은 방법으로 emp.name 을 얻습니다.
select 에서 한 개의 id 를 선택하면 이벤트가 발생합니다. @change 는 v-on:chanage 와 동일하며 change 이벤트가 발생하면 findbyid 메서드가 호출됩니다.
자바스크립트가 다 로딩 될 때까지 내용을 안보이게 하기 위해 v-cloak 을 붙입니다. 이벤트 등이 발생할 때 내용이 잠깐 보였다가 로딩이 되고 해당 값으로 변경되기 때문에 이 현상을 해소해줍니다.

```
〈select name='id' v-model="cid" @change='findbyid' v-cloak〉

  〈option v-for="emp in info" v-bind:value="emp.id"〉

    {{ emp.name }}

  〈/option〉

〈/select〉
```

[그림 7.2] <option v-html>, <select v-model> 디렉티브를 이용한 바인딩

[소스-index02.html]

```
<div class='search_box'>
 <select name='id' v-model="cid" @change='findbyid' v-cloak>
  <option v-for="emp in info" v-bind:value="emp.id">
   {{ emp.name }}
  </option>
</select>
</div>
<table class='list_table'>
<tr>
<td v-html="cemp.id"></td>
  <td v-html="cemp.name"></td>
  <td v-html="cemp.dept_id"></td>
  <td v-html="cemp.title"></td>
  <td>{{ cemp.salary | csalarydecimal }}</td>
</tr>
</table>
</div>
```

```
<script type="text/javascript">
new Vue({
  el: '#app',
  data () {
    return {
      info: [],
      loading: true,
      errored: false ,
      cid:'',
      cemp:{}
    }
  },
filters: {
            salarydecimal (value) {
```

```
        return value.toFixed(2)
    }
  },
```

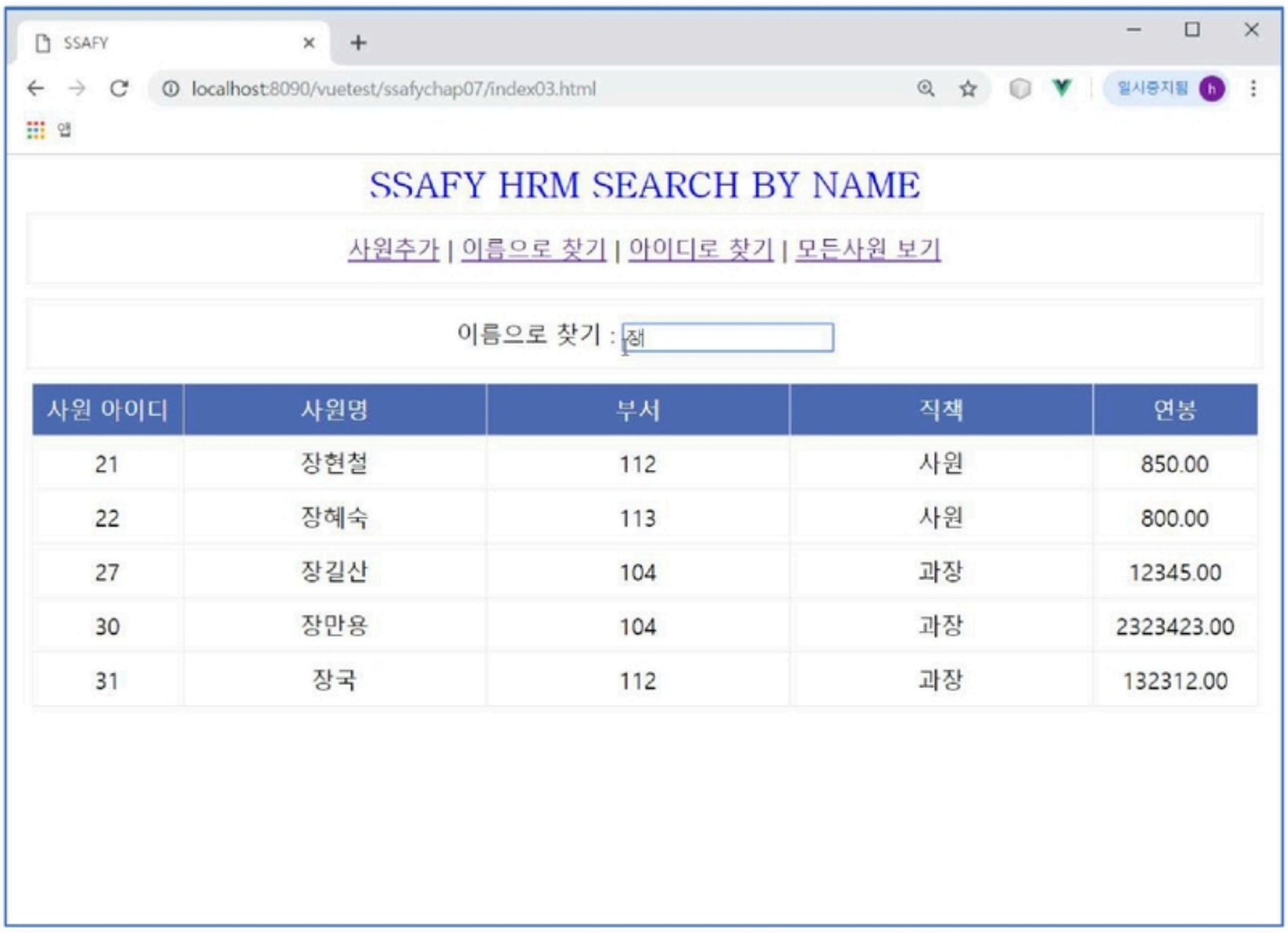
7.2.2 input 폼 데이터 바인딩

폼 데이터는 양방향으로 모델뷰의 정의한 data 와 바인딩합니다.
v-model 은 폼 데이터 값을 변경했을 때 변경된 값을 화면에 반영합니다.
이번에는 〈input〉 폼 데이터 태그에 사원의 이름의 일부분으로 사원을 찾아보겠습니다. input 폼 데이터 태그에 이름 일부분을 입력하고 손을 올릴 때 이벤트가 발생하도록 @keyup 을 등록했습니다.
keyup 하면 searchname 메서드가 실행됩니다.
Input 에 입력한 이름은 cname 으로 뷰 모델 data 에 등록되었기 때문에 searchname 에서 this.cname 으로 사용할 수 있습니다.

[소스-index03.html]

```
<div class='search_box'>
이름으로  찾기 : <input type="text" name='name' v-model="cname"
                @keyup='searchname' v-cloak/>
</div>
<tr v-for="emp in cemps">
  <td v-html="emp.id"></td>
  <td v-html="emp.name"></td>
  <td v-html="emp.dept_id"></td>
  <td v-html="emp.title"></td>
  <td>{{ emp.salary | salarydecimal }}</td>
</tr>
<script type="text/javascript">
new Vue({
  el: '#app',
  data () {
    return {
      info: [],
```

```
        loading: true,
        errored: false ,
        cname:'',
        cemps:[]
    }
},
```



[그림 7.3] <input v-model> 디렉티브를 이용한 바인딩

7.2.3 <form>태그 내부에서의 폼 데이터 바인딩

<form>태그는 브라우저에서 서버로 데이터(파라메터)를 보낼 때 사용합니다.
<form>태그는 input, select, option, checkbox, radio, textarea 등으로 구성한 후 submit 버튼을 누르면 정해진(action)으로 파라메터를 보냅니다.

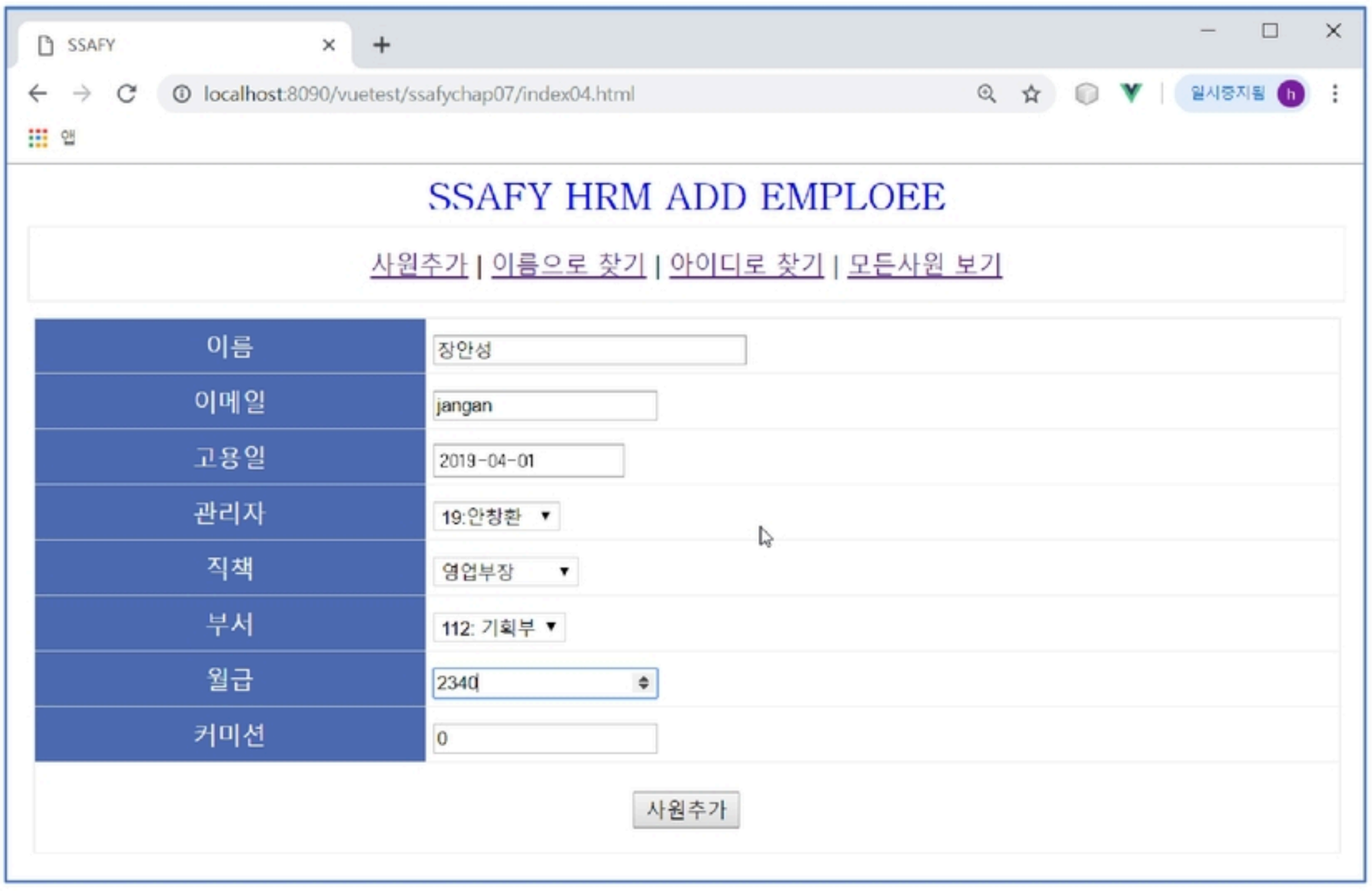
여기서는 <form> 태그의 폼 태그 데이트를 { }내부에 입력 후 JSON 으로 보냅니다. 아래와 같이 "키: v-model 에 바인딩 해서 얻은 값" 으로 JSON 을 만든 후 서버로 보냅니다.

```
{
  commission_pct: this.ccommission_pct,
  dept_id: this.cdept_id,
  mailid: this.cmailid,
  manager_id: this.cmanager_id,
  name: this.cname,
  salary: this.csalary,
  start_date:this.cstart_date,
  title: this.ctitle
}
```

[소스-index04.html]

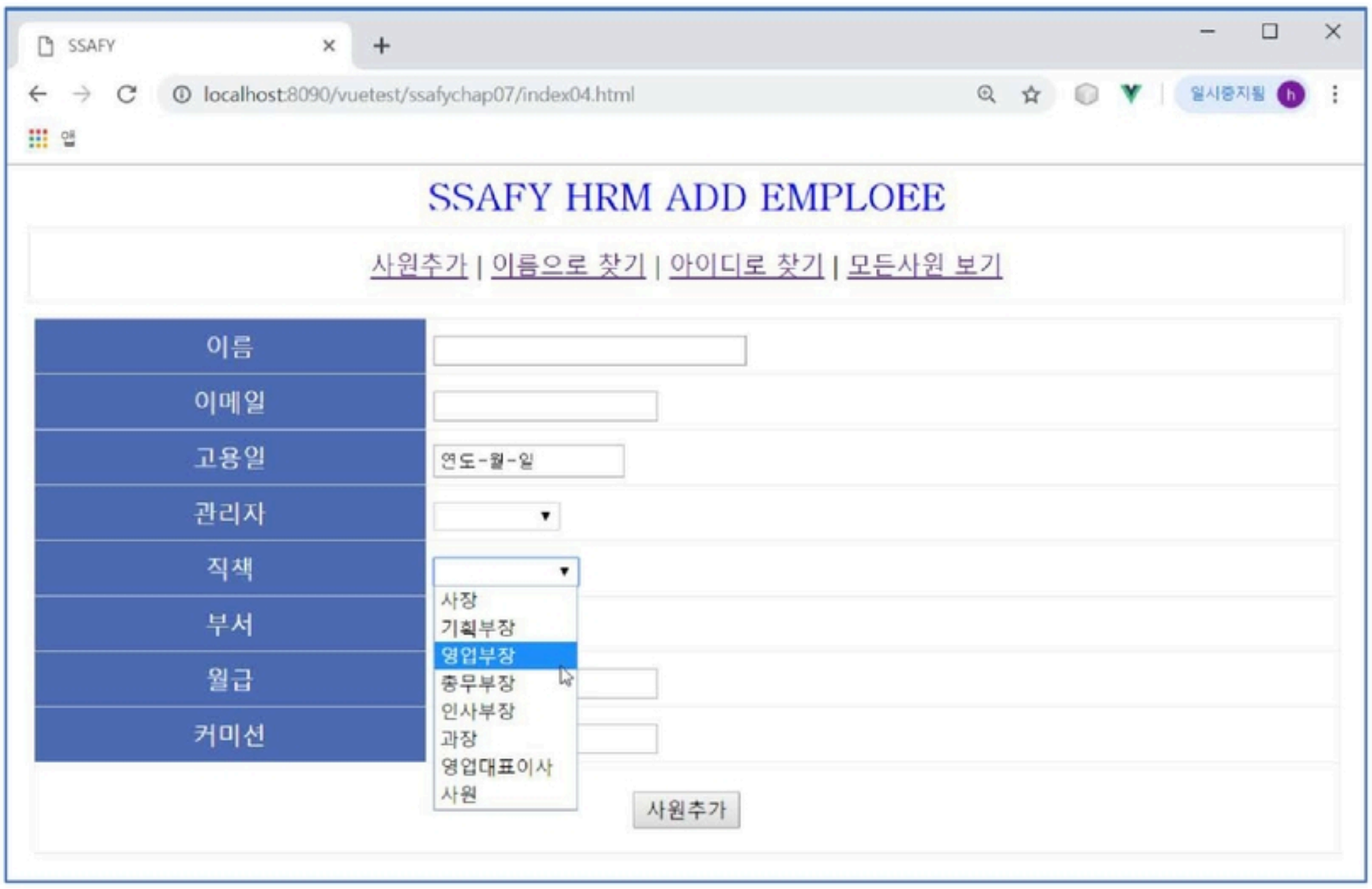
```
<form action="" method="post" id="_frmForm" name="frmForm"
@submit.prevent="addEmployee">
<table class="content_table">
  <colgroup>
    <col style="width:30%;" /><col style="width:70%;" />
  </colgroup>
  <tr>
    <th>이름</th>
    <td>
      <input data-msg="이름" type="text" name="name" id="_name" size="30" v-
model="cname"/>
```

```
    </td>
  </tr>
  <tr>
    <th>이메일</th>
    <td>
      <input data-msg="이메일" type="text" name="mailid"   id="_mailid" size="20" v-model="cmailid" />
    </td>
  </tr>
new Vue({
  el: '#app',
  data () {
    return {
      info: null,
      loading: true,
      errored: false,deps:null,titls:null,
      ccommission_pct:0,
      cdept_id:0,
      cmailid:'',
      cmanager_id:0,
      cname:'',
      csalary:0,
      cstart_date:'',
      ctitle:''
    }
  },
```



[그림 7.4] 폼 데이터 처리

[그림 7.5]에서 직책, 관리자, 부서는 select 의 바인딩을 이용하여 만든 것 입니다.
자연스럽게 AJAX 형태로 개발하게 됩니다.



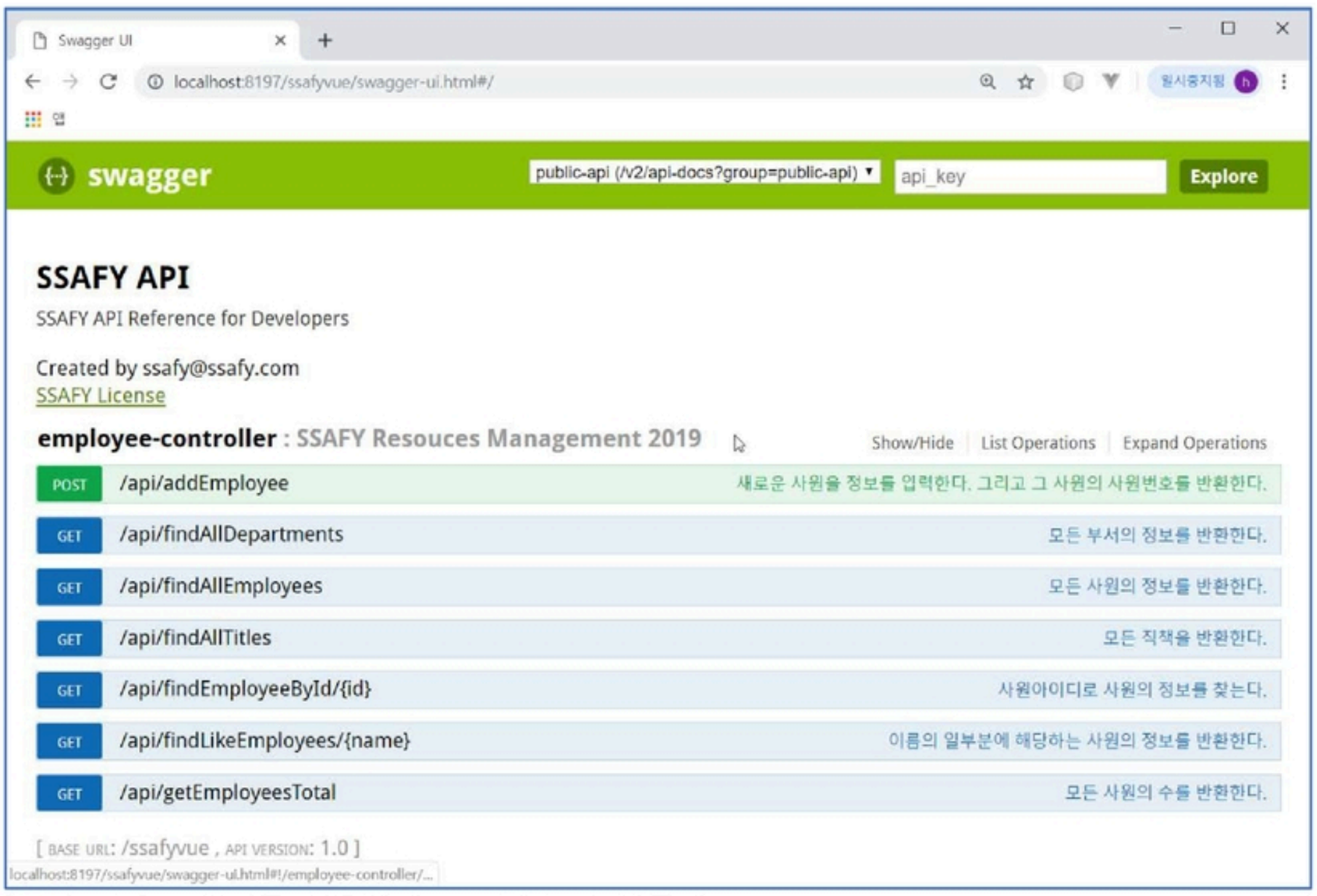
[그림 7.5] 여러 개의 바인딩

8. Vue 를 이용한 애플리케이션

8.1 스프링 부트를 이용한 Restful 서비스 구축

8.1.1 Restful 서비스 API

뷰를 잘 사용하기 위해서는 데이터의 특징을 이해할 필요가 있습니다.
AJAX 와 Restful 서비스는 같이 사용하기에 좋습니다. 요사이 많은 회사들이 빨리 개발하면서 재활용을 높이기 위한 노력을 하고 있습니다. 그 중에 한가지가 뷰+Restful 입니다.
스프링 부트로 Restful 을 만들고 Restful API 를 만듭니다. Restful API 는 만들기가 까다롭습니다. 그래서 스웨거를 사용합니다. 스웨거는 오픈소스로 어노테이션을 이용하여 Restful API 를 쉽게 만들 수 있습니다.



[그림 8.1] 스웨거를 이용하여 Restful API 만들기

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8197/ssafyvue/api/findAllEmployees'
```

Request URL

```
http://localhost:8197/ssafyvue/api/findAllEmployees
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body


```
[
  {
    "id": 1,
    "name": "박구곤",
    "mailid": "sky98",
    "start_date": "1990-03-03",
    "manager_id": 0,
    "title": "사장",
    "dept_id": 118,
    "salary": 5000,
    "commission_pct": 0
  },
```


[그림 8.2] 서비스를 호출하는 방법과 결과

8.1.2 Restful 서비스 API 만들기

스프링 부트를 이용하여 Restful 서비스를 만듭니다. 스프링의 @responseBody 를 이용하면 파라메터가 남게 되어 불편하고 GET/POST 로만 만들 수 있어서 불편합니다. 그래서 스프링 부트로 독립적인 서비스를 만드는 것이 좋습니다. Restful 은 데이터만 JSON 으로 서비스할 때 화면은 전혀 신경쓰기 않기 때문에 개발이 매우 빠르고 편리합니다.

다른 서비스에서 구축한 서비스를 사용할 때 다른 쪽에서 서비스를 사용할 수 있도록 CrossOrigin(origins = {"*"}, maxAge = 6000)을 붙입니다.
레스트 서비스를 하기 위해 @RestController 를 붙입니다. 스웨거 api 를 위해서@Api(value="SSAFY", description="SSAFY Resouces Management 2019")를 붙입니다.
@ApiOperation(value = "모든 사원의 정보를 반환한다.", response = List.class)는 . 스웨거 api 보여줄 타입입니다.
return new ResponseEntity<List<EmployeeDto>>(emps, HttpStatus.OK); Restful 의 저장할 정보를 json 으로 만듭니다.

[소스-EmployeeController.java]

```
CrossOrigin(origins = {"*"}, maxAge = 6000)
@RestController
@RequestMapping("/api")
@Api(value="SSAFY", description="SSAFY Resouces Management 2019")
public class EmployeeController {
public static final Logger logger =
        LoggerFactory.getLogger(EmployeeController.class);
@Autowired
private EmployeeService   employeeService;
@ApiOperation(value = "모든 사원의 정보를 반환한다.", response = List.class)
@RequestMapping(value = "/findAllEmployees", method = RequestMethod.GET)
 public ResponseEntity<List<EmployeeDto>> findAllEmployees() throws Exception {
 logger.info("1------------findAllEmployees----------------"+new Date());
 List<EmployeeDto> emps = employeeService.findAllEmployees();
 if (emps.isEmpty()) {
    return new ResponseEntity(HttpStatus.NO_CONTENT);
 }
 return new ResponseEntity<List<EmployeeDto>>(emps, HttpStatus.OK);
}
```

외부에서 들어오는 파라메터 부분에 ?이나 &를 사용하지 않습니다.

http://localhost:8197/ssafyvue/api/findEmployeeById/1

와 같이 /1 은 /{1}이되고 @PathVariable 이 됩니다. 결국 아이디로 해당 사원을 찾는 것이 됩니다.


```
{
    "id": 1,
    "name": "박구곤",
    "mailid": "sky98",
    "start_date": "1990-03-03",
    "manager_id": 0,
    "title": "사장",
    "dept_id": 118,
```

```
    "salary": 5000,
    "commission_pct": 0
}
```

과 같은 결과를 얻게 됩니다. 이런 방법과 동일하게 필요한 모든 서비스르 만듭니다.

GET 은 SELECT 용, POST 는 INSERT, PUT 은 UPDATE, DELETE 는 DELETE 용으로 결정되어 있어 혼동이 없습니다.

```
@ApiOperation(value = "사원아이디로 사원의 정보를 찾는다.", response = EmployeeDto.class)
@RequestMapping(value = "/findEmployeeById/{id}",method = RequestMethod.GET)
public ResponseEntity<EmployeeDto>
                    findEmployeeById(@PathVariable int id) throws Exception {
 logger.info("1------------findAllEmployees--------------"+new Date());
 EmployeeDto emp = employeeService.findEmployeeById(id);
 if (emp==null || emp.getId()==0) {
        return new ResponseEntity(HttpStatus.NO_CONTENT);
 }
 return new ResponseEntity<EmployeeDto>(emp, HttpStatus.OK);
}
```

Response Content Type */* ▾

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	1	id	path	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out! Hide Response

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8197/ssafyvue/api/findEmployeeById/1'
```

Request URL

```
http://localhost:8197/ssafyvue/api/findEmployeeById/1
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
{
  "id": 1,
  "name": "박구곤",
  "mailid": "sky98",
  "start_date": "1990-03-03",
  "manager_id": 0,
  "title": "사장",
```

[그림 8.3] 서비스를 호출하는 방법과 결과

8.2 Vue에서 서비스 사용하기

8.2.1 axios 이용하기

axios 는 Vue 의 라이브러리로 비동기를 위한 것입니다.

<script src="https://unpkg.com/vue"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.18.0/axios.js"></script>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

사용하려는 API 에서 제공하는 데이터 형식을 확인합니다.

http://localhost:8197/ssafyvue/api/findAllEmployees


```
[{
"id": 1,
"name": "박구곤",
"mailid": "sky98",
"start_date": "1990-03-03",
"manager_id": 0,
"title": "사장",
"dept_id": 118,
"salary": 5000,
"commission_pct": 0
},
{
"id": 2,
"name": "손명성",
"mailid": "2xxsig",
"start_date": "1990-03-08",
"manager_id": 1,
"title": "기획부장",
"dept_id": 110,
"salary": 3000,
"commission_pct": 0
}]
```


바인딩 할 데이터들을 선언합니다.

메서드에서 바인딩 할 데이터들은 함수형식이므로 data(){ return { 바인딩 데이터} }를 선언합니다.
문자열은 ‘’, 배열은 []로 초기화 시킵니다.

[소스-index01.java]

```
<script type="text/javascript">
  new Vue({
    el: '#app',
    data () {
      return {
        info: [],
        loading: true,
        errored: false
      }
    },
    filters: {
      salarydecimal (value) {
        return value.toFixed(2)
      }
    },
    mounted () {
      axios
        .get('http://localhost:8197/ssafyvue/api/findAllEmployees')
        //.get('./emp.json')
        .then(response => (this.info = response.data))
        .catch(error => {
          console.log(error)
          this.errored = true
        })
        .finally(() => this.loading = false)
    }
  })
</script>
```

axios 에 레스트 풀 API 의 경로를 입력합니다. 중요한 점은 Vue 의 axios 로 JSON 데이터를 받으면 키로 data 가 붙는 다는 것입니다.
{data: Restful 서비스 데이터} 형식이 됩니다.
response.data 는 앞에서 파악한 배열([])이 됩니다. 이것을 info[]에 저장합니다. 이것은 바인딩 된 것으로 화면에서 {{}}나 v-html, v-bind, v-model 로 바인딩하여 사용할 수 있습니다.

9. Component

Component란 UI를 구성하는 각각의 부품들이라고 생각하면 됩니다. UI상에 같은 부분을 재사용하고 싶은 경우 component화 해서 재사용할 수 있는 것이 component의 장점입니다. 대부분의 애플리케이션들이 트리 형태의 component들로 구성되어 있습니다.

예를 들면, 웹 페이지의 헤더, 사이드 바, 본문에 해당되는 component를 만들 수도 있고, 헤더 부분은 네비게이션 링크들, 본문 부분은 포스트들 등 각각의 component들은 또 다른 component들로 이루어집니다.

Vue.js 에서는 이 component라는 강력한 기능을 제공합니다. Vue.js의 component는 재사용 가능한 Vue 인스턴스입니다. 따라서 앞에서 사용한 data, methods, filter, computed 등의 속성을 사용할 수 있습니다.

Component를 선언할 때는 Vue.component(컴포넌트이름, {template, data(){}, methods:{}}) 형식을 사용합니다.

Component의 이름은 우리의 마음대로 지정할 수 있고, component의 기본적인 모양새를 template 이라는 속성을 통해 정해주어야 합니다.

9.1 템플릿 만들기

Vue의 템플릿이란, 위에서 정의한 대로 component의 기본적인 모양새를 만들어주는 역할을 하는데, 이 템플릿은 html 엘리먼트들을 사용해서 원하는 대로 정의해 줍니다.

9.1.1 템플릿 선언하기

컴포넌트를 선언할 때 그 안에 템플릿의 내용을 선언해줄 수 있습니다.

```
<script type="text/javascript">
    Vue.component('firstcomponent',{
        template: '<div>최상위 태그는 반드시 하나여야 합니다.</div>',
        data(){}
    }
    ….
</script>
```

하지만 위 방식은 템플릿의 내용이 많아질수록 사용하기가 힘들어집니다. 템플릿의 내용을 선언해줄 수 있는 다른 방법은 바로 “text/x-template” type 의 script 태그를 사용하는 방법입니다.
사용할 때는 <script type="text/x-template" id="템플릿아이디"> 형태로 이용합니다.
text/x-template 은 Vue 에서 약속한 템플릿 지정 문서 타입(MIME)이라 Vue 에서만 렌더링 됩니다.
템플릿 내부는 반드시 루트 엘리먼트를 가져야 합니다. 보통 태그로 시작합니다.
컴포넌트를 선언할 때 template 속성에 앞에서 선언한 스크립트 태그 <script type="text/x-template" id="템플릿아이디">에 지정한 id 를 셀렉터 형식으로 입력해 줍니다.

[소스-index00.html]

```
<div id='first'>
  <firstcomponent></firstcomponent>
</div>
<script type="text/x-template" id="atemplate">
  <div><h1>최상위 태그는 반드시 하나여야 합니다. <h1></div>
</script>
<script type="text/javascript">
  Vue.component('firstcomponent',{
    template: '#atemplate'
  });
  var first = new Vue({
```

```
    el : '#first'
  });
  Vue.config.devtools = true;
</script>
```

9.1.2 자바스크립트 분리

자바스크립트 코드가 너무 길어져서 보기 힘들어질 때에는, 자바스크립트의 원하는 부분들을 나눠 자바스크립트 파일로 저장해서 독립적으로 분리시킬 수 있습니다. [소스-index01_1.html]는 위의 예제와 똑같이 x-template 방식을 활용하여 템플릿을 정의하였고 Vue 인스턴스 생성 부분을 자바스크립트 [소스-listhrm.js]로 분리시킨 것입니다. 템플릿만 있어서 보기에 편리합니다.

[소스-index01_1.html]

```
<div id='app'>
    <listhrm></listhrm>
</div>
<script type="text/x-template" id="listhrmtemplate">
<div>
    <div class='headtitle'><h2>SSAFY HRM LIST</h2></div>
    <div class='search_box'>
        <a href='./index01.html'>모든사원 보기</a> |
        <a href='./index02.html'>아이디로 찾기</a> |
        <a href='./index03.html'>이름으로 찾기</a> |
        <a href='./index04.html'>사원추가</a>
    </div>
    <div>
        <table class='list_table'>
            ...
        </table>
    </div>
<div>
</script>
<script type = "text/javascript" src = "listhrm.js"></script>
```

[소스-index01_2.html]는 x-template 문서 타입을 사용하면서 뷰 컴포넌트의 data 속성을 활용하는 예제입니다. 이렇게 각각의 컴포넌트마다 data 속성을 따로 가질 수 있고 템플릿을 통해 data 를 활용할 수 있습니다. Vue 인스턴스는 이러한 컴포넌트를 여러 개를 가질 수 있으며 이러한 방식을 이용해 SPA(Single Page Application)를 구현할 수 있습니다.

[소스-index01_2.html]

```
<script type="text/x-template" id="listhrmtemplate">
  ...
  <tr v-for="emp in info">
    <td v-html="emp.id"></td>>
    <td v-html="emp.title"></td>
    <td>{{ emp.salary | salarydecimal }}</td>
  </tr>
</script>
<script type="text/javascript">
  var listhrm = Vue.component('listhrm',{
    template: '#listhrmtemplate',
    data( ){
      return {
        info: [],
        loading: true,
        errored: false
      }
    },
    mounted ( ) {
      axios
        .get('http://localhost:8197/ssafyvue/api/findAllEmployees')
        .then(response => (this.info = response.data))
        .catch(error => {
          console.log(error)
          this.errored = true
        })
        .finally(() => this.loading = false)
    }
  });
  new Vue({
```

```
    el: '#app'
  })
</script>
```

9.2 컴포넌트

9.2.1 컴포넌트 선언하기

작은 애플리케이션에서는 문제가 발생하지 않지만 여러 부분에서 겹치고 반복되는 것을 볼 수 있습니다.
컴포넌트는 겹치고 반복되는 것을 압축해서 재사용할 수 있도록 만든 코드입니다.
데이터만 다르고 화면이 동일할 때, 같은 기능이 많을 때 사용하면 좋습니다.
Vue.component(컴포넌트이름, {템플릿이름 ,data(){}, methods:{}})형식을 사용합니다.
Vue 인스턴스에 선언한 것과 동일합니다. 결국 Vue 인스턴스도 컴포넌트였던 것입니다.

컴포넌트에서 data 속성을 사용할 때 주의해야 할 점이 있는데, data 를 정의할 때 함수형으로 return 해주어야 한다는 점입니다. 이렇게 해주어야 같은 컴포넌트를 여러 번 재활용 했을 때 생성된 각 인스턴스가 각각 독립된 data 객체를 유지할 수 있습니다.
함수형으로 선언하지 않고 기존의 방식으로 선언을 하면 생성된 같은 컴포넌트의 여러 인스턴스들이 같은 객체를 참조하게 됩니다. 또한 함수형으로 사용할 때에도 직접 객체를 return 해주지 않고 외부 객체를 return 해주어도 같은 객체를 참조하게 됩니다.

예를 들어, 카운터 기능의 컴포넌트를 만들 때 외부 객체를 데이터로써 사용하면 다음과 같은 오류가 발생합니다.

[소스-index00_1.html]

```
<div id="example-2">
    <counter></counter>
    <counter></counter>
    <counter></counter>
</div>
<script type="text/x-template" id="counter">
    <button v-on:click="count += 1">{{ "카운트 : "+count }}</button>
</script>
<script type="text/javascript">
    let data = {
        count: 0
    };
    Vue.component('counter', {
        template: '#counter',
```

```
        data(){
            return data;
        }
    });
    new Vue({
        el: '#example-2'
    });
    Vue.config.devtools = true;
</script>
```

카운트 : 4 카운트 : 4 카운트 : 4

모든 인스턴스들이 같은 객체를 참조하기 때문에 data 값이 모든 인스턴스에 동시에 적용되어 카운트가 동시에 증가하는 것을 확인할 수 있습니다. 이것을 방지하기 위해 인스턴스마다 각자 객체를 return 해줍니다.(return 부분 수정)

[소스-index00_2.html]

```
Vue.component('counter', {
        template: '#counter',
        data(){
            return {
                count: 0
            }
        }
    });
```

카운트 : 8 카운트 : 4 카운트 : 3

이제 각 인스턴스가 서로 다른 객체를 사용하기 때문에 카운트도 각각 증가하는 것을 확인할 수 있습니다.

※ data(){} 형태는 객체에 내부 함수를 정의할 때 사용할 수 있는 생략법입니다. ES6(2015 년~)부터 객체 내부함수 선언 시 data: function(){} 대신 data(){}처럼 생략형으로 선언할 수 있게 되었습니다.

[소스-index01.html]

```
<script type="text/javascript">
var listhrm = Vue.component('listhrm',{
      template: '#listhrmtemplate',
      data(){},
      methods:{}
});
new Vue({
   el: '#app',
   data: {
              currentview: 'listhrm'
    }
})
</script>
```

SSAFY HRM LIST

사원추가 | 이름으로 찾기 | 아이디로 찾기 | 모든사원 보기

사원 아이디	사원명	부서	직책	연봉
1	박구곤	118	사장	5000.00
2	손명성	110	기획부장	3000.00
3	이은선	102	영업부장	3500.00
4	서의홍	101	총무부장	3000.00
5	김심선	118	인사부장	3200.00
6	전지영	110	과장	2400.00
7	김정미	111	과장	2400.00
8	신기웅	112	과장	2100.00
9	이상헌	113	과장	2300.00
10	최순철	113	과장	2300.00
11	최정선	102	영업대표이사	2450.00
12	안은경	103	영업대표이사	2500.00
13	김석우	104	영업대표이사	2650.00

[그림 9.1] 사원 목록 리스트 컴포넌트

[그림 9.2] 사원 추가 컴포넌트

9.3 컴포넌트 통합하기

9.3.1 SPA(Single Page Application)

웹 프로그래밍은 보통 *CRUD 에 대하여 각 화면을 갖게 됩니다. 즉, 어떤 변화가 생겼을 때에 따른 화면이 각각 필요하게 된다는 말입니다. 따라서, 그에 따른 여러 개의 화면이 필요하게 됩니다.

예를 들어, 한 페이지에서 어떠한 데이터를 요청하고 수신한 데이터를 가지고 다른 페이지로 이동한 뒤 출력합니다. 또한 이 데이터를 가공해 다른 페이지로 이동하여 사용합니다. 이런 식으로 여러 페이지를 이동하며 CRUD 작업을 하게 된다면 프로세스가 굉장히 복잡해 질 수가 있습니다. 만약, 한 페이지에서 이러한 CRUD 작업을 모두 수행 할 수 있다면 매우 편리할 것입니다.

여기서 생겨난 개념이 바로 SPA 입니다. SPA 는 Single Page Application 의 약자로 말 그대로 페이지를 하나만 사용하여 애플리케이션을 구현한 것입니다. 이러한 SPA 는 CRUD 작업을 한 페이지에서 모두 수행 할 수 있기 때문에 웹이지만 마치 애플리케이션처럼 동작할 수 있습니다. 이러한 이유 덕에 SPA 라 불리게 된 것이기도 합니다.
SPA 는 컴포넌트화 되어있기 때문에 상대적으로 유지보수가 쉽고 개발속도가 빠릅니다. 또한 한 페이지 내에서 변화가 필요한 컴포넌트 부분만 새로 렌더링 되기 때문에 기존 웹에 비해서 화면 전환이 신속하게 이루어질 수 있다는 장점이 있습니다.
반대로 애플리케이션의 초기 구동 시 시간이 상대적으로 오래 걸린다는 단점도 존재합니다. 따라서 개발하려는 서비스의 특성에 따라 SPA 로 구현할 지의 여부를 판단할 필요가 있습니다.

*CRUD
대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리 기능인 Create(생성), Read(읽기), Update(갱신), Delete(삭제)를 묶어서 일컫는 말입니다.

아래 예제는 통합하려는 인사관리는 사원의 목록을 보이고, 한 사원을 선택하면 해당 사원의 정보를 보입니다. 그리고 사원을 추가할 수 있고 수정, 삭제할 수 있습니다. 아이디와 이름으로 찾을 수 있습니다.

Vue 인스턴스의 data 에 선언한 currenttitle 과 바인딩 되어서 화면이 이동할 때 현재 화면의 제목을 출력합니다. 사원추가를 클릭하면 @click 이벤트가 발생하여 showlist(숫자)를 호출합니다. 각 번호는 사원추가, 찾기, 목록을 알려주는 숫자입니다. 현재화면을 알려주는 currentview 를 바인딩 시키고 이것을 이용하여 해당 컴포넌트를 보입니다. 여러 컴포넌트를 <component></component>에 보여서 한 페이지에서 CRUD 를 모두 할 수 있습니다.

[소스-index05.html]

```
<div id='app'>
  <div class='headtitle'><h2 v-html='currenttitle'></h2></div>
  <div class='search_box'>
    <a href='#' @click='showlist(1)'>사원추가</a> |
    <a href='#' @click='showlist(3)'>이름으로 찾기</a> |
    <a href='#' @click='showlist(2)'>아이디로 찾기</a> |
    <a href='#' @click='showlist(0)'>모든사원 보기</a>
  </div>
  <component v-bind:is = "currentview"></component>
</div>
```

listhrmtemplate 템플릿은 사원의 목록을 테이블로 렌더링합니다.

한 사원을 선택하면 @click 클릭이벤트가 발생하고 show_detail(사원아이디)를 호출하여 해당 사원의 상세정보를 보입니다.

[소스-index06.html]

```
<script type="text/x-template" id="listhrmtemplate">
  <div>
    <div>
      <table class='list_table'>
        <col width="10%"><col width="20%">
        <col width="20%"><col width="20%"><col width="10%">
        <tr>
          <th>사원 아이디</th>
          <th>사원명</th>
          <th>부서</th>
          <th>직책</th>
          <th>연봉</th>
        </tr>
        <tr v-for="emp in info">
          <td v-html="emp.id" @click="show_detail(emp.id)"></td>
          <td v-html="emp.name"></td>
          <td v-html="emp.dept_id"></td>
          <td v-html="emp.title"></td>
```

```
            <td>{{ emp.salary | salarydecimal }}</td>
          </tr>
        </table>
      </div>
    </div>
</script>
```

Idhrmtemplate는 사원의 이름 중에서 선택을 한 명을 선택하면 해당 사원의 아이디를 이용하여 해당 사원의 정보를 보입니다. 사원의 이름을 변경하면 @change 이벤트가 발생하여 findbyid()메서드가 실행됩니다. 선택된 아이디를 이용하여 사원의 정보를 보입니다.

[소스-index07.html]

```
<script type="text/x-template" id="idhrmtemplate">
  <div>
    <div class='search_box'>
      <select name='id' v-model="cid" @change='findbyid' v-cloak>
        <option v-for="emp in info" v-bind:value="emp.id">
          {{ emp.name }}
        </option>
      </select>
    </div>
    <table class='list_table'>
      <col width="10%"><col width="20%">
      <col width="20%"><col width="20%"><col width="10%">
      <tr>
        <td v-html="cemp.id"></td>
        <td v-html="cemp.name"></td>
        <td v-html="cemp.dept_id"></td>
        <td v-html="cemp.title"></td>
        <td>{{ cemp.salary | csalarydecimal }}</td>
      </tr>
    </table>
  </div>
</script>
```

namehrmtemplate 는 사원의 이름 일부분을 입력하면 @keyup 이벤트가 발생하며 이름에 해당하는 사원들을 보입니다.

[소스-index08.html]

```
<script type="text/x-template" id="namehrmtemplate">
  <div>
    <div class='search_box'>
      이름으로  찾기 : <input type="text" name='name' v-model="cname"
@keyup='searchname' v-cloak/>
    </div>
    <table class='list_table'>
      <col width="10%"><col width="20%">
      <col width="20%"><col width="20%"><col width="10%">
      <tr>
        <th>사원 아이디</th>
        <th>사원명</th>
        <th>부서</th>
        <th>직책</th>
        <th>연봉</th>
      </tr>
      <tr v-for="emp in cemps">
        <td v-html="emp.id"></td>
        <td v-html="emp.name"></td>
        <td v-html="emp.dept_id"></td>
        <td v-html="emp.title"></td>
        <td>{{ emp.salary | salarydecimal }}</td>
      </tr>
    </table>
  </div>
</script>
```

상세보기 컴포넌트는 한 명의 아이디를 입력 받아서 해당 사원의 정보를 보입니다.
비슷하지만 모든 컴포넌트에서 사용하기 위해 Vue 인스턴스의 data에 empid를 선언했습니다.
이것을 사용하기 위해서 Vue 인스턴스의 이름인 App을 이용하여 App,empid로 사원의 정보를 찾아 보여줍니다.

```
var detailhrm = Vue.component('detailhrm',{
  template: '#detailtemplate',
  data () {
    return {
      info: [],
      loading: true,
      errored: false ,
      cid:'',
      cemp:{}
    }
  },
  filters: {
    salarydecimal (value) {
      return value.toFixed(2)
    }
  },
  mounted () {
    axios
      .get('http://localhost:8197/ssafyvue/api/findEmployeeById/'+App.empid)
      //.get('./emp.json')
      .then(response => (this.cemp = response.data))
      .catch(error => {
        console.log(error)
        this.errored = true
      })
      .finally(() => this.loading = false);
  }
});
```

10. 라우팅

10.1 라우팅

10.1.1 템플릿 만들기

라우팅(routing)은 어떤 영역 안에서 데이터를 보낼 경로를 선택해서 보내는 것입니다. SPA 에서 요청 경로에 따라 다른 템플릿을 보여줄 필요가 있습니다. 이때 필요한 것이 라우팅(routing)입니다.
라우팅을 위해서 Vue Router 를 준비해야 합니다.
Vue Router 는 Vue.js 에서 제공하는 공식 라우터 라이브러리로, SPA 를 구현하기 위한 라우팅 기능을 제공합니다.

그럼 가장 기본적인 라우터 사용법에 대한 예제를 살펴보겠습니다. Vue Router 는 지금까지와 같은 방법으로 CDN 을 이용하겠습니다
라우터 라이브러리를 사용하기 위해 <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>를 선언합니다.

이제 [소스-index01.html]와 같이 <router-link to>태그를 이용하여 라우트 할 경로들을 to="요청 경로"형식을 사용해 설정한 다음, 라우팅한 템플릿을 렌더링 할 위치에 <router-view></router-view>을 선언합니다. 그리고 라우팅 할 템플릿들을 x-template 방식을 사용해 나열합니다.

[소스-index01.html]

```
<div id="app">
  <div class='search_box'>
    <router-link to = "/route1">사원추가</router-link>
    <router-link to = "/route2">이름으로 찾기</router-link>
    <router-link to = "/route3">아이디로 찾기</router-link>
    <router-link to = "/route4">모든사원 보기</router-link>
  </div>
  <router-view></router-view>
</div>
<script type="text/x-template"   id="listhrmtemplate">
  <div>사원추가 화면</div>
</script>
```

```
<script type="text/x-template"   id="idhrmtemplate">
  <div>아이디로 찾기 화면</div>
</script>
<script type="text/x-template"   id="namehrmtemplate">
  <div>이름으로 찾기 화면</div>
</script>
<script type="text/x-template"   id="addhrmtemplate">
  <div>사원추가 화면</div>
</script>
```

나열한 각 x-template 의 뷰 컴포넌트들을 선언합니다.

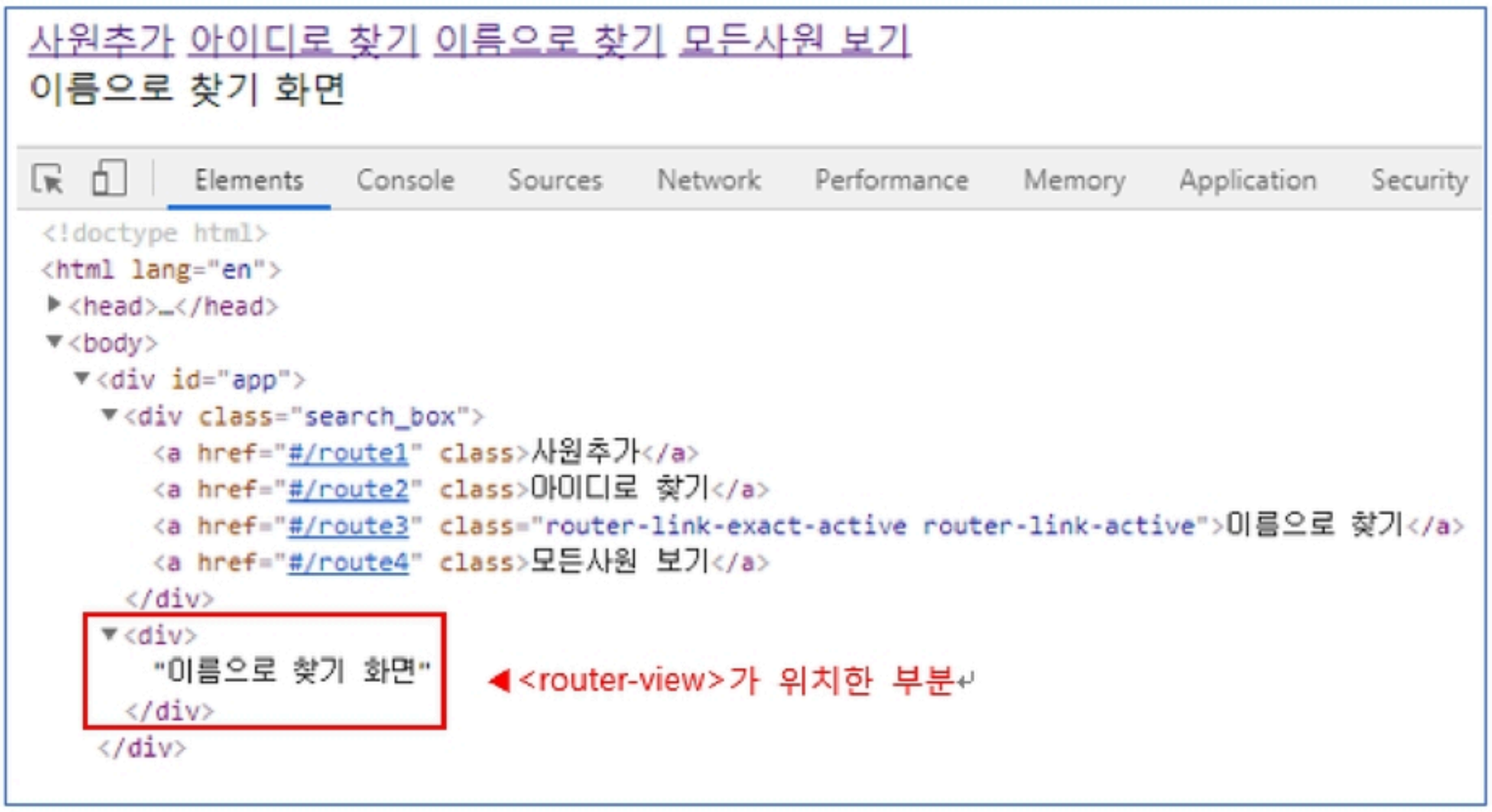
```
<script type="text/javascript">
var listhrm = Vue.component('listhrm',{
      template: '#listhrmtemplate'
});
var idhrm = Vue.component('idhrm',{
      template: '#idhrmtemplate'
});
var detailhrm = Vue.component('detailhrm',{
      template: '#detailtemplate'
});
var namehrm = Vue.component('namehrm',{
      template: '#namehrmtemplate'
});
var addhrm = Vue.component('addhrm',{
      template: '#addhrmtemplate'
});
```

라우트 경로와 템플릿을 매핑합니다.

```
const Route1 = { template: listhrmtemplate}
const Route2 = { template: idhrmtemplate }
const Route3 = { template: namehrmtemplate}
```

```
const Route4 = { template: addhrmtemplate }
const routes = [
    { path: '/route1', component: Route1 },
    { path: '/route2', component: Route2 },
    { path: '/route3', component: Route3 },
    { path: '/route4', component: Route4 }
];
const router = new VueRouter({
     routes
});
var App=new Vue({
   el: '#app',
   router
})
 </script>
```

브라우저에서 보고자 하는 링크 내용을 클릭하면 해당 템플릿으로 라우팅 되어 <router-view></router-view>에 해당 템플릿을 보입니다.



10.2 SPA 구현

10.2.1 라우팅을 이용하여 SPA 구현해보기

라우팅(routing)은 어떤 영역 안에서 데이터를 보낼 경로를 선택해서 보내는 것이라 했습니다.
Vue 라우터는 한 페이지에 여러 템플릿을 선언해 놓은 후 경로 요청에 따라 해당 템플릿만 보여주어 SPA 를 구현할 수 있게 합니다.

다시 한 번 간략히 뷰의 라우팅 사용법에 대해 살펴보겠습니다.

① 라우팅 경로들을 선언합니다. <a href='#'>사원추가</a> 효과와 비슷합니다.

```
<router-link to = "/route1">사원추가</router-link>
```

② 여러 템플릿 중 선택되었을 때 보일 템플릿을 위한 자리를 선언합니다.

```
<router-view></router-view>
```

③ 템플릿을 선언합니다.

```
<script type="text/x-template" id="listhrmtemplate">
```

④ 컴포넌트를 선언합니다.

```
var listhrm = Vue.component('listhrm',{
      template: '#listhrmtemplate'
});
```

⑤ 라우트 할 경로와 템플릿을 매핑합니다.

```
const Route1 ={   template: listhrmtemplate}
```

⑥ 라우트에 해당 컴포넌트의 경로를 등록합니다.

```
const routes = [
    { path: '/route1', component: Route1 },
    { path: '/route2', component: Route2 },
    { path: '/route3', component: Route3 },
```

```
    { path: '/route4', component: Route4 }
];
```

⑦ 이 경로들을 라우터에 등록합니다.

```
const router = new VueRouter({
    routes
});
```

⑧ 라우터를 Vue 인스턴스에 선언합니다.

```
var App=new Vue({
   el: '#app',
   router
})
```

다시 한 번 기본적인 뷰의 라우팅 사용법에 대해 알아보았으니 이를 활용하여 9 장에서 만든 템플릿을 한 페이지에 선언해 SPA 를 만들어 보겠습니다. 10.1 에서 템플릿 위치에 템플릿들을 선언합니다.

[소스-index02.html]

```
<div id='app'>
  <div class='headtitle'><h2 >SSAFY HRM</h2></div>
  <div class='search_box'>
    <router-link to = "/route1">사원추가</router-link>
    <router-link to = "/route2">이름으로 찾기</router-link>
    <router-link to = "/route3">아이디로 찾기</router-link>
    <router-link to = "/route4">모든사원 보기</router-link>
  </div>
  <router-view></router-view>
</div>
<script type="text/x-template" id="listhrmtemplate">
  <div>
    <div>
      <table class='list_table'>
```

```
        <col width="10%"><col width="20%">
        <col width="20%">
        <col width="20%">
        <col width="10%">
        <tr>
          <th>사원 아이디</th>
          <th>사원명</th>
          <th>부서</th>
          <th>직책</th>
          <th>연봉</th>
        </tr>
        <tr>
          <td>4</td>
          <td>홍길동</td>
          <td>영업부</td>
          <td>사원</td>
          <td>12345</td>
        </tr>
      </table>
    </div>
  </div>
</script>
```

이제 9 장에서 만든 템플릿들을 위에 배웠던 뷰 라우팅 사용법을 적용하여 아래의 그림과 같이 출력할 수 있도록 구현해 봅니다.

[그림 10.1] 한 페이지에서 라우팅을 이용하여 사원 리스트 보이기

[그림 10.2] 한 페이지에서 라우팅을 이용하여 아이디로 찾기 화면 보이기

[그림 10.3] 한 페이지에서 라우팅을 이용하여 사원 추가 화면 보이기

11. Vue 프로젝트 1

Vue-CLI

Vue-CLI 는 Vue에서 공식으로 지원하는 애플리케이션 개발 환경 설정 도구입니다.
개발 환경을 제공해주는 기능도 하지만, @vue/cli-service-global을 통해서 그런 과정조차도 필요 없는 프로토 타입 애플리케이션을 만들 수도 있습니다.

SPA를 제작하기 위해서 앞서 사용했었던 여러 컴포넌트들을 각각 하나의 파일로 만들어서 애플리케이션을 구성하게 되는데, 이런 각각의 컴포넌트 파일들을 단일 파일 컴포넌트(SFC: Single File Components) 라고 부릅니다.
이 SFC들은 각각 세가지 요소들(<template>,<script>,<style>)로 구성되어 있습니다. 이 세 가지 요소들은 HTML 문법으로 작성할 수 있습니다.
<template>은 컴포넌트의 template 옵션과 같은 기능으로, 템플릿을 정의할 수 있습니다. SFC 환경에서는 한 컴포넌트당 하나의 template만을 선언할 수 있습니다.
<script>는 기존의 HTML 파일에서 사용했던 UI를 반응형으로 동작시키기 위해 DOM을 제어하는 자바스크립트 코드들을 작성하고, 해당 컴포넌트가 자신을 export 하거나 다른 컴포넌트를 import 할 수 있는 요소입니다. template과 마찬가지로 하나의 컴포넌트에 하나만 선언할 수 있고 자바스크립트로 작성하면 됩니다.
<style>은 HTML에서 UI를 꾸미기 위해 사용한 <style> 태그와 동일합니다. 다만, 하나의 컴포넌트에서 여러 개의 <style> 요소를 선언할 수 있고, 이 <style> 요소는 2가지의 종류를 가집니다.
첫 번째는 기존 HTML CSS 처럼 전역에 적용되는 CSS를 작성할 수 있는 <style>문입니다.
두 번째는 해당 <style>문이 작성된 컴포넌트에만 적용시킬 수 있는 scoped style 문을 작성할 수 있습니다. 이 경우 해당 컴포넌트의 CSS가 적용될 요소에 식별 가능한 특수한 데이터 속성이 할당됩니다.
SFC는 보통 다음과 같은 형식으로 작성됩니다.

```
<template>
  <div class="app">{{ message }}</div>
</template>

<script>
export default {
  data () {
    return {
      message: 'Hello SSAFY!'
    }
  }
}
</script>

<style scoped>
```

```
.app {
   color: red;
}
</style>
```

11.1 개발환경(노드 설치 후)

11.1.1 npm

1. 뷰관련 npm 명령어

vue add element

npm install -g @vue/cli

vue create hellovues

npm run serve

npm install jquery popper.js bootstrap -save

npm install axios --save

npm install vuelidate --save

npm install vue-router --save

npm install vuex -save

11.1.2 hellovues 프로젝트 만들기

2. 라이브러리 설치

npm install -g @vue/cli

vue create hellovues

cd hellovues

npm install axios -save

npm install vue-router --save

```
npm install vuex -save

npm install d3 -save

npm run serve
```

[그림 11.1] npm 설치

[그림 11.2] 프로젝트 만들기 1 (default)

[그림 11.3] 프로젝트 만들기 2

3. 환경설정(package.json)

cd hellovues

```
{
  "name": "hellovues",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "core-js": "^2.6.5",
    "vue": "^2.6.10"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^3.11.0",
    "@vue/cli-plugin-eslint": "^3.11.0",
    "@vue/cli-service": "^3.11.0",
    "babel-eslint": "^10.0.1",
    "eslint": "^5.16.0",
    "eslint-plugin-vue": "^5.0.0",
    "vue-template-compiler": "^2.6.10"
  },
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
    "extends": [
      "plugin:vue/essential",
      "eslint:recommended"
    ],
    "rules": {},
    "parserOptions": {
```

```
        "parser": "babel-eslint"
    }
  },
  "postcss": {
    "plugins": {
        "autoprefixer": {}
    }
  },
  "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
}
```

4. 서버실행 : yarn serve (or npm run serve)

5. 서버 정지 : ctrl+c

6. 기본 라이브러리

```
"dependencies": {
    "core-js": "^2.6.5",
    "vue": "^2.6.10"
  },
```

7. 라우터 라이브러리 추가(예)

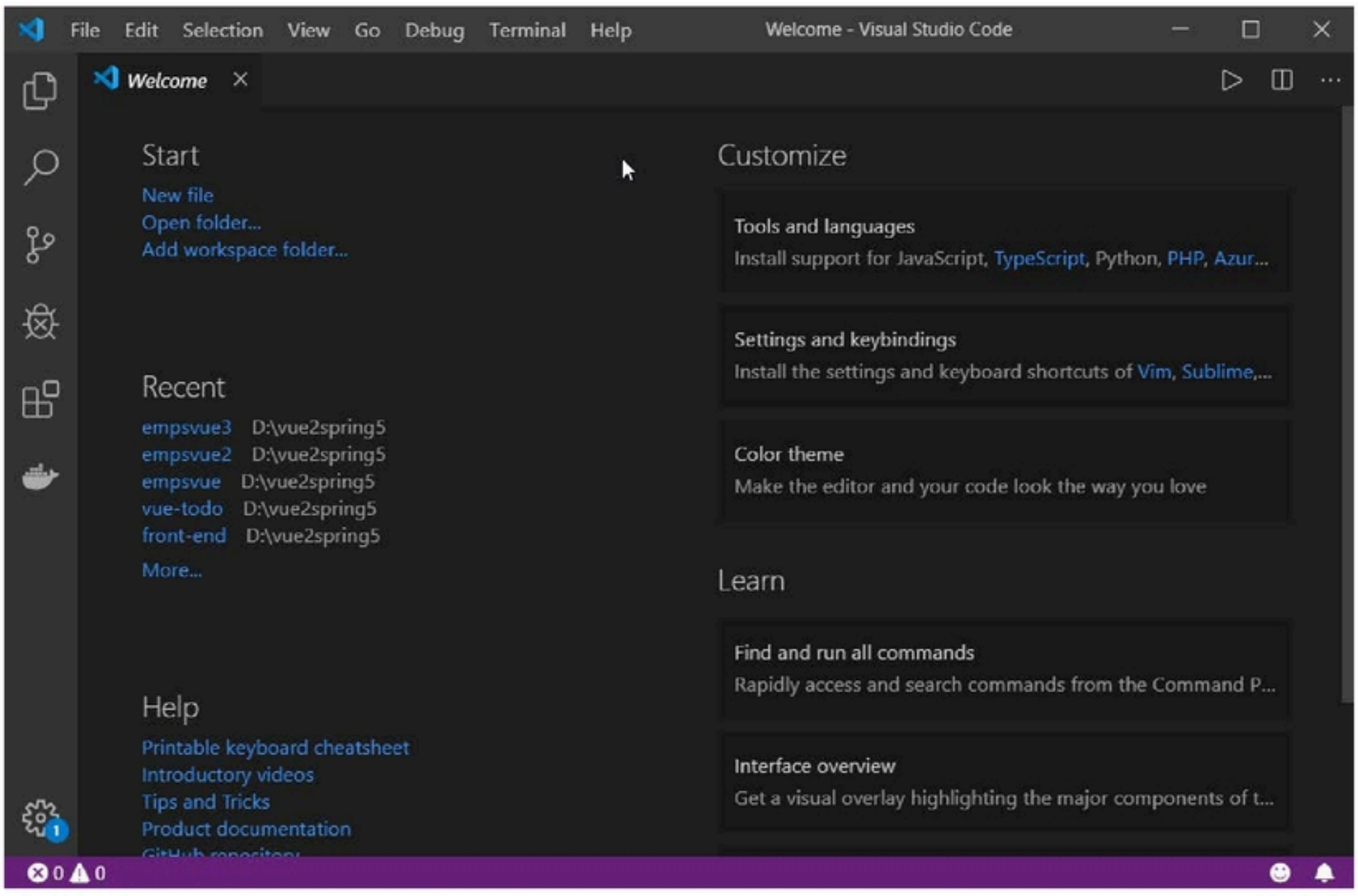
npm install vue-router -save

```
  "dependencies": {
      "core-js": "^2.6.5",
      "vue": "^2.6.10",
      "vue-router": "^3.1.3"
  },
```

11.2 Visual Studio Code

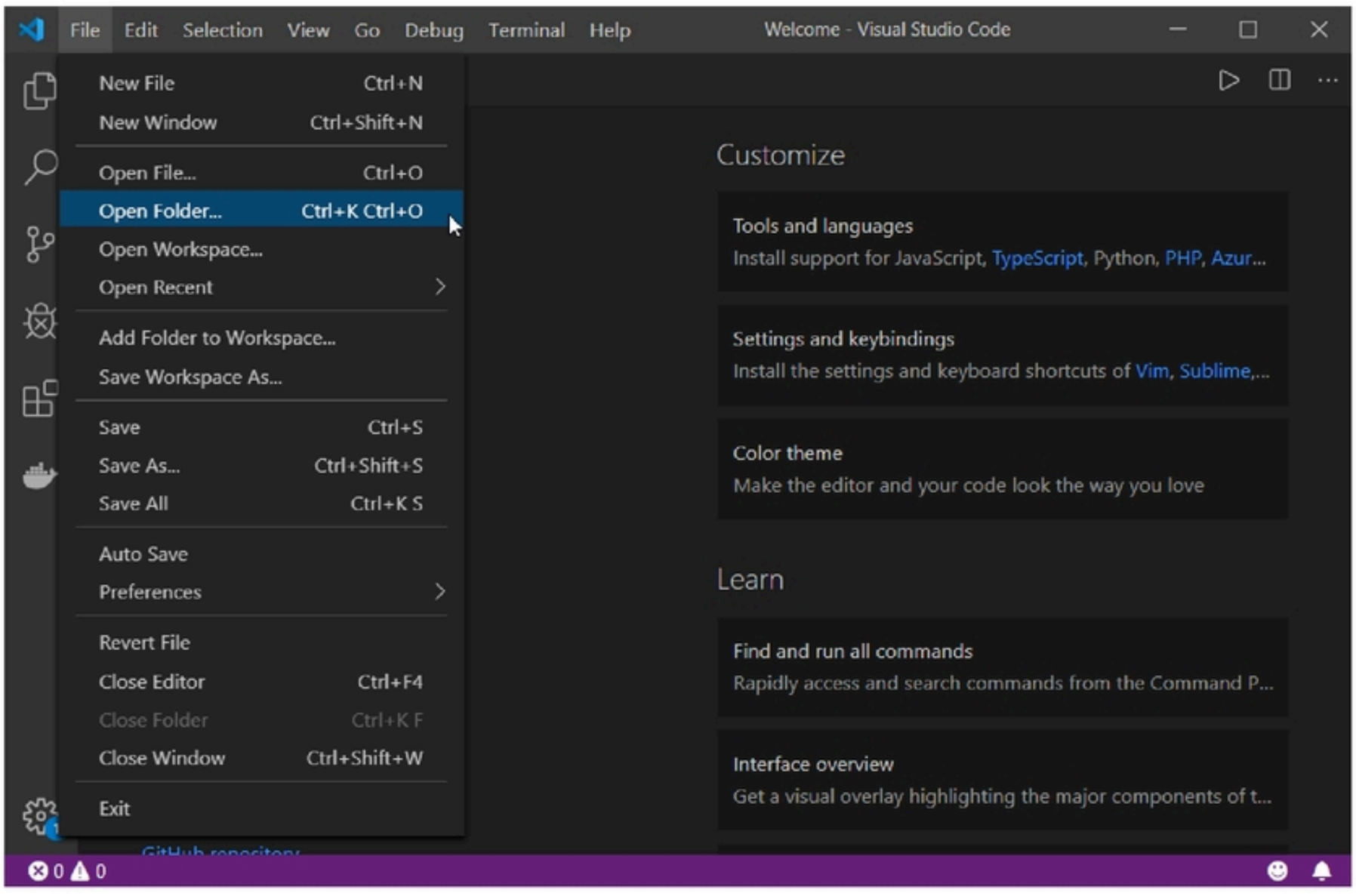
11.2.1 vs에서 프로젝트 구현

1. Visual Studio Code를 실행하십시오.



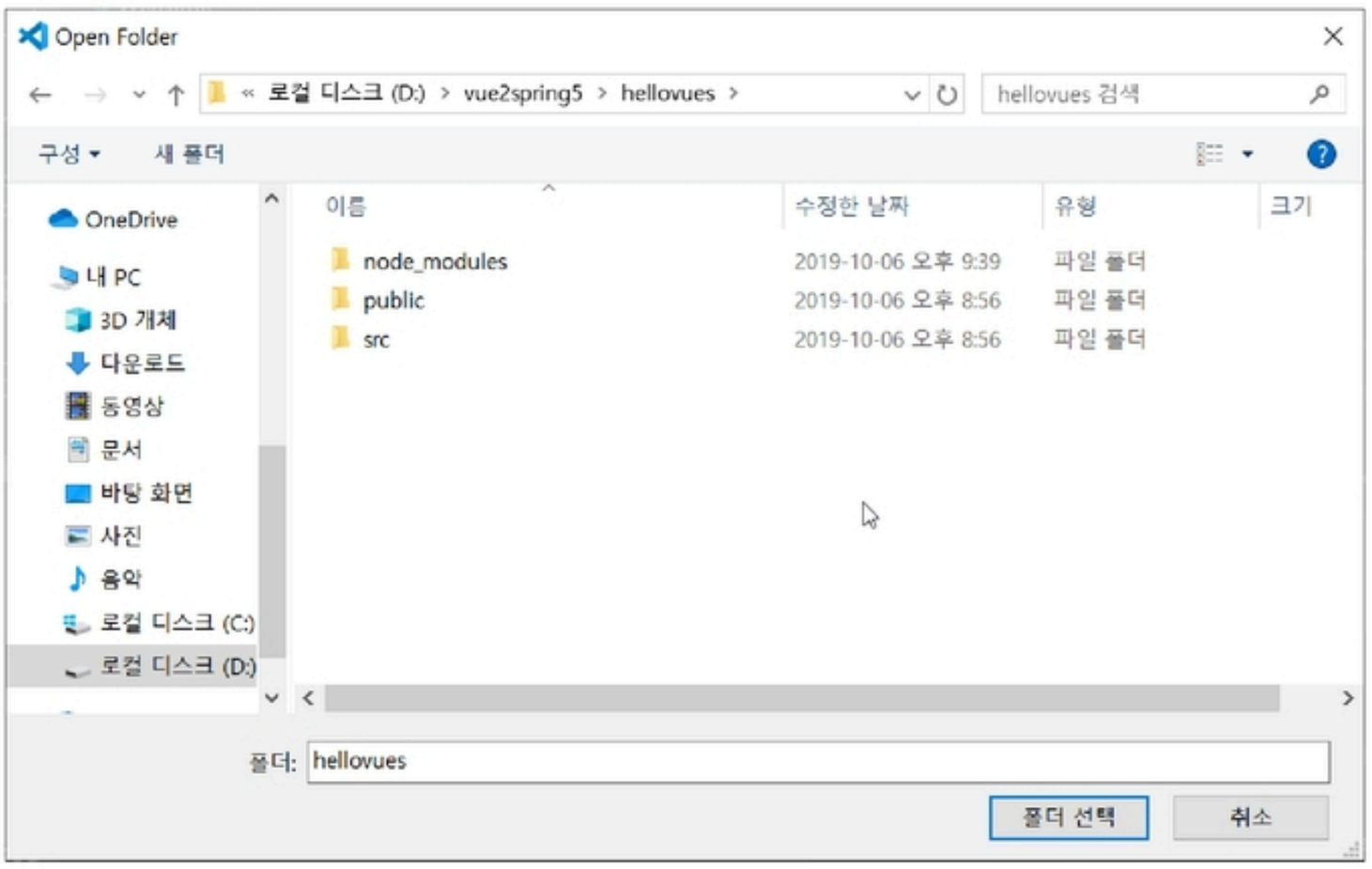
[그림 11.4] vs 실행시키기

2. Open Folder를 선택하십시오.



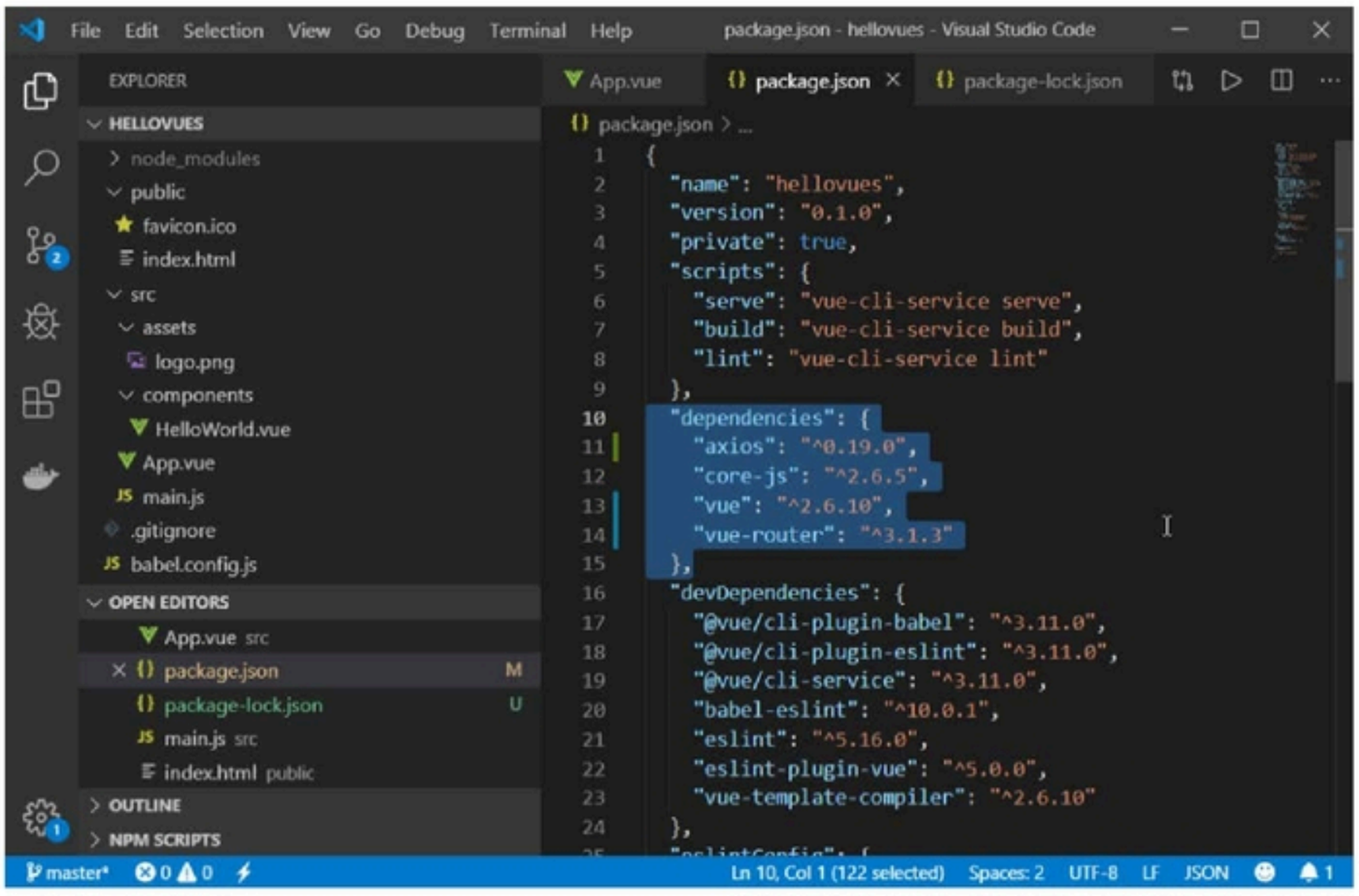
[그림 11.5] 프로젝트를 오픈 폴더로 연다.

3. Open Folder로 생성한 hellovues 폴더를 찾으세요.



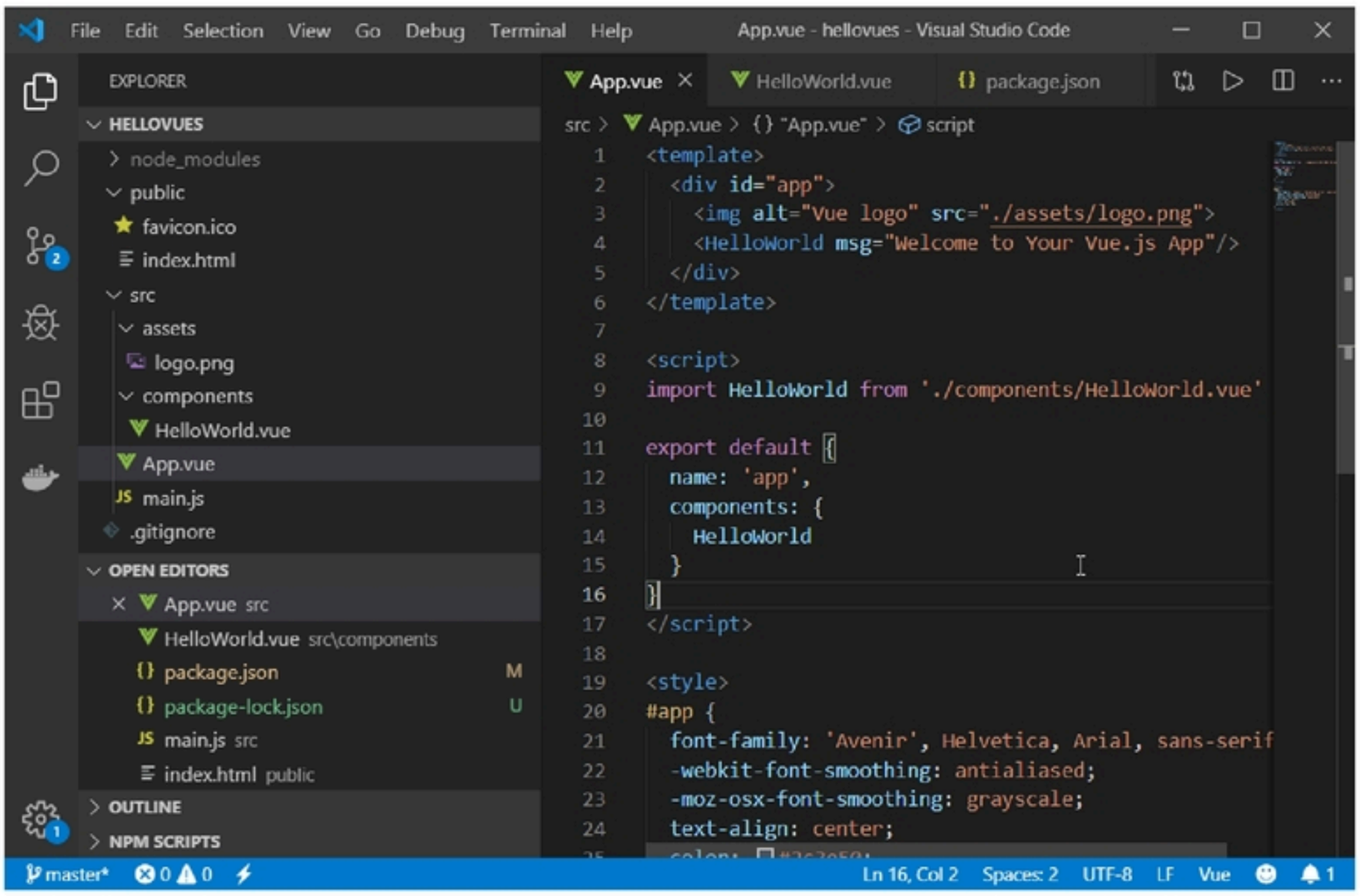
[그림 11.6] hellovues 를 선택합니다.

4. package.json에서 추가된 의존관계를 확인합니다.



[그림 11.7] 의존관계를 확인합니다.

5. App.vue를 확인합니다.



[그림 11.8] App.vue 를 선택합니다.

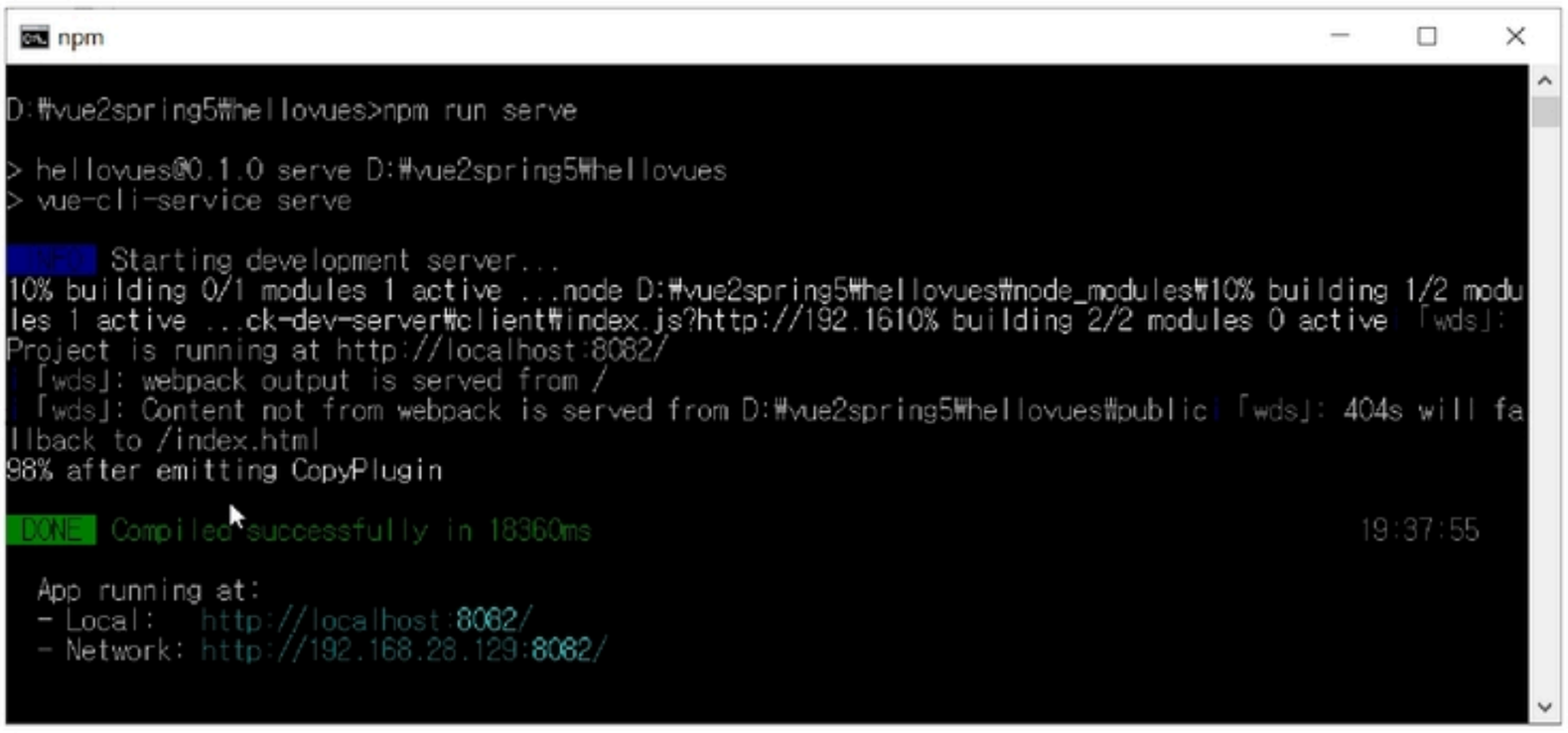
App.vue는 시작 컴포넌트(메인 컴포넌트)입니다. 기본 설정으로 에 자식 컴포넌트인 Helloworld가 있습니다. 등록을 위해서 Helloworld를 import합니다. Helloworld를 components 옵션에 등록합니다.

서버를 실행시킵니다.

Hellovues 디렉토리에서 서버를 실행시킵니다.

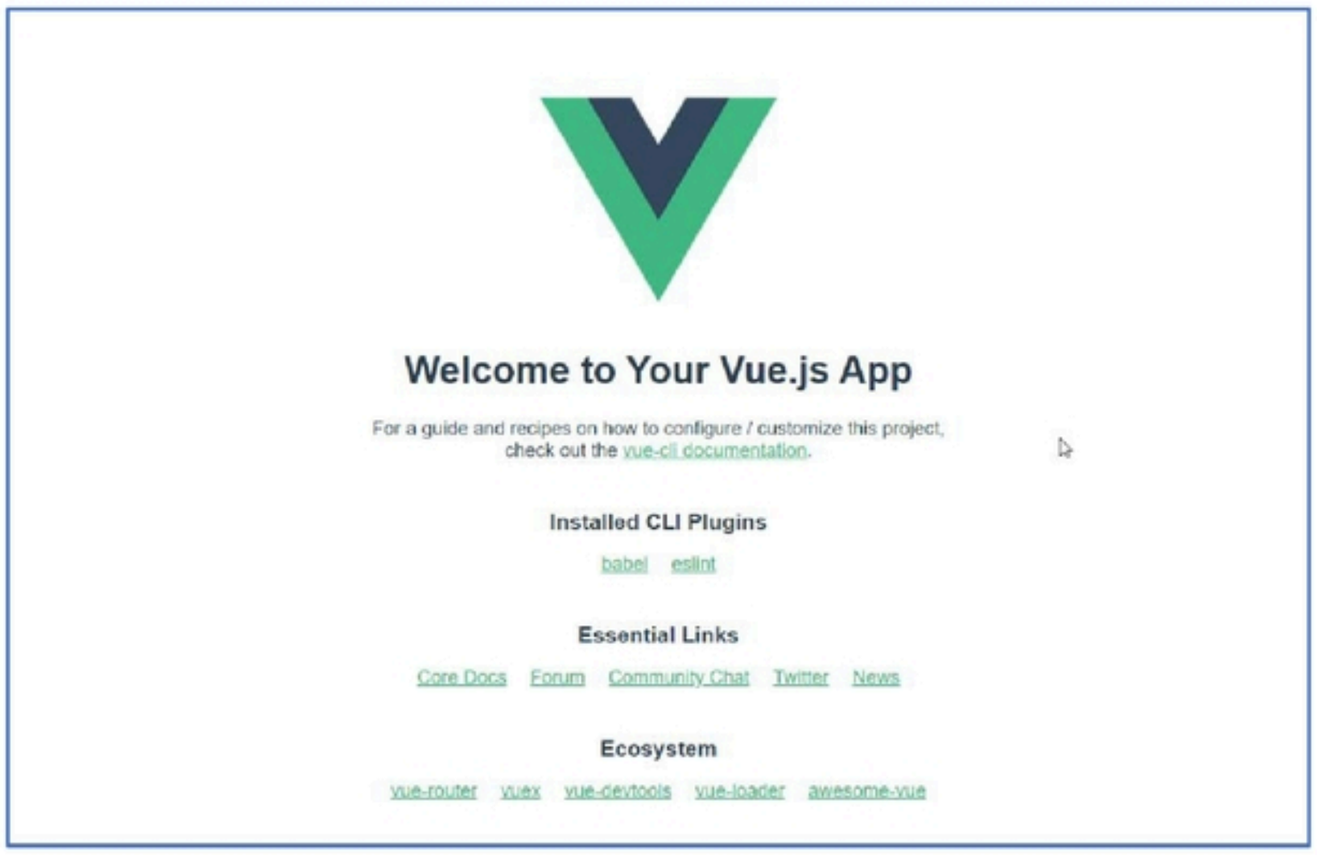
npm run serve

실행port를 확인하십시오.



[그림 11.9] npm 으로 서버 실행시키기.

6. 브라우저를 실행시킵니다.



[그림 11.10] 브라우저로 확인합니다.

11.3 서비스 확인

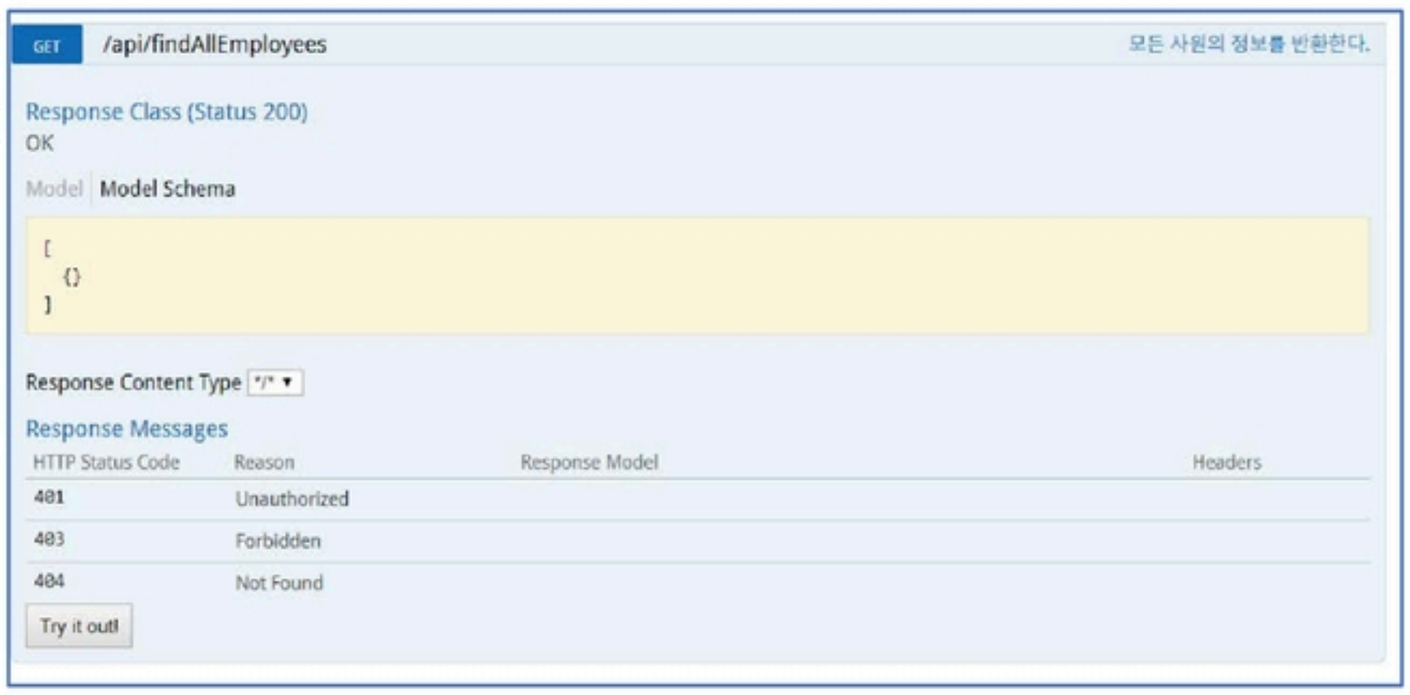
11.3.1 웹 서비스 및 API

vueboot 메이븐 프로젝트의 SSAFYApplication를 실행시키세요.

http://localhost:8197/ssafyvue/swagger-ui.html

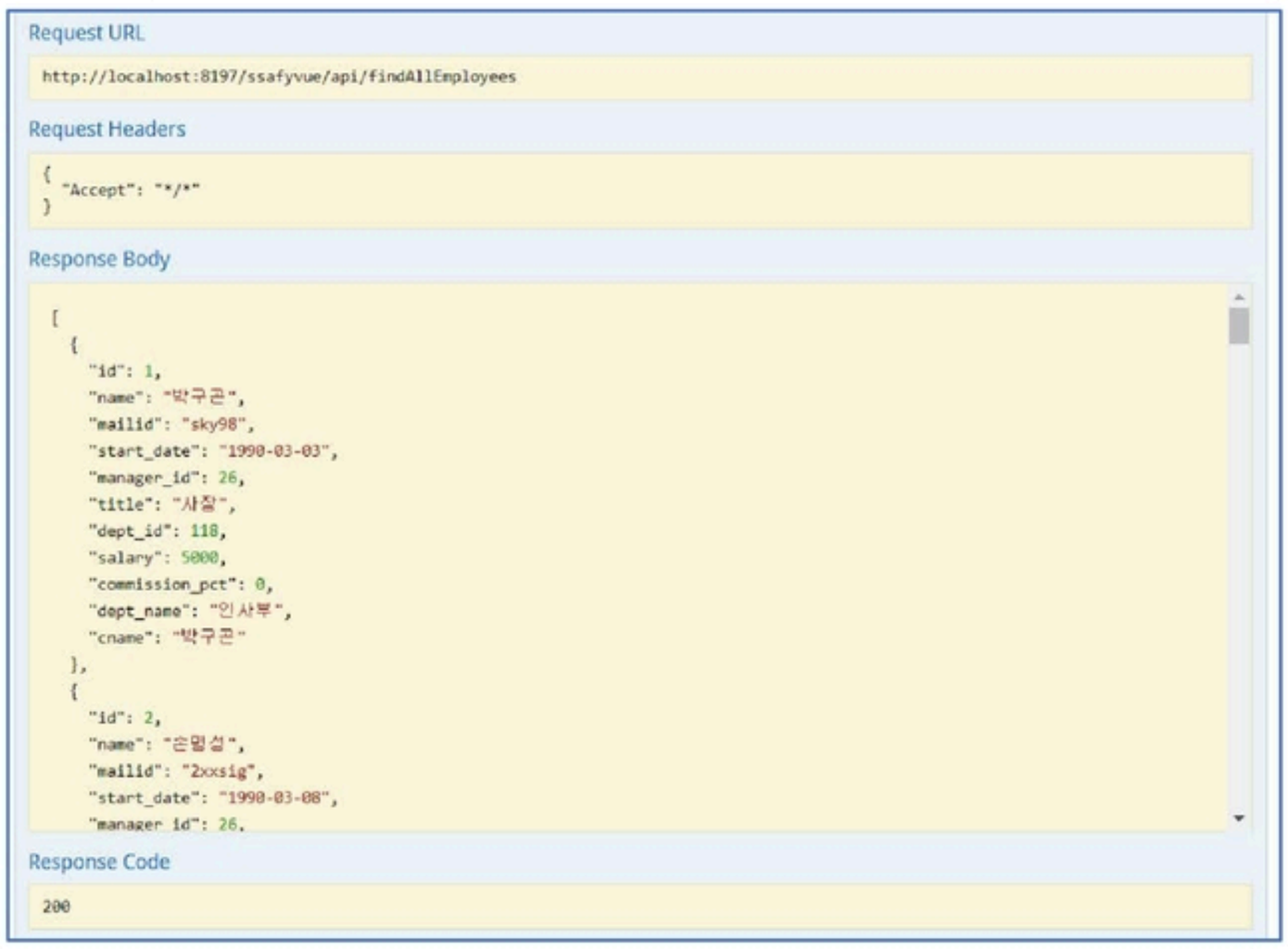
브라우저에 스웨거 API 경로를 입력하십시오.

1. Swagger API 확인



[그림 11.11] 스웨거로 서비스 확인하기

2. 요청경로와 실행결과를 확인합니다. 결과 데이터를 확인합니다.



[그림 11.12] 스웨거로 서비스 확인하기

11.4 뷰 프로젝트 구현

11.4.1 사원 목록 보기

[소스-main.js]

```
import Vue from "vue";
import App from "./App.vue";
import router from './router'
Vue.config.productionTip = false;
new Vue({
  router,
  render: h => h(App)
}).$mount("#app");
```

[main.js]에 router 를 추가합니다. Import 할 때 './router'는 현재 경로(.)에서 router.js 를 임포트 한다는 의미입니다.

[소스-CustomersList.vue/template]

```
<template>
  <div>
    <div>
      <table class='list_table'>
        <col width="10%"><col width="20%">
        <col width="20%"><col width="20%"><col width="10%"><col width="10%">
        <tr>
          <th>사원 아이디</th>
          <th>사원명</th>
          <th>부서</th>
          <th>직책</th>
          <th>연봉</th>
          <th>퇴사</th>
        </tr>
        <tr v-for="emp in info" class = "nicecolor">
          <td v-html="emp.id"></td>
```

```
        <td v-html="emp.name"></td>
        <td v-html="emp.dept_name"></td>
        <td v-html="emp.title"></td>
        <td>{{ emp.salary | salarydecimal }}</td>
        <td class='button'><input type="button" class='blue' value='퇴사'> </td>
      </tr>
    </table>
  </div>
</div>
</template>
```

info 리스트(배열)에서 한 개의 사원정보를 emp 로 정의하고 한 개씩 사원의 아이디, 이름, 부서이름, 직책, 연봉을 출력합니다. 이때 필터(salarydecimal)를 이용하여 연봉을 소수 둘째 자리까지만 출력합니다.

[소스-CustomersList.vue]

```
<script>
import http from "../http-common";
export default {
  name: "customers-list",
  data(){
    return {
      info: [],
      loading: true,
      errored: false
    }
  },
  methods:{
    retrieveCustomers() {
      http
        .get('/findAllEmployees')
          .then(response => (this.info = response.data))
          .catch(error => {
            this.errored = true
          }) .finally(() => this.loading = false);
```

```
  }},
filters: {
        salarydecimal (value) {
          var a=parseInt(value);
        return a.toFixed(2);
      }},
  mounted () {
        this.retrieveCustomers();
    }};
</script>
```

화면이 렌더링 되기 전에 모든 사원의 정보를 가져옵니다. 그리고 렌더링(mounted)할 때 사원의 정보를 출력합니다. [http-commons.js]에서 선언한 기본 경로 http://localhost:8197/ssafyvue/api 에 /findAllEmployees 붙여서 http://localhost:8197/ssafyvue/api/findAllEmployees 를 호출합니다.

[소스-http-commons.js]

```
import axios from "axios";
export default axios.create({
  baseURL: "http://localhost:8197/ssafyvue/api",
  headers: {
    "Content-type": "application/json",
  }
});
```

[main.js]옆에 [router.js]를 만듭니다. 기본경로에 상세경로를 붙여 사용할 것입니다.

[소스-router.js]

```
import Vue from "vue";
import Router from "vue-router";
import CustomersList from "./components/CustomersList.vue";
Vue.use(Router);
export default new Router({
  mode: "history",
```

```
routes: [
  {
    path: "/",
    name: "customers",
    alias: "/customer",
    component: CustomersList
  }
  ]
});
```

<router-link to="/">나 <router-link to="/customer">는 <router-view/> 에 CustomersList.vue 를 보이도록 합니다.

[소스-Vue.vue]

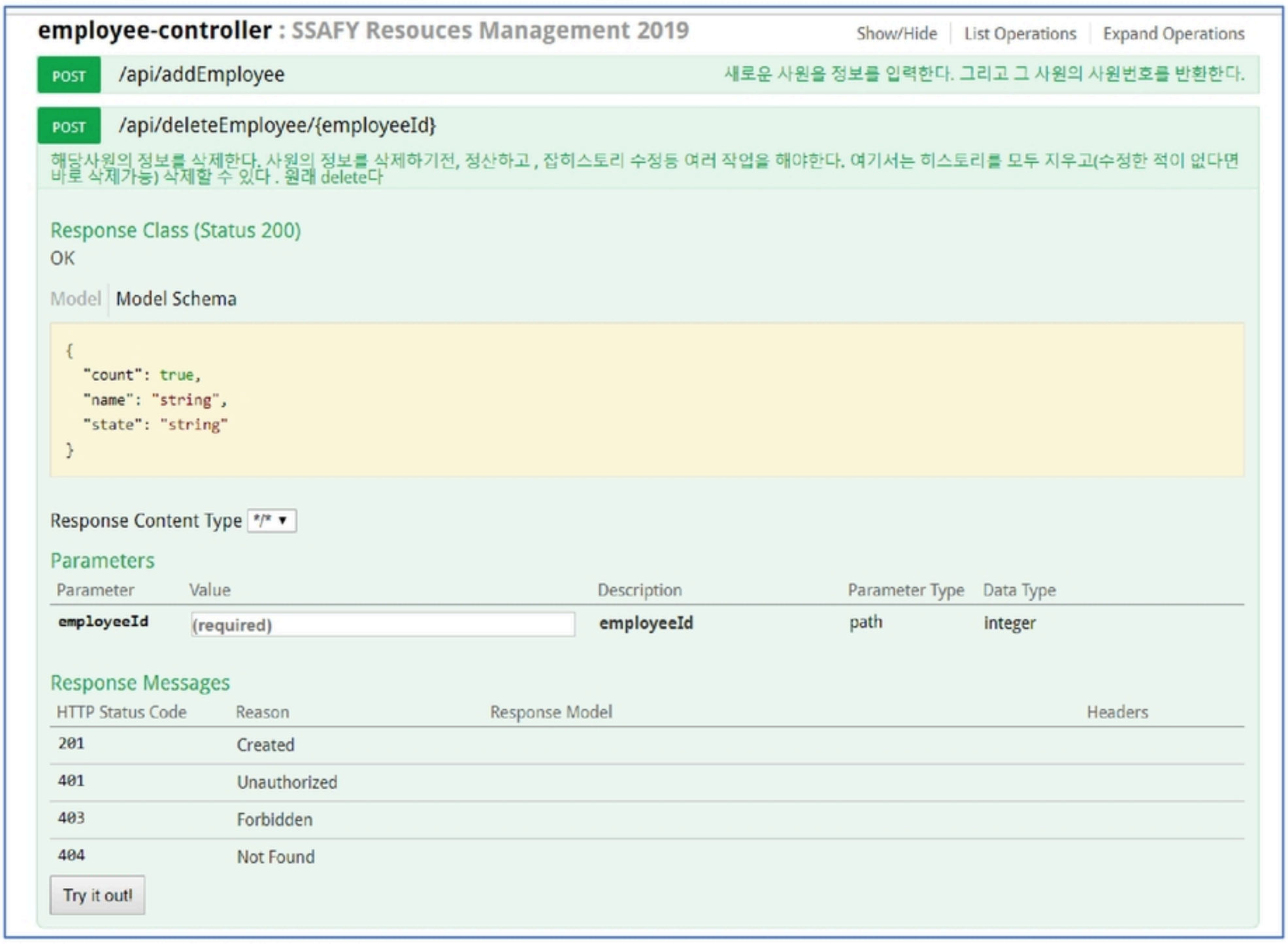
```
<template>
  <div id="app">
    <div class='headtitle'><h2>SSAFY HRM LIST</h2></div>
    <div class='search_box'>
      <nav>
        <router-link class="btn btn-primary" to="/">모든사원 보기</router-link> |
      </nav>
    </div>
    <router-view/>
  </div>
</template>
<script>
  export default {
    name: "app"
  };
</script>
<style></style>
```

<router-link to="/">나 <router-link to="/customer">는 <router-view/> 에 CustomersList.vue 를 보입니다.

[그림 11.13] 모든 사원 보기 화면

11.4.2 사원 삭제와 상세보기

사원 삭제를 위하여 삭제 서비스를 스웨거로 확인합니다.



[그림 11.14] 사원 삭제 API

[소스-CustomersList.vue] 추가

```
<tr v-for="emp in info" class = "nicecolor">
  <td v-html="emp.id"></td>
  <td v-html="emp.name"></td>
  <td v-html="emp.dept_name"></td>
  <td v-html="emp.title"></td>
  <td>{{ emp.salary | salarydecimal }}</td>
  <td class='button' @click="delete_emp(emp.id)"><input type="button" class='blue' value='퇴사
'> </td>
</tr>
```

사원 삭제를 위한 소스를 추가합니다. 퇴사 버튼을 클릭하면 delete_emp(사원아이디) 를 호출합니다.

[소스-CustomersList.vue] 추가

```
delete_emp(employeeid){
  if(employeeid==1){
    alert("사장님은 퇴사할 수 없습니다.");
    return;
  }
  alert(employeeid+"가 퇴사합니다.");
  http
    .post('/deleteEmployee/'+employeeid)
    .then(response => {
      if (response.data.state=='succ') {
        alert("퇴사처리를 하였습니다.");
        this.retrieveCustomers();
      }else{
        alert("퇴사처리를 하지 못했습니다.");
      }
    })
    .catch(error => {
      this.errored = true
    })
    .finally(() => this.loading = false);
  }
}
```

해당사원의 아이디를 받아서 해당사원을 삭제합니다. 삭제에 성공했다면 methods:에 선언한 retrieveCustomers() 를 호출하여 다시 모든 사원의 정보를 출력합니다. 이때 멤버에 해당하는 메서드를 호출하기 때문에 **this**. retrieveCustomers()를 이용합니다.

11.5 사원정보 삭제, 찾기

11.5.1 사원 삭제, 찾기

[hellovues]를 복사한 후 복사본을 [hellovues2]로 수정하십시오. 그리고 [package.json]와[package-lock.json] 의 이름을 아래와 같이 hellovues2 로 수정하십시오.

```
{
  "name": "hellovues2",
  ...
}
```

아래 그림과 같이 상세보기를 추가할 것입니다.

[그림 11.15] 사원 정보 보기(사원 선택)

[그림 11.16] 사원 상세 정보
[소스-DetailCustomer.vue]

```
<template>
  <div>
    <table class='list_table'>
      <col width="10%"><col width="20%">
      <col width="20%"><col width="20%"><col width="10%">
      <tr>
        <th>사원 아이디</th><th>사원명</th><th>부서</th><th>직책</th><th>연봉</th>
      </tr>
       <tr>
        <td v-html="cemp.id"></td>
        <td v-html="cemp.name"></td><td v-html="cemp.dept_id"></td>
        <td v-html="cemp.title"></td> <td>{{ cemp.salary | salarydecimal }}</td>
      </tr>
    </table>
    <div class='search_box'>
      <h3><span style='color:blue;' v-html="cemp.name"></span>의 부하 직원들</h3>
    </div>
    <table class='list_table'><col width="10%"><col width="20%">
      <col width="20%"><col width="20%"><col width="10%">
      <tr>
        <th>사원 아이디</th><th>사원명</th><th>부서</th><th>직책</th><th>연봉</th>
      </tr>
      <tr v-for="emp in info" class = "nicecolor">
        <td v-html="emp.id" ></td><td v-html="emp.name"></td>
        <td v-html="emp.dept_id"></td><td v-html="emp.title"></td>
        <td>{{ emp.salary | salarydecimal }}</td>
      </tr>
    </table>
  </div>
</template>
```

한 사원의 정보를 cemp 객체에 저장 후 아이디, 이름 등을 출력합니다. 해당 사원을 상사로 모시는 부하 직원들의 정보를 info 리스트에 저장 후 하나씩 부하 직원들을 출력합니다.

[소스-DetailCustomer.vue- script]

```
<script>
  import http from "../http-common";
  export default {
    name: "detailcustomer",
    props: ['id'],
    data () {
      return {
        info: [],
        cemp:{}
      }
    },
    filters: {
      salarydecimal (value) {
        var a=parseInt(value);
        return a.toFixed(2);
      }
    },
    mounted () {
      http
        .get('/findEmployeeById/'+this.id)
        .then(response =>
          (this.cemp = response.data), this.cmanager_id=this.cemp.manager_id)
        .catch(error => {this.errored = true})
        .finally(() => this.loading = false);
      http
        .get('/findEmployeesByManagerId/'+this.id)
        .then(response => (this.info = response.data))
        .catch(error => {this.errored = true})
        .finally(() => this.loading = false);
    }
  };
</script>
```

id 에 해당하는 한 사원의 정보를 findEmployeeById 로 얻습니다. 해당 사원을 상사로 모시는 부하직원들을 findEmployeesByManagerId 로 얻습니다. 여기서 id 는 props 로 넘어온 id 입니다. 사원목록(리스트)에서 선택된 아이디를 props 에 저장한 후, 상세보기에서 props 에 저장되었던 id 를 this.id 로 찾아 사용합니다.

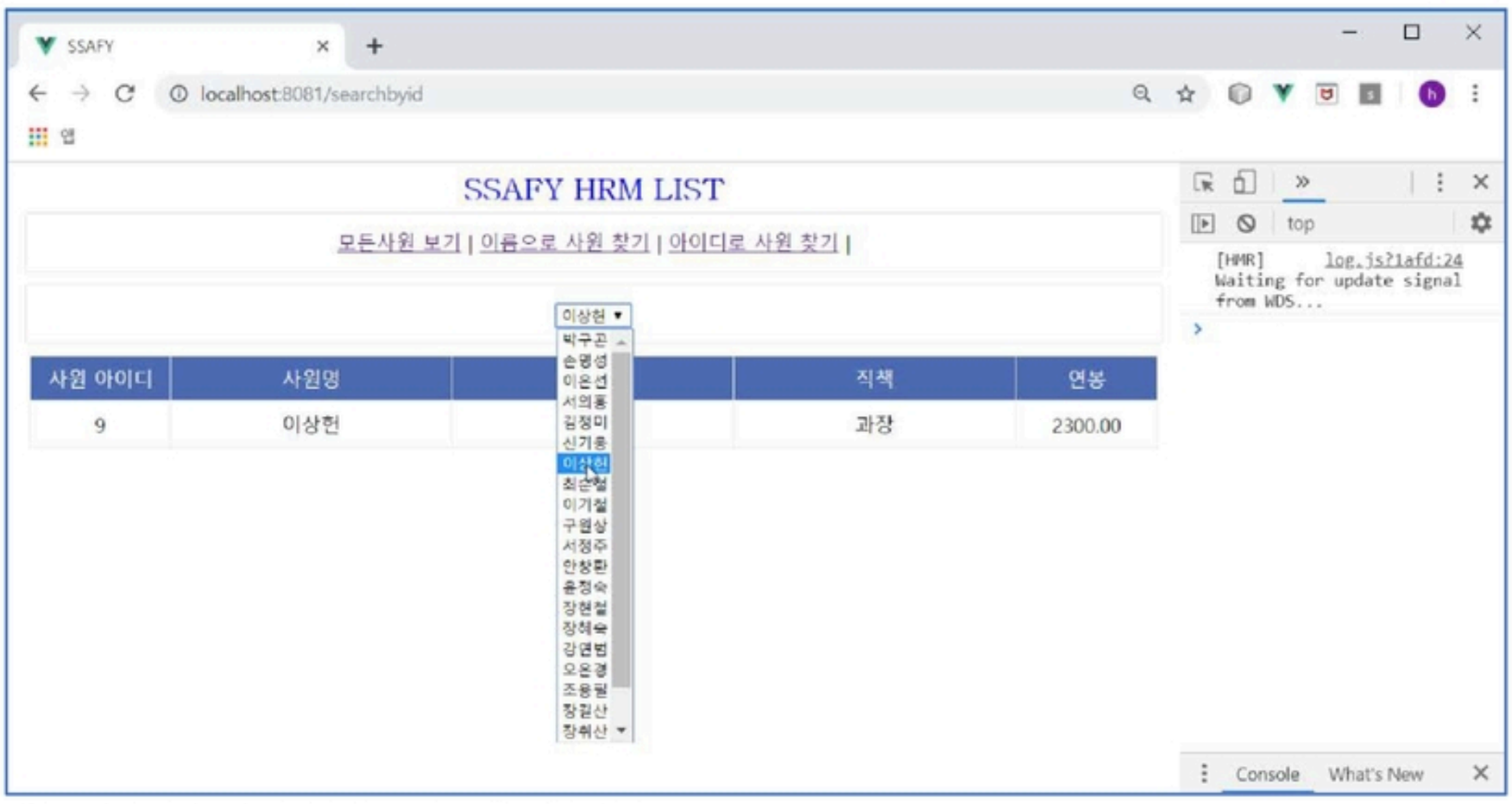
[소스-router.js]

```
import Vue from "vue";
import Router from "vue-router";
import CustomersList from "./components/CustomersList.vue";
import DetailCustomer from "./components/DetailCustomer.vue";
Vue.use(Router);

export default new Router({
  mode: "history",
  routes: [
    {
      path: "/",
      name: "customers",
      alias: "/customer",
      component: CustomersList
    },
    {
      path: '/detailcustomer/:id',
      name: 'detailcustomer',
      component: DetailCustomer,
      props: true,
    }
  ]
});
```

<router-view/>에 DetailCustomer.vue 를 보이기 위해 router 에 DetailCustomer 를 추가합니다. 여기서 props:true 가 중요합니다. 모든 사원 중 한 사원을 선택할 때 props 를 이용하여 선택된 아이디를 형제 컴포넌트(CustomersList 와 DetailCustomer 가 형제)인 DetailComponent 로 보낼 수 있습니다. 변수는 컴포넌트 내부에서만 사용할 수 있기 때문에 CustomersList 에서 선택된 id 를 DetailCustomer 로 보내기 위해 프로퍼티(props)에 저장합니다.

이름과 사번으로 사원찾기를 하겠습니다.

[그림 11.17] 이름으로 사원 찾기



[그림 11.18] 사원 아이디로 사원 찾기

[소스-SearchCustomers.vue]

```
<template>
  <div>
    <div class='search_box'>
      이름으로  찾기 :
      <input type="text" name='name' v-model="cname" @keyup='searchname' v-cloak/>
    </div>
    <table class='list_table'>
      <col width="10%"><col width="20%">
      <col width="20%"><col width="20%"><col width="10%">
      <tr>
        <th>사원 아이디</th>
        <th>사원명</th>
        <th>부서</th>
        <th>직책</th>
        <th>연봉</th>
      </tr>
      <tr v-for="emp in cemps" class = "nicecolor">
        <td v-html="emp.id"></td>
        <td v-html="emp.name"></td>
        <td v-html="emp.dept_id"></td>
        <td v-html="emp.title"></td>
        <td>{{ emp.salary | salarydecimal }}</td>
      </tr>
    </table>
  </div>
</template>
```

v-model="cname" @keyup='searchname' data 에 등록된 cname 으로 keyup 이벤트가 발생하면 searchname 메서드를 호출하여 이름으로 사원을 찾습니다.

[소스-SearchCustomers.vue]

```
<script>
import http from "../http-common";
export default {
  name: "search-customer",
  data () {
        return {
          loading: true,
          errored: false ,
          cname:'',
          cemps:[]
    } },
filters: {
      salarydecimal (value) {
      var a=parseInt(value);
      return a.toFixed(2);
   }},
  methods: {
     searchname(e) {
       if(e.which == 13){ //enter
          http.get('/findLikeEmployees/'+this.cname)
          .then(response => (this.cemps = response.data))
          .catch(error => {this.errored = true
          }).finally(() => this.loading = false);
       }
     }}};
</script>
```

e.which==13은 키업된 이벤트 중 엔터키를 눌렀을 때 해당 이름으로 사원들을 찾습니다.

[소스-SearchCustomersById.vue]

```
<template>
  <div>
    <div class='search_box'>
      <select name='id' v-model="cid" @change='findbyid' v-cloak>
        <option v-for="emp in info" v-bind:value="emp.id">{{ emp.name }}</option>
      </select>
    </div>
    <table class='list_table'>
      <col width="10%"><col width="20%">
      <col width="20%">><col width="20%">><col width="10%">
      <tr>
        <th>사원 아이디</th>
        <th>사원명</th>
        <th>부서</th>
        <th>직책</th>
        <th>연봉</th>
      </tr>
      <tr>
        <td v-html="cemp.id"></td>
        <td v-html="cemp.name"></td>
        <td v-html="cemp.dept_name"></td>
        <td v-html="cemp.title"></td>
        <td>{{ cemp.salary | salarydecimal }}</td>
      </tr>
    </table>
  </div>
</template>
```

v-model="cid" @change='findbyid' select 에서 해당 아이디를 찾으면 아이디에 대한 사원을 찾습니다. 아이디에 대한 사원을 찾아 cemp 에 저장후 해당 정보를 출력합니다. <select>를 이용하여 이름을 보이고, 이름을 선택하면 해당 아이디를 이용하여 사원의 정보를 찾습니다.

[소스-SearchCustomersById.vue]

```
<script>
  import http from "../http-common";
  export default {
    name: "search-customer-id",
    data () {
      return {
        info: [],
        loading: true,
        errored: false ,
        cid:'',
        cemp:{}
      }
    },
    mounted () {
      http
        .get('/findAllEmployees')
        .then(response => (this.info = response.data))
        .catch(error => {this.errored = true}).finally(() => this.loading = false);
    },
    methods: {
      findbyid() {
        http
          .get('/findEmployeeById/'+this.cid)
          .then(response => (this.cemp = response.data))
          .catch(error => {this.errored = true}).finally(() => this.loading = false);
      }
    }
  };
</script>
```

“findAllEmployees” 메서드로 모든 사원의 정보를 얻은 후 <select>에 이름을 보이게 한 후, 선택하면 선택된 아이디와 ‘findEployeeById’를 이용하여 해당 아이디에 속한 사원의 정보를 가져옵니다.

[소스-router.js]

```
import Vue from "vue";
import Router from "vue-router";
import CustomersList from "./components/CustomersList.vue";
import DetailCustomer from "./components/DetailCustomer.vue";
import SearchCustomers from "./components/SearchCustomers.vue";
import SearchCustomersById from "./components/SearchCustomersById.vue";
Vue.use(Router);
export default new Router({
  mode: "history",
  routes: [
    {
      path: "/", name: "customers",
      alias: "/customer", component: CustomersList
    },
    {
      path: '/detailcustomer/:id',
      name: 'detailcustomer',
      component: DetailCustomer,
      props: true,
    },
    {
      path: "/search",name: "search",
      component: SearchCustomers
    },
    {
      path: "/searchbyid", name: "searchbyid",
      component: SearchCustomersById
    }]
});
```

<router-view/> 에 SearchCustomers.vue, SearchCustomersById.vue 를 보이기 위해 router 에 SearchCustomers, SearchCustomersById 를 추가합니다.

11.6 사원정보 추가

11.6.1 사원 추가

[hellovues2]를 복사한 후 복사본을 [hellovues3]로 수정하십시오 그리고 [package.json]와[package-lock.json] 의 이름을 아래와 같이 hellovues3 로 수정하십시오.

```
{
   "name": "hellovues3",
   ...
}
```

[소스-AddCustomer.vue]

```
<template>

<div>
<div v-if="!submitted">
<form action="" method="post" id="_frmForm" name="frmForm"
@submit.prevent="addEmployee">
<table class="content_table">
<colgroup>
                    <col style="width:30%;" />
                    <col style="width:70%;" />

</colgroup>
<tr>
<th>이름</th>
<td><input data-msg="이름" type="text" name="name" id="_name"   v-model="cname"
style="width:30%"/></td>
</tr>
<tr>
<th>이메일</th>
<td><input data-msg="이메일" type="text" name="mailid"   id="_mailid" size="20" v-
model="cmailid" style="width:30%"/></td>
</tr>
```

```
<tr>
<th>고용일</th>
<td><input data-msg="고용일" type="date" name="start_date"   id="_start_date" size="30" v-
model="cstart_date" style="width:30%"/></td>
</tr>
<tr>
<tr>
<th>관리자</th>
<td>
 <select name='manager_id' v-model="cmanager_id" style="width:30%">
  <option v-for="emp in info" v-bind:value="emp.id">
   {{ emp.id }}:{{ emp.name }}
  </option>
</select>
</td>
</tr>
<tr>
<th>직책</th>
<td>
 <select name='title' v-model="ctitle" style="width:30%">
  <option v-for="emp in titls" v-bind:value="emp.title">
   {{ emp.title }}
  </option>
</select>
</td>
</tr>
<tr>
<th>부서</th>
<td>
 <select name='dept_id' v-model="cdept_id" style="width:30%">
  <option v-for="dept in deps" v-bind:value="dept.dept_id">
  {{ dept.dept_id   }}: {{ dept.name }}
  </option>
</select>
</td>
</tr>
```

```
<tr>
<th>월급</th>
<td><input data-msg="월급" type="number" name="salary"   id="_salary" size="30" min="0" v-model="csalary" style="width:30%"/></td>
</tr>
<tr>
<th>커미션</th>
<td><input data-msg="커미션" type="number" name="commission_pct"   id="_commission_pct" size="30" min="0" v-model="ccommission_pct" style="width:30%"/></td>
</tr>
<tr>
<td colspan="2" style="height:50px; text-align:center;">
<button type="submit" name="button">사원추가</button></td>
</tr>
</table>
</form>
</div>

<div v-else>
        <h4>성공적으로 입사처리를 완료하였습니다!</h4>
        <button class="btn btn-success" v-on:click="newCustomer">새로 사원 추가하기</button>
    </div>
</div>

</template>

<script>
import http from "../http-common";

export default {
        name: "add-customer",
        data() {
                return {
                        info: null,
                        loading: true,
                        errored: false,
```

```
                deps:null,
                titls:null,
                ccommission_pct:0,
                cdept_id:0,
                cmailid:'',
                cmanager_id:0,
                cname:'',
                csalary:0,
                cstart_date:'',
                ctitle:'',
                submitted: false
        };
},
filters: {
        salarydecimal (value) {
                var a=parseInt(value);
                return a.toFixed(2);
        }
},
mounted () {
http
        .get('/findAllEmployees')
        .then(response => (this.info = response.data))
        .catch(error => {
        //console.log(error)
                this.errored = true
        })
        .finally(() => this.loading = false);
http
        .get('/findAllDepartments')
        .then(response => (this.deps = response.data))
        .catch(error => {
        //console.log(error)
                this.errored = true
        })
        .finally(() => this.loading = false);
```

```
http
        .get('/findAllTitles')
        .then(response => (this.titls = response.data))
        .catch(error => {
        //console.log(error)
                this.errored = true
        })
        .finally(() => this.loading = false);
},
methods: {
addEmployee() {
        if(this.cname==''){ alert('아이디를 입력하세요.'); return ;}
        if(this.cdept_id==''){ alert('부서를 선택하세요.'); return ;}
        if(this.cmailid==''){ alert('이메일을 선택하세요.'); return ;}
        if(this.cmanager_id==''){ alert('관리자를 선택하세요.'); return ;}
        if(this.csalary==''){ alert('월급을 입력하세요.'); return ;}
        if(this.cstart_date==''){ alert('입사일을 선택하세요.'); return ;}
        if(this.ctitle==''){ alert('직책을 선택하세요.'); return ;}

        http.post('/addEmployee', {
                commission_pct: this.ccommission_pct,
                dept_id: this.cdept_id,
                mailid: this.cmailid,
                manager_id: this.cmanager_id,
                name: this.cname,
                salary: this.csalary,
                start_date:this.cstart_date,
                title: this.ctitle
        }
        ).then(response => {
                if (response.data.state=='succ') {
                        alert("입사처리를 하였습니다.");
                        //console.log(response.data.name);
                }else{
                        alert("입사처리를 하지 못했습니다.");
                        //console.log(response.data.name);
```

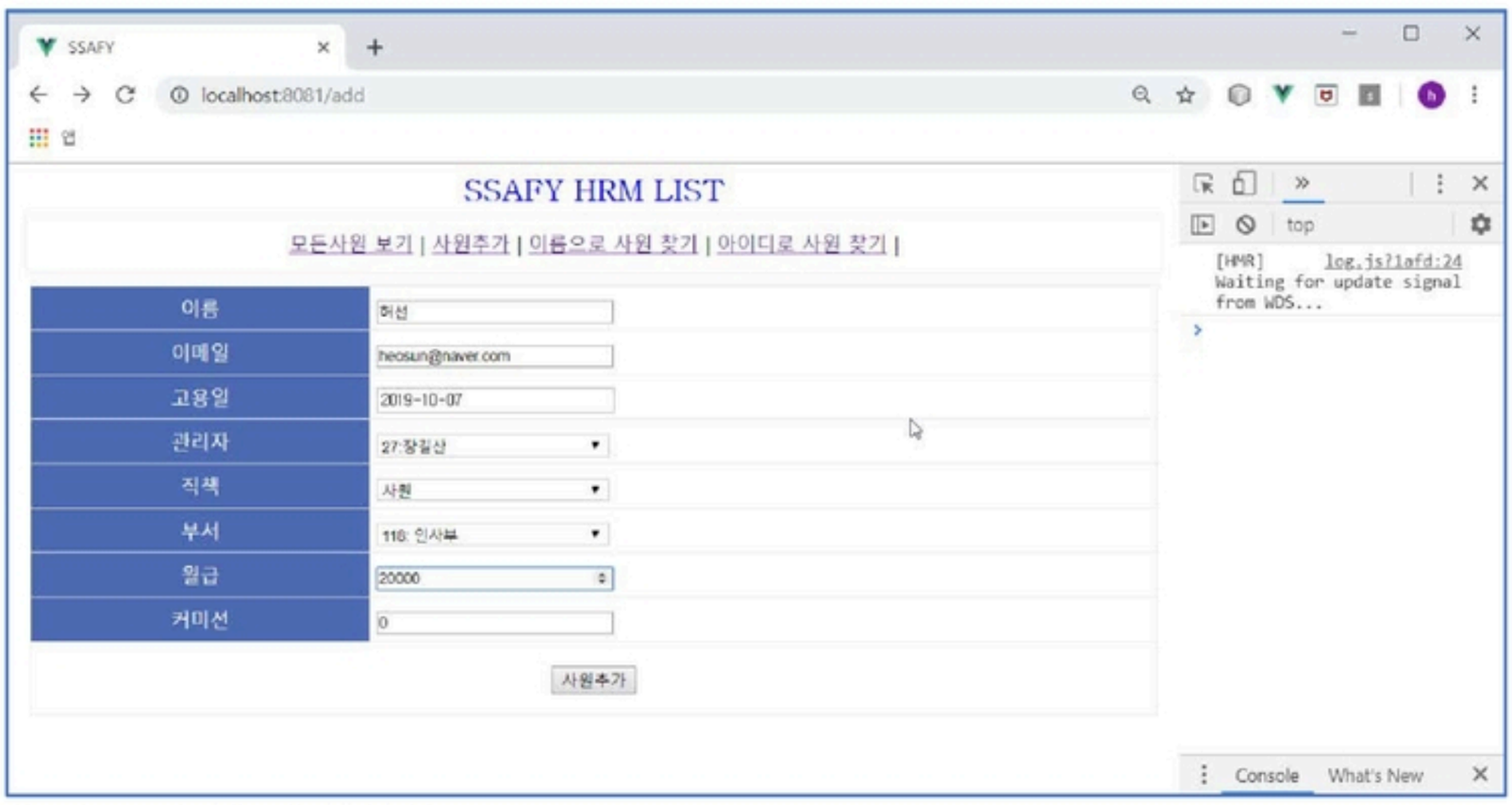
```
                                        }
                    });
                    this.submitted = true;
            },
            newCustomer() {
                this.submitted=false,
                this.info=null,
                    this.loading=true,
                    this.errored=false,
                    this.deps=null,
                    this.titls=null,
                    this.ccommission_pct=0,
                    this.cdept_id=0,
                    this.cmailid='',
                    this.cmanager_id=0,
                    this.cname='',
                    this.csalary=0,
                    this.cstart_date='',
                    this.ctitle=''
            }
    }
};
</script>
```

사원정보를 입력하기 위한 기본 입력정보를 얻습니다. {키 1:value1, 키 2:value2}의 형식으로 사원의 정보를 입력 후 **addEmployee** 를 호출하여 정보를 입력합니다.

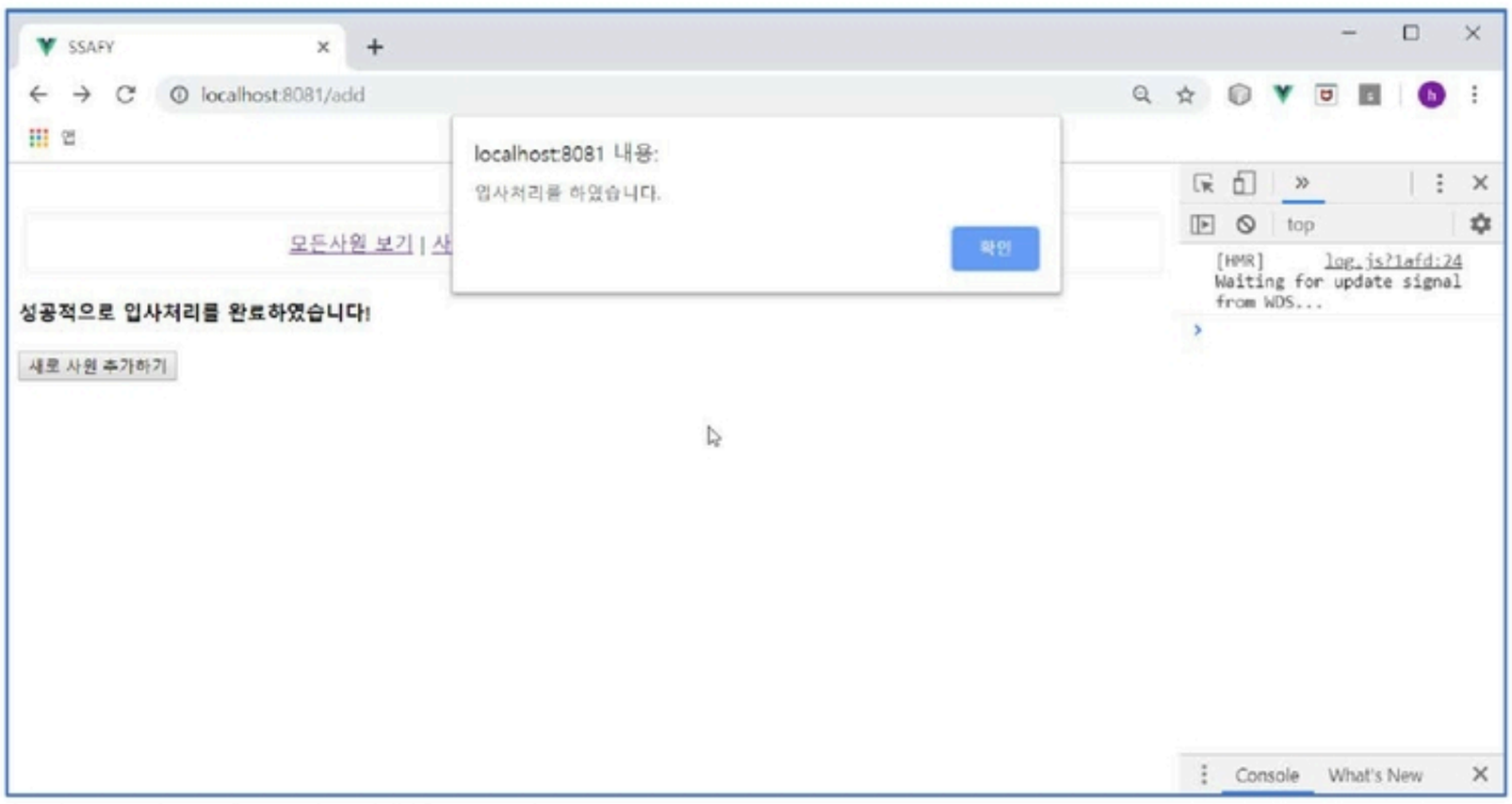
[소스-router.js]

```
{
        path: "/add",
        name: "add",
        component: AddCustomer
}
```

[router.js]에 AddCustomer 를 추가합니다.

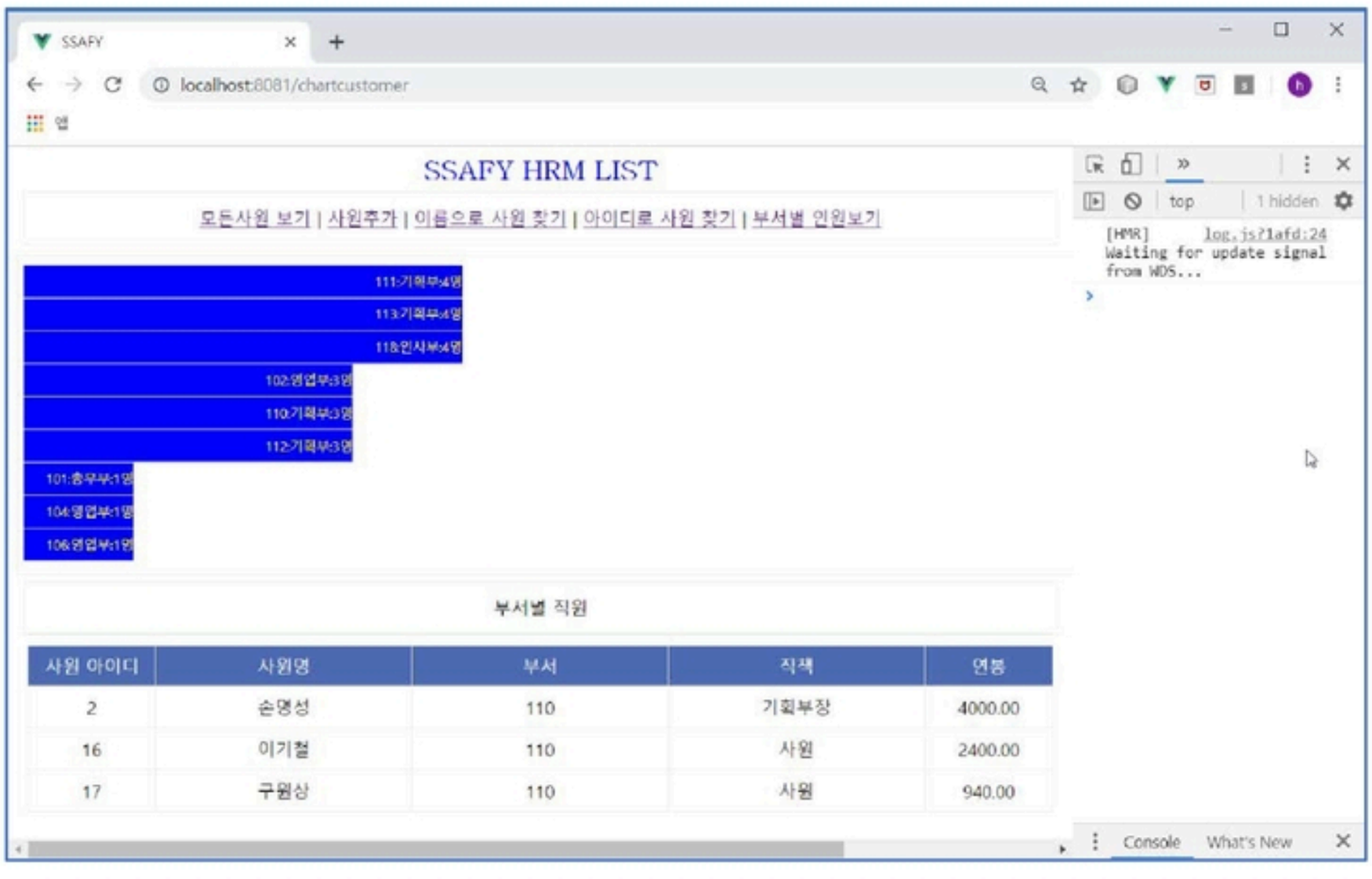


[그림 11.19] 사원추가하기



[그림 11.20] 사원 추가 후 화면

부서별 인원보기입니다.



[그림 11.21] 부서별 인원보기

[소스-router.js]

```
{path: "/chartcustomer",
 name: "chartcustomer",
 component: ChartCustomer}
```

[router.js]에 ChartCustomer 를 추가합니다.

[소스-App.vue]

```
<nav>
<router-link class="btn btn-primary" to="/">모든사원 보기</router-link> |
<router-link class="btn btn-primary" to="/add">사원추가</router-link> |
<router-link class="btn btn-primary" to="/search">이름으로 사원 찾기</router-link> |
<router-link class="btn btn-primary" to="/searchbyid">아이디로 사원 찾기</router-link> |
<router-link class="btn btn-primary" to="/chartcustomer">부서별 인원보기</router-link></nav>
</div>
        <router-view/>
```

App.vue 에 chartcustomer 를 추가합니다.

[소스-ChartCustomer.vue]

```
<template>
<div>
<table class="list_table">
<col style="width:80%;" />
<tr>
<td>
<svg :width="totalmember()*50" :height="depcountheight()">
  <template   v-for="(depcount, index) in depcounts">
    <g :transform="transform(0, index)" @mouseover="showdept(depcount.dept_id)">
    <rect :width='100*depcount.count' height='29'></rect>
    <text :x='100*depcount.count' y='15' dy='.35em'>
{{depcount.dept_id}}:{{depcount.name}}:{{depcount.count}}명</text>
    </g>
   </template>
  </svg>
</td>
</tr>
</table>
```

svg 에 막대의 길이, 높이를 입력합니다. 그리고 부서 만큼 막대(rect)를 그립니다. 막대 위에 마우스를 올려놓으면(mouseover) 해당 부서의 사원들을 보입니다.

[소스-ChartCustomer.vue]

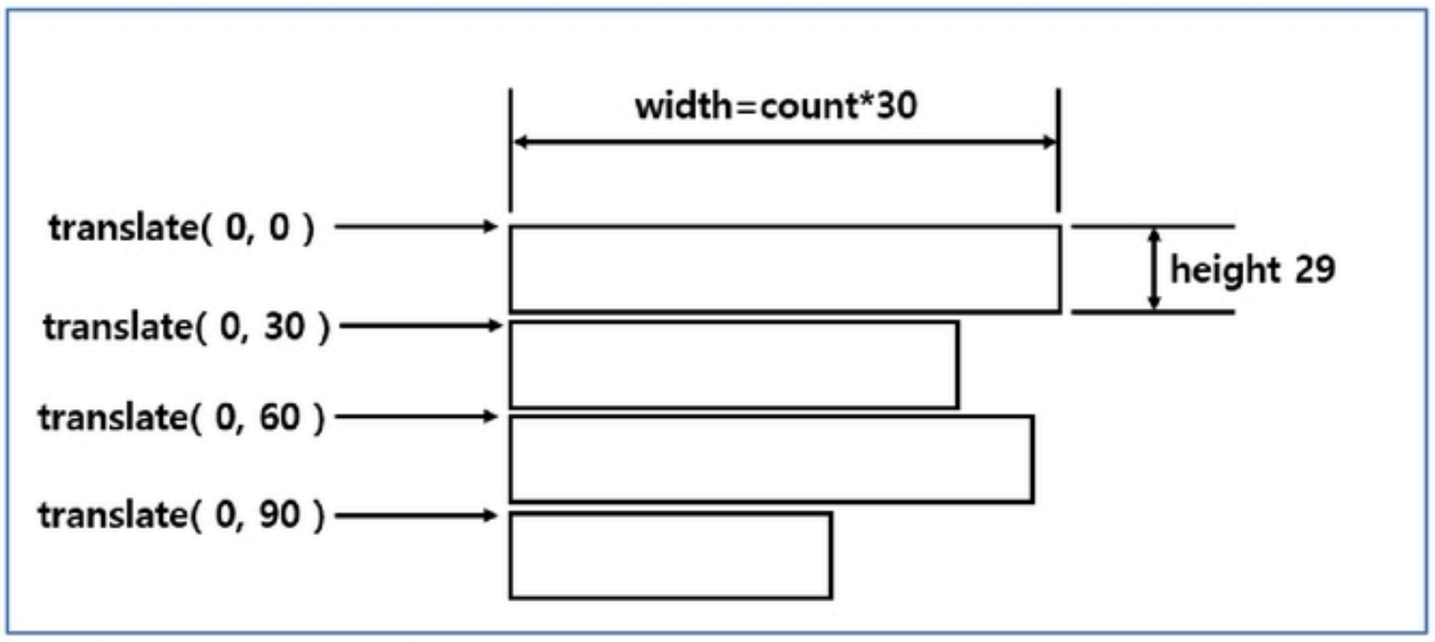
```
<div v-if='showme()'>
<div class='search_box'>부서별 직원</div>
<table class='list_table'>
<col width="10%"><col width="20%">
<col width="20%"><col width="20%"><col width="10%">
<tr>
  <th>사원 아이디</th>
  <th>사원명</th>
  <th>부서</th>
  <th>직책</th>
  <th>연봉</th>
</tr>
<tr v-for="emp in info" class = "nicecolor">
  <td v-html="emp.id"></td>
  <td v-html="emp.name"></td>
  <td v-html="emp.dept_id"></td>
  <td v-html="emp.title"></td>
  <td>{{ emp.salary | salarydecimal }}</td>

</tr>
</table>
</div>
</div>
</template>
```

info 리스트에 있는 사원정보들을 얻은 후 사원정보를 출력합니다.

[소스- ChartCustomer.vue]

```
<script>
import http from "../http-common";
export default {
  name: "chart-customer",
  data () {
        return {
          deptid2:0,
          depcounts: [],
          loading: true, errored: false ,
          info:[]
        }},
methods: {
  totalmember(){
    var a=0;
    for(var i=0; i<this.depcounts.length;i++){
      a+=this.depcounts[i].count;
    }
    return a;
  },
  transform(val1, val2) {
      return `translate(${val1}, ${val2*30})`;
  },
  depcountheight(){
    return this.depcounts.length*30;
  },
  showme(){
    return this.deptid2!=0;
  },
```

사원수를 모두 더한 후, 비율을 구합니다. svg 에서 원하는 위치로 이동할 때 translate()를 이용합니다.
transform(val1, val2)를 호출하여 원하는 위치로 이동합니다.
막대 길이는 사원수*30, 높이는 29, 부서만큼 y 좌표를 30 씩 이동시켜 막대를 그립니다.



[그림 11.22] 차트 기본 설정

[소스- ChartCustomer.vue]

```
showdept(deptid){
        this.deptid2=deptid;
        http.get('/findDepartmentsBydeptid/'+deptid)
           .then(response => (this.info = response.data))
           .catch(error => {this.errored = true})
           .finally(() => this.loading = false)
    }
},
mounted () {
   http.get('/findAllDepCounts')
      .then(response => (this.depcounts = response.data))
      .catch(error => {this.errored = true})
      .finally(() => this.loading = false);
},
filters: {
      salarydecimal (value) {
      var a=parseInt(value);
        return a.toFixed(2);
     }}
};
</script>
```

렌더링되기 전에 부서별 사원수를 얻습니다. 그리고 비율을 구하여 원하는 막대길이를 만들고, 원하는 위치로 이동시킨 후 막대를 그립니다.

11.7 사원정보 수정

[hellovues3]를 복사한 후 복사본을 [hellovues4]로 수정하십시오 그리고 [package.json]와[package-lock.json] 의 이름을 hellovues4 로 수정하십시오.

[소스-router.js]

```
{
        path: '/updatecustomer/:id',
        name: 'updatecustomer',
        component: UpdateCustomer,
        props: true,
}
```

[router.js]에 UpdateCustomer 를 추가합니다. DetailCustomer 에서 선택된 아이디를 props 에 저장한 후, UpdateCustomer 에서 저장된 id 를 찾아서 사용합니다.

[소스-DetailCustomer.vue]

```
<table>
 <tr @click="show_detail(cemp.id)">
<td v-html="cemp.id"></td>
  <td v-html="cemp.name"></td>
  <td v-html="cemp.dept_id"></td>
  <td v-html="cemp.title"></td>
  <td>{{ cemp.salary | salarydecimal }}</td>
</tr>
</table>
```

<tr>을 클릭하면 해당 아이디를 UpdateCustomer 의 props 에 저장 한 후 UpdateCustomer 는 렌더링 하기 전에 props 에서 저장된 아이디를 이용하여 사원정보를 가져온 후 렌더링합니다.

[소스-DetailCustomer.vue]

```
props: ['id'],
methods:{
        show_detail:function(employeeid){
          alert(employeeid+"수정하기");
          //아래를 수정했다.
          this.$router.push('/updatecustomer/' + employeeid);
        },
        show_init:function(){
          http
          .get('/findEmployeeById/'+this.id)
          .then(response => (this.cemp = response.data))
          .catch(error => {
            this.errored = true
          })
          .finally(() => this.loading = false);

          http
            .get('/findEmployeesByManagerId/'+this.id)
            .then(response => (this.info = response.data))
            .catch(error => {
              this.errored = true
            })
            .finally(() => this.loading = false);
          }
      },
      mounted () {
        this.show_init();
      }
};
```

DetailCustomer 에서 해당사원을 선택하면 id 를 props 에 저장한 후 UpdateCustomer 로 이동합니다. 이 때 라우터를 이용합니다. UpdateCustomer 에서는 props 에 저장되었던 id 를 이용하여 사원정보를 보입니다. 사원정보를 수정 후 수정 버튼을 눌러 수정할 수 있습니다.

[소스-UpdateCustomer.vue]

```
<template>
<div>
<div class='search_box'>
<h3><span style='color:blue;'>사원정보수정</span></h3>
<button class="btn btn-success" @click="show_detail()">사원 정보 보기</button>
</div>
<div v-if="!submitted">
<form action="" method="post" id="_frmForm" name="frmForm"
@submit.prevent="updateEmployee">
<table class="content_table">
<colgroup>
                    <col style="width:30%;" />
                    <col style="width:70%;" />

</colgroup>
<tr>
<th>이름</th>
<td><input data-msg="이름" type="text" name="name" id="_name" v-model="customer.cname"
style="width:30%"/></td>
</tr>
<tr>
<th>이메일</th>
<td><input data-msg="이메일" type="text" name="mailid"   id="_mailid" size="20"
v-model="customer.mailid" style="width:30%"/></td>
```

수정버튼을 클릭하면 updateEmployee 를 호출합니다.

[소스-UpdateCustomer.vue]

```
import http from "../http-common";
export default {
        name: "updatecustomer",
    props: ['id'],
        data() {
                return {
                        info: null,
                        loading: true,
                        errored: false,
                        deps:null,
                        titls:null,
                        customer: {
                                commission_pct:0,
                                dept_id:0,
                                mailid:'',
                                manager_id:0,
                                cname:'',
                                salary:0,
                                start_date:'',
                                  title:''
                        },
                        submitted: false
                };
        },
```

customer 객체를 선언하여 여러 멤버 필드를 저장할 수 있도록 합니다.

[소스-UpdateCustomer.vue]

```
updateEmployee() {
        if(this.customer.cname==''){ alert('아이디를 입력하세요.'); return ;}
        if(this.customer.dept_id==''){ alert('부서를 선택하세요.'); return ;}
        if(this.customer.mailid==''){ alert('이메일을 선택하세요.'); return ;}
        if(this.customer.manager_id==''){ alert('관리자를 선택하세요.'); return ;}
        if(this.customer.salary==''){ alert('월급을 입력하세요.'); return ;}
        if(this.customer.start_date==''){ alert('입사일을 선택하세요.'); return ;}
        if(this.customer.title==''){ alert('직책을 선택하세요.'); return ;}
        http.post('/updateEmployee', {
            commission_pct: this.customer.commission_pct,
            dept_id: this.customer.dept_id,
            mailid: this.customer.mailid,
            manager_id: this.customer.manager_id,
            name: this.customer.cname,
            salary: this.customer.salary,
            start_date:this.customer.start_date,
            title: this.customer.title,
            id:this.id
        } ).then(response => {
            if (response.data.state=='succ') {
                alert("정보를 수정 하였습니다.");
            }else{
                alert("정보를 수정 하지 못했습니다.");
            }
        });
        this.submitted = true;
    },
```

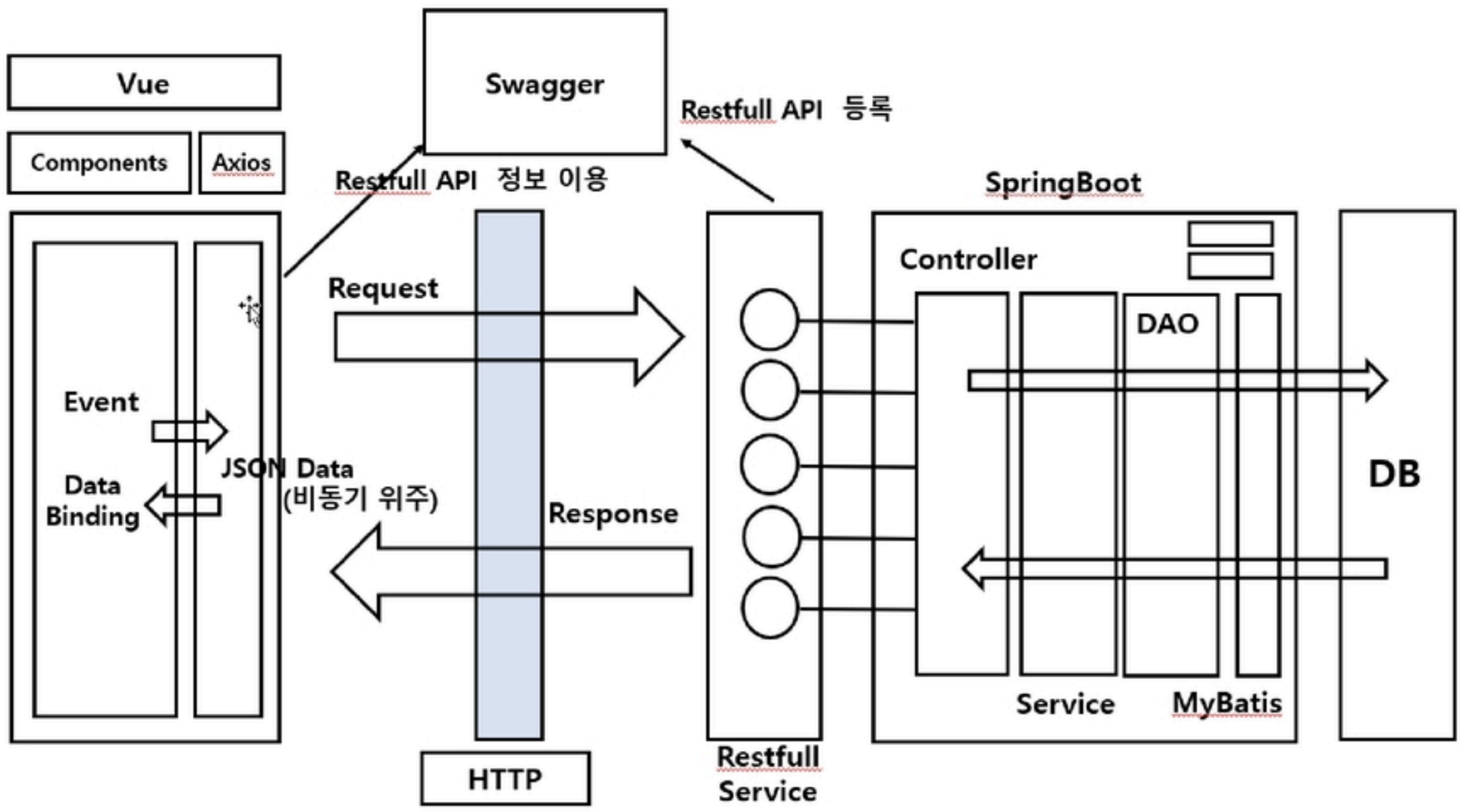
해당사원의 정보를 수정 합니다. 수정하기 위해{}에 사원의 아이디, 사원의 이름, 이메일, 연봉 등을 key:value 형식으로 입력합니다. 그리고 updateEmployee API 를 이용하여 사원의 정보를 수정합니다.

12. Vue 프로젝트 2

12.1 HRM 아키텍쳐

12.1.1 HRM 아키텍쳐

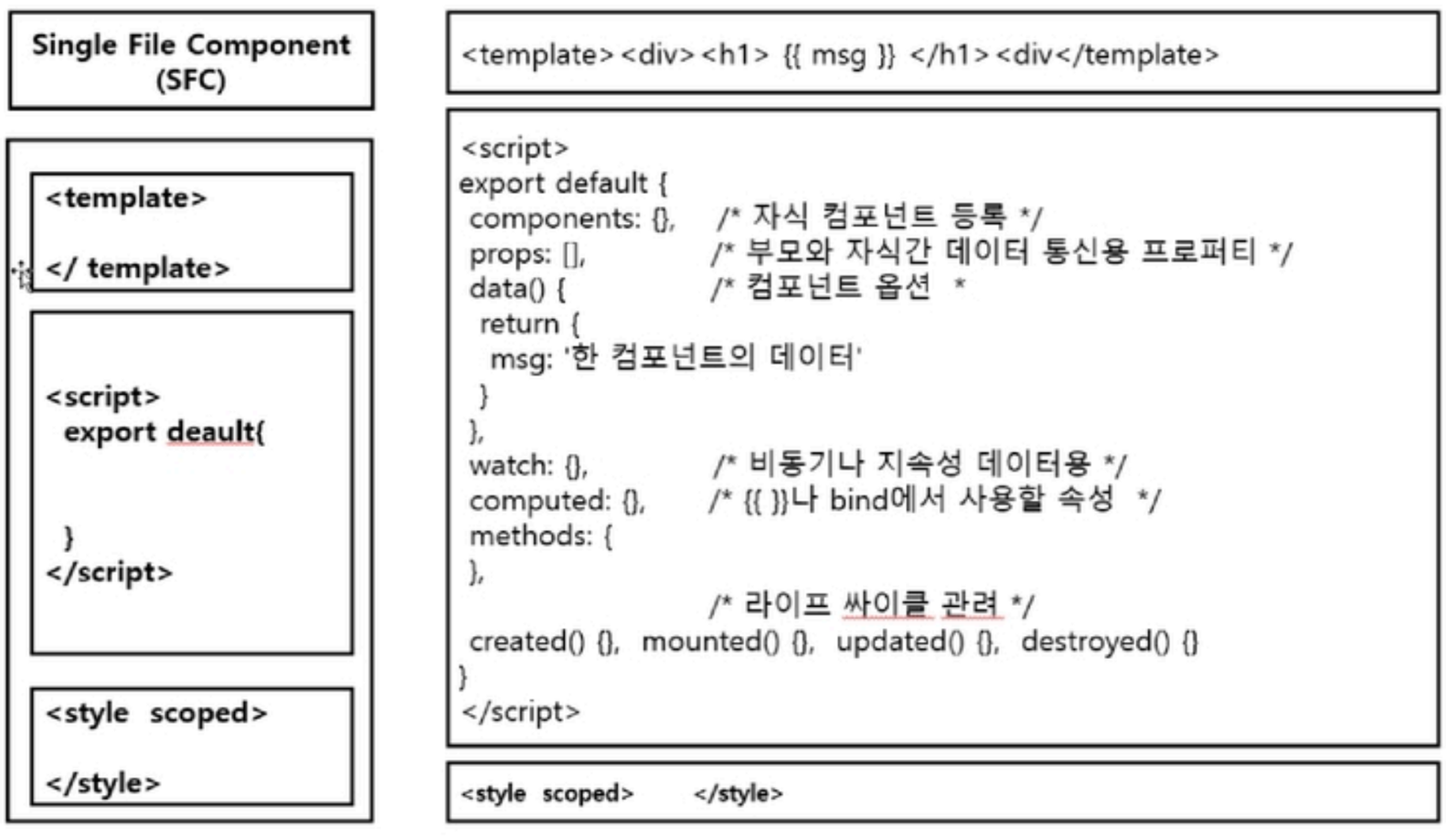
뷰 HRM 인사관리 시스템은 클라이언트와 서버로 구성됩니다. SpringMVC 는 서버에서 클라이언트코드를 생성 후 브라우저로 보내고 브라우저에서 렌더링하는 동기 프로그래밍이 기본입니다. 뷰 HRM 은 비동기 프로그래밍으로 서버에서는 Data 만 얻어옵니다. 뷰 HRM 은 SpringMVC 에서 View 를 제거한 것으로 보십시오. SpringBoot 는 Restfull 을 이용하여 Data 를 브라우저로 보내고, 브라우저에서 Data 를 HTML 과 함께 렌더링합니다. 뷰 HRM 에서 요청과 응답을 Restfull 서비스로 합니다. 실질적인 처리는 SpringBoot 에서 컨트롤이 DAO 를 호출하여 데이터를 DB 에 저장하거나 가져옵니다.



[그림 12.1] HRM 아키텍쳐

12.1.2 SFC(Single File Component) 구조

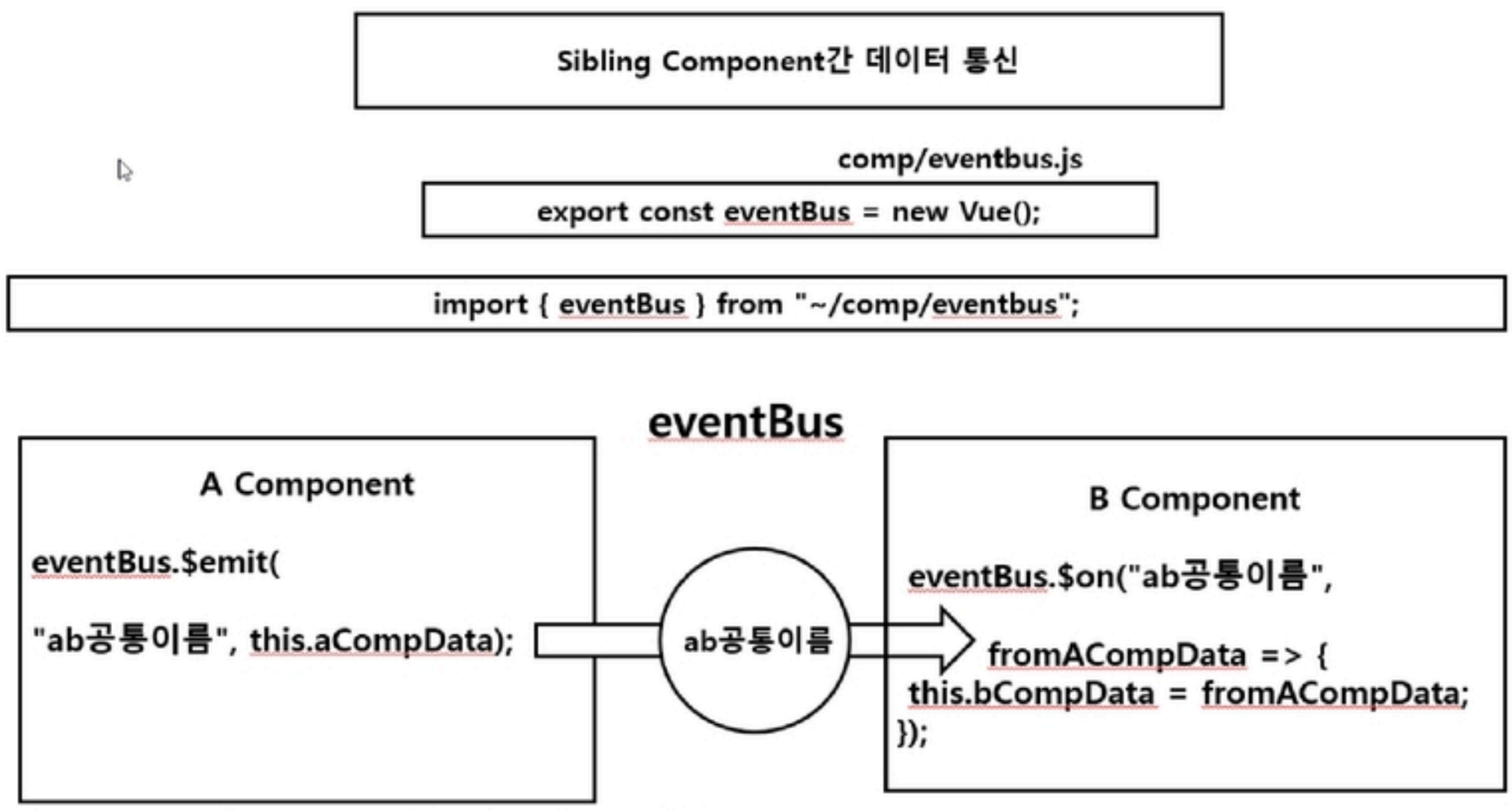
SFC 는 3 영역으로 구성됩니다. HTML 과 Data 를 출력하는 〈template〉과 View 코드(JS 포함)를 선언하는 〈script〉, CSS 를 선언하는 〈style〉로 구성됩니다. export default{}내부에 컴포넌트의 구성 옵션을 선언합니다. 구성옵션은 앞에서 배운 data, watch, props, methods, 라이프 사이클 관련 hook 메서드 등으로 구성됩니다. SFC 구조로 만들면 다른 컴포넌트에서 등록하고 사용할 수 있습니다.



[그림 12.2] SFC 구조

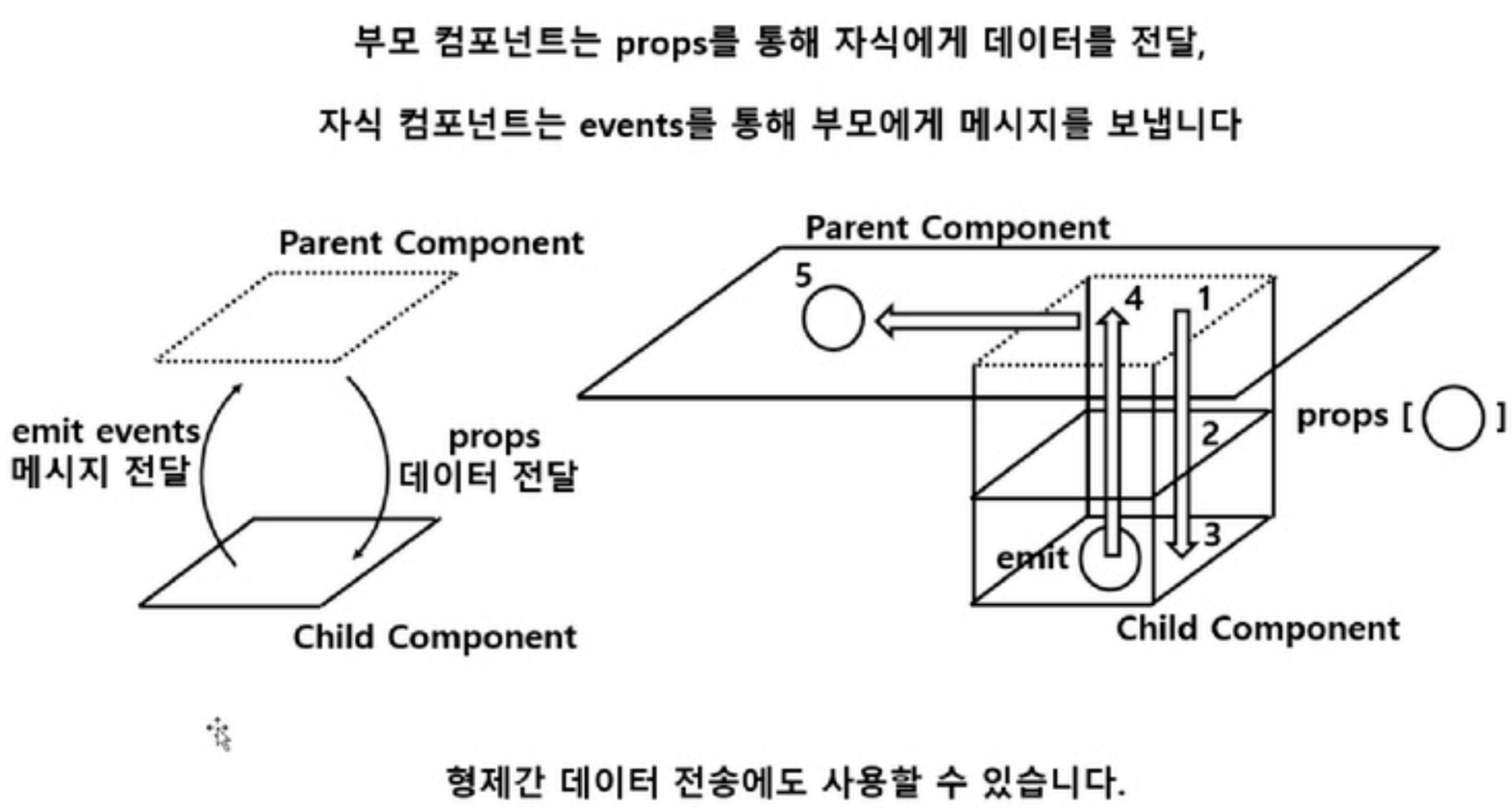
12.1.2 EventBus

EventBus 는 컴포넌트 사이에 데이터 통신을 하기 위해 사용합니다. 보통 형제(sibling) 컴포넌트 사이에 데이터 통신을 위해 사용하지만 부모와 자식간에도 사용할 수 있습니다.
우선 EventBus 를 생성합니다. A 컴포넌트에서 이벤트가 발생할 때 eventBus.$emit("공통이름", B 에 전달하려는 객체) 형식을 이용하여 B 컴포넌트로 데이터를 전달합니다. B 컴포넌트에서는 eventBus.$on("공통이름", A 에서 B 로 전달하려는객체 val => {val 처리 })형식으로 이벤트버스에 등록한 공통이름을 통해 데이터를 전달 받을 수 있습니다.



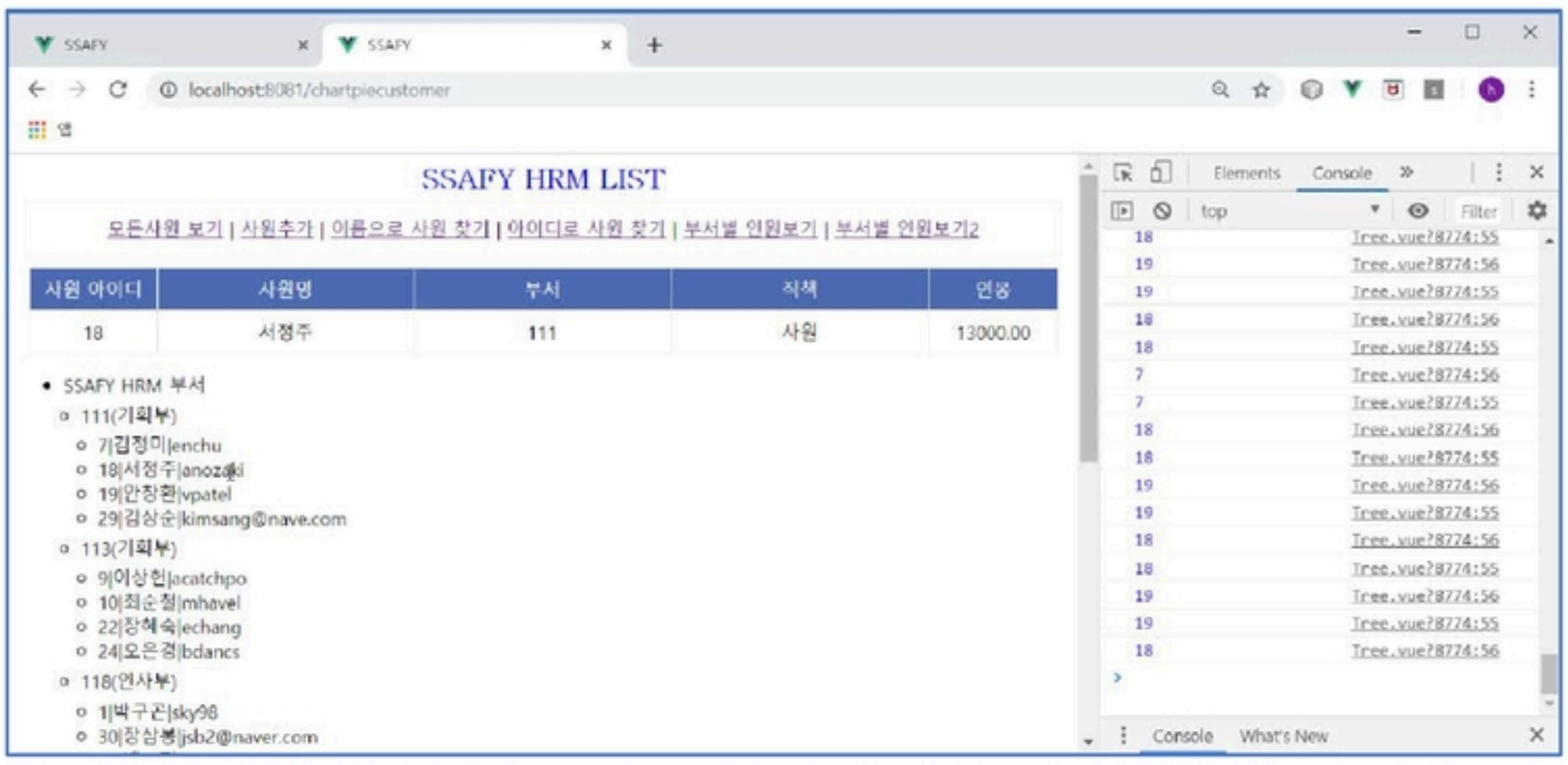
[그림 12.3] 컴포넌트 사이에 데이터 통신

부모와 자식 컴포넌트 사이의 데이터 통신을 할 때는 props 와 emit 을 사용합니다. 부모컴포넌트는 props 를 통해 자식에게 데이터를 전달하고, 자식 컴포넌트는 events(emit)을 통해 부모에게 메시지를 보냅니다.

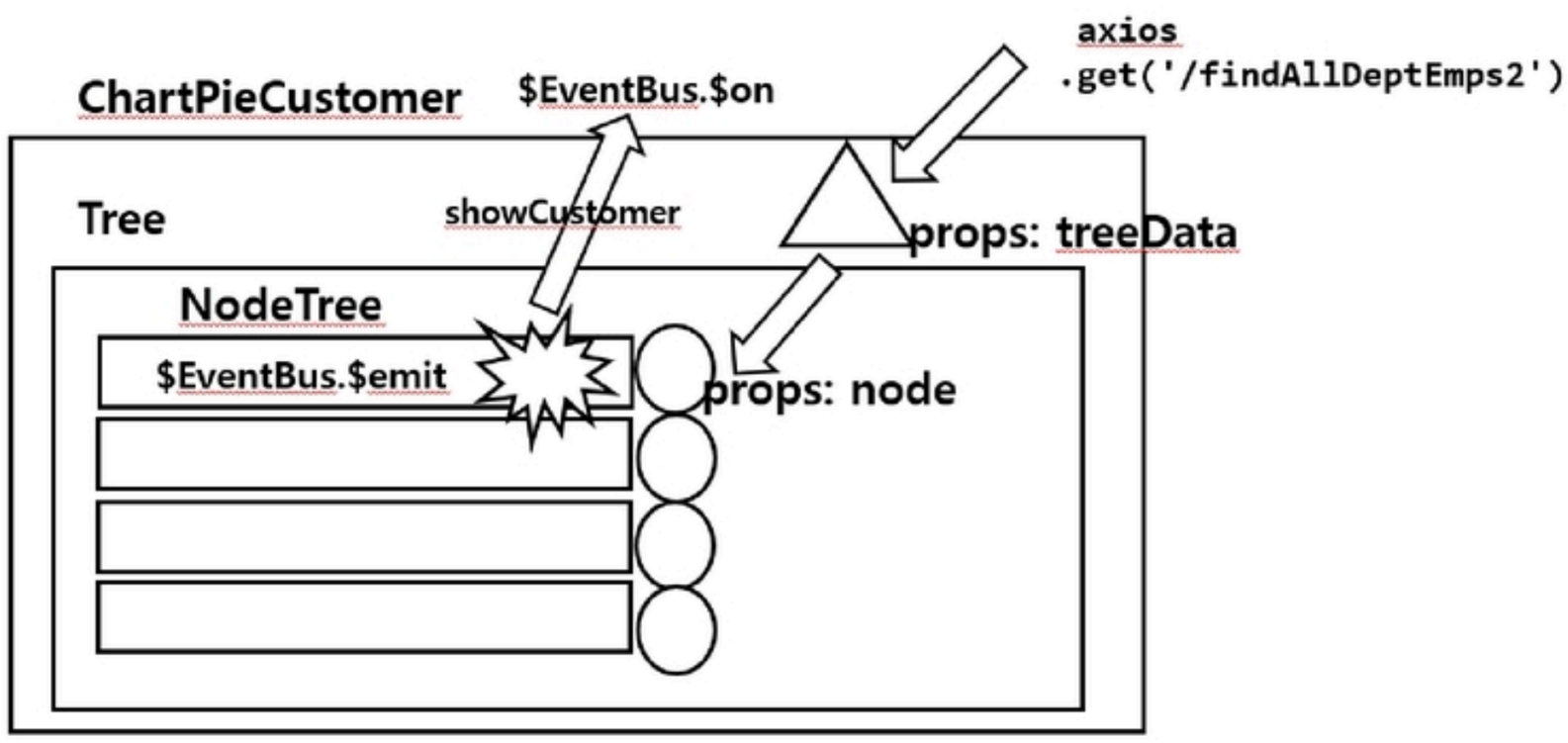


[그림 12.4] 부모 자식 컴포넌트 사이에 데이터 통신

12.1.3 컴포넌트 간에 데이터 전달



[그림 12.5] 트리 부모 컴포넌트와 노드 자식 컴포넌트 사이의 데이터 전달화면



[그림 12.6] 트리 부모 컴포넌트와 노드 자식 컴포넌트 사이의 데이터 전달원리

ChartPieCustomer 에서 axios.get('./findAllDeptEmps2')로 부서별 사원의 정보를 얻습니다. 얻은 트리 구조의 부서에 대한 사원 정보를 tree-data 로 Tree 에 전달합니다. props 에 '-'를 제외한 treeData 를 저장합니다. 그리고 이 정보들 중 해당 부서의 정보를 Tree 의 자식인 NodeTree 에 node 라는 이름으로 props 에 전달합니다. NodeTree 는 각 해당 부서별로 부서원들을 출력합니다. 이때 사원을 한명 선택하면 한 사원의 정보를 $EventBus.$emit 을 이용하여 부모 컴포넌트인 Tree 에 전달합니다. 그리고 Tree 에서 $EventBus.$on 으로 사원 아이디를 받아 사원 정보를 axios 로 찾고 해당 사원이 정보를 출력합니다.

[hellovues4]를 복사한 후 복사본을 [hellovues5]로 수정하십시오 그리고 [package.json]와[package-lock.json] 의 이름을 hellovues5 로 수정하십시오.

[소스-ChartPieCustomer.vue]

```
<template>
<div>
<div>
    <tree :tree-data="tree"></tree>
</div>
</div>
</template>
<script>
import http from "../http-common";
import Tree from "./Tree";
export default {
  name: "chartpiecustomer",
  data () {
        return {
          deptid2:0,
          depcounts: [],
          loading: true,
          errored: false ,
          info:[],
          tree: {
            deptname: "SSAFY HRM 부서",
            children: []
          }
        }
  },
```

트리구조로 만든 부서에 대한 사원 정보를 Tree 컴포넌트에 tree-data 이름으로 전달합니다.

[소스-ChartPieCustomer.vue]

```
components: {
        Tree
  },
  mounted () {
    http
      .get('/findAllDeptEmps2')
      .then(response => (this.tree.children= response.data))
      .catch(error => {
        this.errored = true
      })
      .finally(() => this.loading = false);
  }
};
</script>
```

this.tree.children= response.data 로 대입을 하면 아래와 같이 tree 에 부서에 대한 children 이 만들어 지고 그 안에 부서 이름이 나오고 다시 부서내의 사원들의 정보를 트리로 만듭니다. 이것을 Tree 자식 컴포넌트로 데이터를 보내 트리구조를 출력합니다.

```
tree: {
    deptname: "SSAFY HRM 부서",
    children: [
        {
          "deptname": "111(기획부)",
          "children": [
            {"deptname": "7|김정미|enchu"},
            {"deptname": "18|서정주|anozaki"},
            {"deptname": "19|안창환|vpatel"},
            {"deptname": "29|김상순|kimsang@nave.com"}
          ]
        }
    ]
}
```

[소스-Tree.vue]

```
<template>
<div><div>
<table class='list_table'>
<col width="10%"><col width="20%">
<col width="20%"><col width="20%"><col width="10%">
<tr>
  <th>사원 아이디</th><th>사원명</th><th>부서</th>
  <th>직책</th><th>연봉</th>
</tr>
 <tr>
  <td v-html="cempt.id"></td>
  <td v-html="cempt.name"></td>
  <td v-html="cempt.dept_id"></td>
  <td v-html="cempt.title"></td>
  <td>{{ cempt.salary | salarydecimal }}</td>
</tr>
</table>
  </div>
    <div class="tree">
    <ul class="tree-list">
      <node-tree :node="treeData"></node-tree>
    </ul>
  </div>
  </div>
</template>
```

부서별로 자식 컴포넌트인 NodeTree 를 이용하여 부서원들을 출력합니다. 부서원을 선택하면 해당부서원의 정보를 부모인 Tree 에 전달하고 Tree 는 이 정보를 받아서 테이블에 사원의 정보를 출력합니다.

[소스-Tree.vue]

```
<script>
import Vue from 'vue'
import NodeTree from "./NodeTree";
import http from "../http-common";
Vue.prototype.$EventBus = new Vue();
export default {
  data () {
    return {
      cempt:{salary:0}
    }
  },
  props: {
    treeData: {}
  },
  components: {
    NodeTree
  },
  filters: {
    salarydecimal (value) {
        var a=parseInt(value);
        return a.toFixed(2);
    }
  },
  watch: {
        'cempt' (value, oldValue) {
            console.log(oldValue.id)
            console.log(value.id)
        }
  },
```

Vue.prototype.$EventBus = new Vue(); 이벤트버스를 생성합니다. props를 이용하여 자식 컴포넌트로 데이터를 전달하고, $EventBus를 이용하여 자식에서 부모를 이벤트와 데이터를 전달할 수 있습니다. ‘cempt’의 데이터 변화를 관찰하기 위해 watch를 이용합니다.

[소스-Tree.vue]

```
mounted () {
        this.$EventBus.$on("showCustomer", val => {
            var index=val.indexOf('|');
            if(index!=-1){
              var ids=val.substr(0,index).trim();
             http
                .get('/findEmployeeById/'+ids)
                .then(response => {
                  this.cempt = response.data
                })
                .catch(error => {
                  this.errored = true
                })
                .finally(() => this.loading = false);

            }
          });
        }
};
</script>
```

자식인 NodeTree 에서 $EventBus 를 통하여 부모 Tree 로 전송한 데이터를 받아서 해당 사원의 정보를 출력합니다.

[소스-NodeTree.vue]

```
<template>
  <li class="node-tree">
    <span class="label" @mouseover="showLog(node.deptname)">
        {{ node.deptname }}</span>
    <ul v-if="node.children && node.children.length">
      <node v-for="child in node.children" :node="child" ></node>
    </ul>
  </li>
</template>
<script>
import Vue from 'vue';
Vue.prototype.$EventBus = new Vue();
export default {
  name: "node",
  props: {
    node: {}
  },
  },
   methods: {
    showLog: function(aaa){
      this.$EventBus.$emit('showCustomer', aaa);
    }
  }
};
</script>
```

트리 구조의 '부서별 사원인원 정보'를 부서별로 NodeTree 로 출력합니다. 사원을 클릭하면 해당사원의 정보를 $EventBus 를 통하여 부모 Tree 로 전송합니다.

12.1.4 다른 JS 연동하기

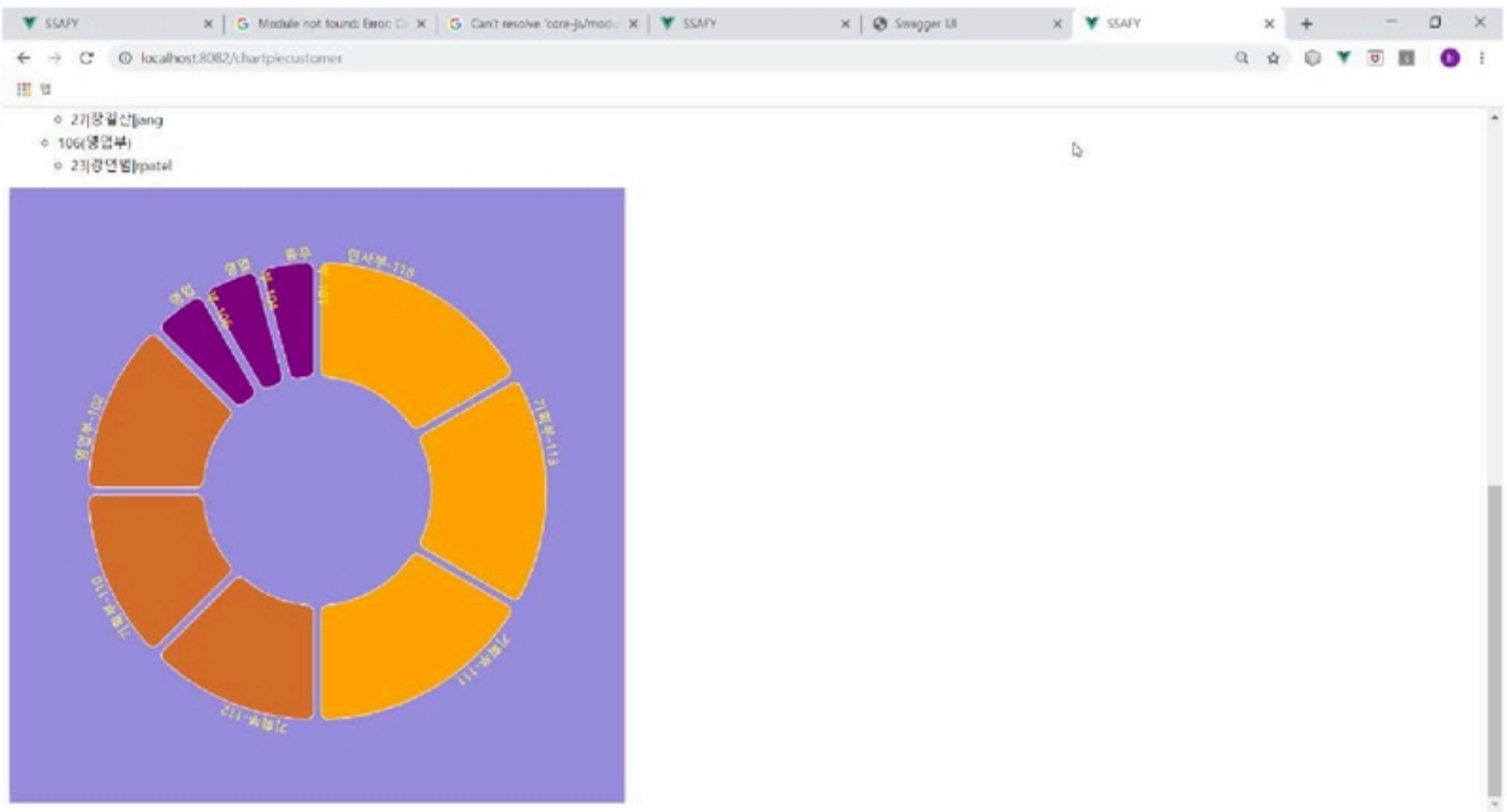
다른 JS 를 사용하기 위해서는 element 가 필요합니다. 이번 장에서는 그래픽을 위한 d3 을 사용합니다.

```
vue create hellovues6
      - default (babel, eslint) 엔터
 npm install -save
 npm install axios -save

 npm install d3 -save

 vue add element
```

- Import on Demand(아래 화살표를 이용하여 이동할 수 있습니다.)
- ko(아래 화살표를 이용하여 이동할 수 있습니다.)

vue add element 는 App.vue 를 새로 만들기 때문에, 원본이 삭제됩니다. App.vue 를 따로 복사해 두는 것이 좋습니다.

hellovues5 의 src 에 있는 것을 복사하여 hellovues6 의 src 에 놓습니다. Hellovues6 의 public 에 있는 것을 복사하여 hellovues6 의 public 에 놓습니다.



[그림 12.7] d3 사용하기

[소스-ChartPieCustomer.vue] 추가

```
<template>
<div><div>
    <tree :tree-data="tree"></tree>
    <pie :data="names" />
</div></div>
</template>
<script>
import http from "../http-common";
import Tree from "./Tree";
import Pie from "./Pie";
export default {
  name: "chartpiecustomer",
  data () {
        return {
          tree: {
            deptname: "SSAFY HRM 부서",
            children: []
          },
          names:[]
        }
  },
  components: {
      Tree,
      Pie
  },
  mounted () {
    http
      .get('/findAllDepCounts')
      .then(response => {this.names = response.data
      })
      .catch(error => {
        this.errored = true
      })
      .finally(() => this.loading = false);
```

```
  }
}
</script>
```

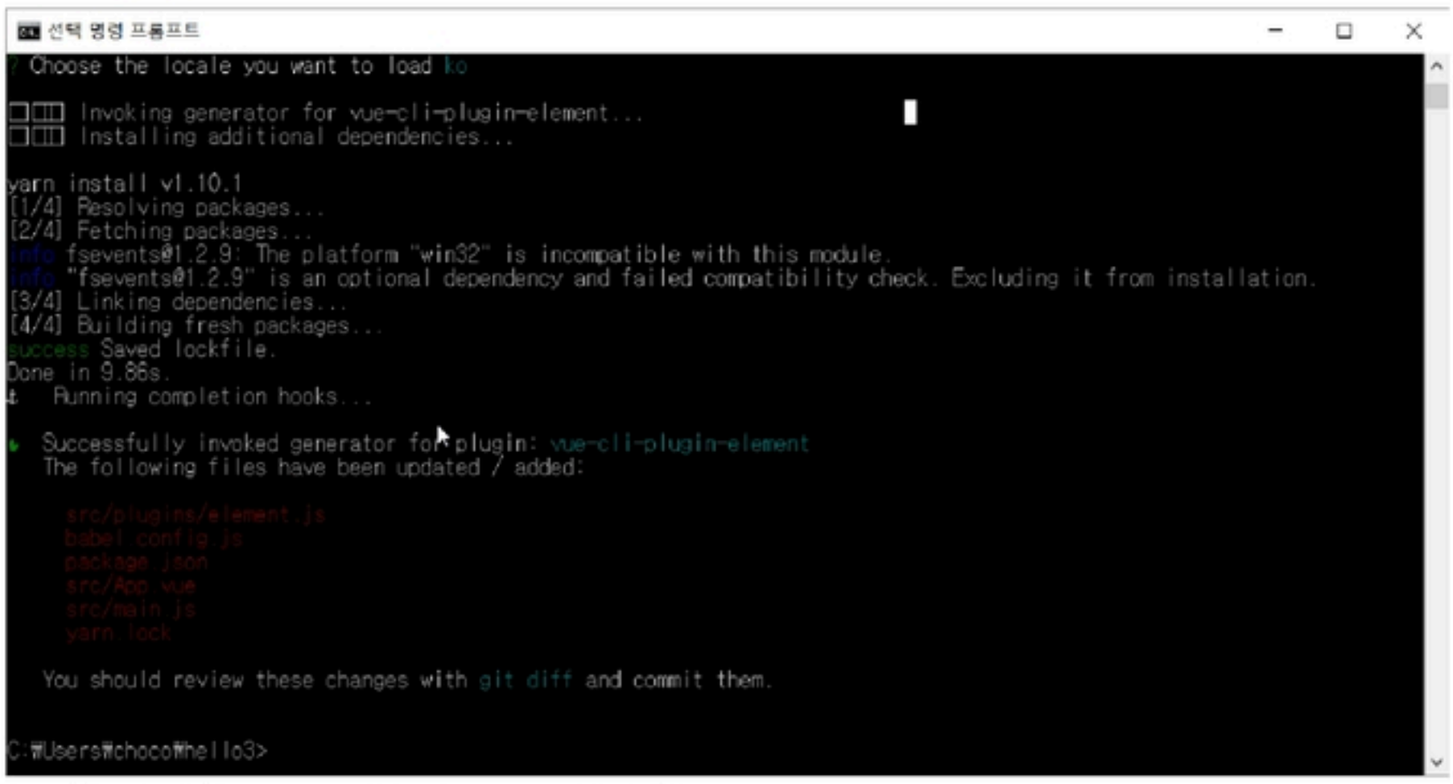
자식 컴포넌트인 Pie 를 선언하고 호출합니다 Pie 에 부서별 인원수 정보를 전달합니다.

[소스-Pie.vue]

```
<template>
  <svg width="700" height="700" ></svg>
</template>
<script>
//const d3 = require('d3');
import * as d3 from 'd3';

export default {
  props: ['data'],
  watch: {
    data: function(newData) {
      this.drawChart(newData);
    }
  },
  methods: {
    drawChart: function(data) {
```

부모 컴포넌트에서 전달한 부서별 인원수 정보를 props 에 저장합니다 그리고 이 정보를 이용하여 파이 차트를 작성합니다.

[참고] vue add element 실행



[그림 12.8] vue add element 이후

vue add element 이후 아래와 같은 내용이 추가되거나 수정됩니다. App.vue 가 삭제되고 새로 추가되니 원본을 복사해 두십시오.

src/plugins/element.js

babel.config.js

package.json

src/App.vue

src/main.js

yarn.lock

Eslint 를 standard 로 변경하면 엄격하게 check 하여 “”을 인식하지 못할 수 있습니다.
[package.json]의 eslintConfig 에서 standard 를 recommended 로 변경하면 “”를 인식가능하게 할 수 있습니다.

[소스-package.json]

```
<template>
  <svg width="700" height="700" ></svg>
</template>
<script>
//const d3 = require('d3');
import * as d3 from 'd3';

export default {
  props: ['data'],
  watch: {
    data: function(newData) {
      this.drawChart(newData);
    }
  },
  methods: {
    drawChart: function(data) {
```

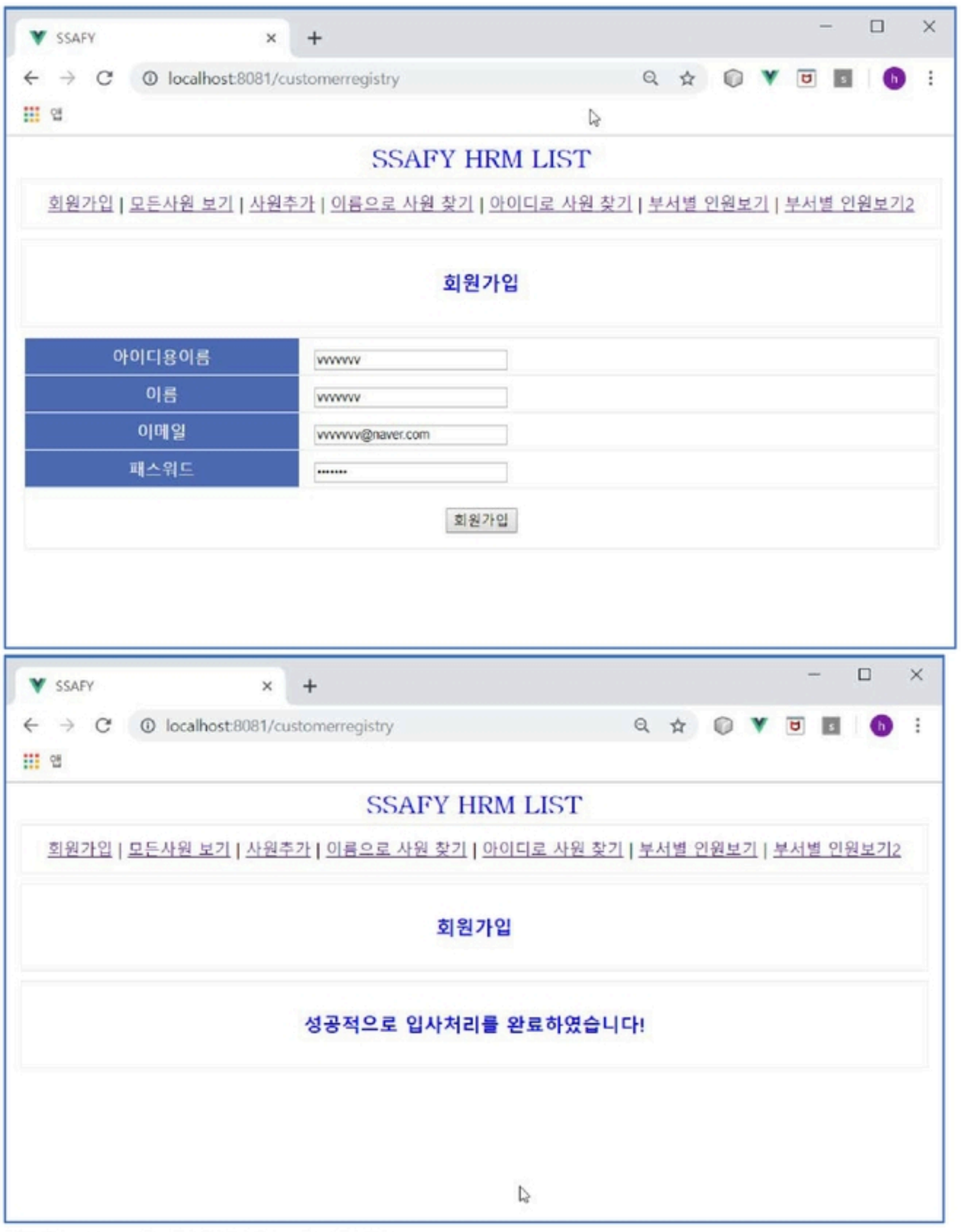
13. 로그인과 상태관리

13.1 회원가입

13.1.1 회원가입 구현

[hellovues6]를 복사한 후 복사본을 [hellovues7]로 수정하십시오 그리고 [package.json]와[package-lock.json] 의 이름을 hellovues7 으로 수정하십시오.

회원가입을 구현하겠습니다. 레스트풀은 회원가입을 위한 "/signup"서비스를 제공합니다. 뷰화면에서 아이디용이름, 이름, 이메일, 패스워드를 입력받아 JSON 으로 변경한 후 "/signup"서비스로 보내면 회원가입을 할 수 있습니다.



[그림 13.2] 회원가입 후 화면

[소스-CustomerRegitry.vue]

```
 <div v-if="!submitted">
<form action="" method="post" id="_frmForm" name="frmForm" @submit.prevent =
"registryMyself">
<table class="content_table">
<colgroup><col style="width:30%;" />
<col style="width:70%;" /></colgroup>
<tr><th>아이디용이름</th>
<td><input data-msg="이름" type="text" name="username" id="_username" v-model =
"customer.username" style="width:30%"/></td></tr>
<tr><th>이름</th>
<td><input data-msg="이름" type="text" name="cname" id="_cname"   v-model =
"customer.cname" style="width:30%"/></td></tr>
<tr><th>이메일</th>
<td><input data-msg="이메일" type="email" name="email"   id="_email" size="20" v-model =
"customer.email" style="width:30%"/></td></tr>
<tr><th>패스워드</th>
<td><input data-msg="패스워드" type="password" name="password"   id="_password" size =
"30" v-model="customer.password" style="width:30%"/></td></tr>
<tr><td colspan="2" style="height:50px; text-align:center;">
<button type="submit" name="button">회원가입</button></td></tr>
</table></form></div>
<div class='search_box' v-else>
<h3><span style='color:blue;'>성공적으로 입사처리를 완료하였습니다!</span></h3>
<button type="button" @click="newCustomer()">회원가입</button>
</div>
</div>
</template>
```

회원가입 submit 버튼을 클릭하면 registryMyself 메서드를 호출하여 회원가입을 합니다.

registryMyself 메서드에서 { username: this.customer.username, name: this.customer.cname, email: this.customer.email, password: this.customer.password

}형식으로 회원정보를 서버로 보내서 가입을 합니다.

[소스-CustomerRegitry.vue]

```
registryMyself() {
        if(this.customer.username==''){ alert('아이디용이름 입력하세요.'); return ;}
        if(this.customer.cname==''){ alert('이름을 입력하세요.'); return ;}
        if(this.customer.email==''){ alert('이메일을 입력하세요.'); return ;}
        if(this.customer.password==''){ alert('패스워드를 입력하세요.'); return ;}
        http2.post('/signup', {
                username: this.customer.username,
                name: this.customer.cname,
                email: this.customer.email,
                password: this.customer.password
        }).then(response => {
                console.log(response.data);
                if (response.data.success==true ||response.data.success=='true') {
                        alert("성공적으로 가입 하였습니다. 로그인 하십시오.");
                        this.visablelogin=true;
                        this.submitted = true;
                }else{
                        alert("가입을 하지 못했습니다.");
                        this.visablelogin=false;
                        this.submitted = false;
                        this.newCustomer();
                }
        }).catch(error => {
                        alert("가입을 하지 못했습니다.");
                        this.visablelogin=false;
                        this.submitted = false;
                        this.newCustomer();
        });
},
newCustomer() {
        this.customer={
                        username:'',
                        cname:'',
                        email:'',
```

```
                                    password:''
            };
            this.submitted = false;
}
```

form 태그 내의 회원정보를 입력 받고, “/sinup”을 호출하여 회원가입 합니다.

tp-commons2.js]

```
import axios from "axios";
export default axios.create({
   baseURL: "http://localhost:8397/api/auth",
   headers: {
      "Content-type": "application/json"
   }
});
```

axios 비동기를 편리하게 사용하기 위해 [http-commons2.js]를 만듭니다. 회원가입, 로그인 서비스는 ‘/api/auth’가 기본경로입니다.

[router.js]에 CustomerRegistry.vue 를 등록합니다.

[소스-router.js]

```
{
            path: '/customerregistry',
            name: 'customerregistry',
            component: CustomerRegistry
}
```

CustomerRegistry.vue 호출할 수 있도록 [App.vue]에 ‘/customerregistry’추가합니다.

[소스-App.vue]

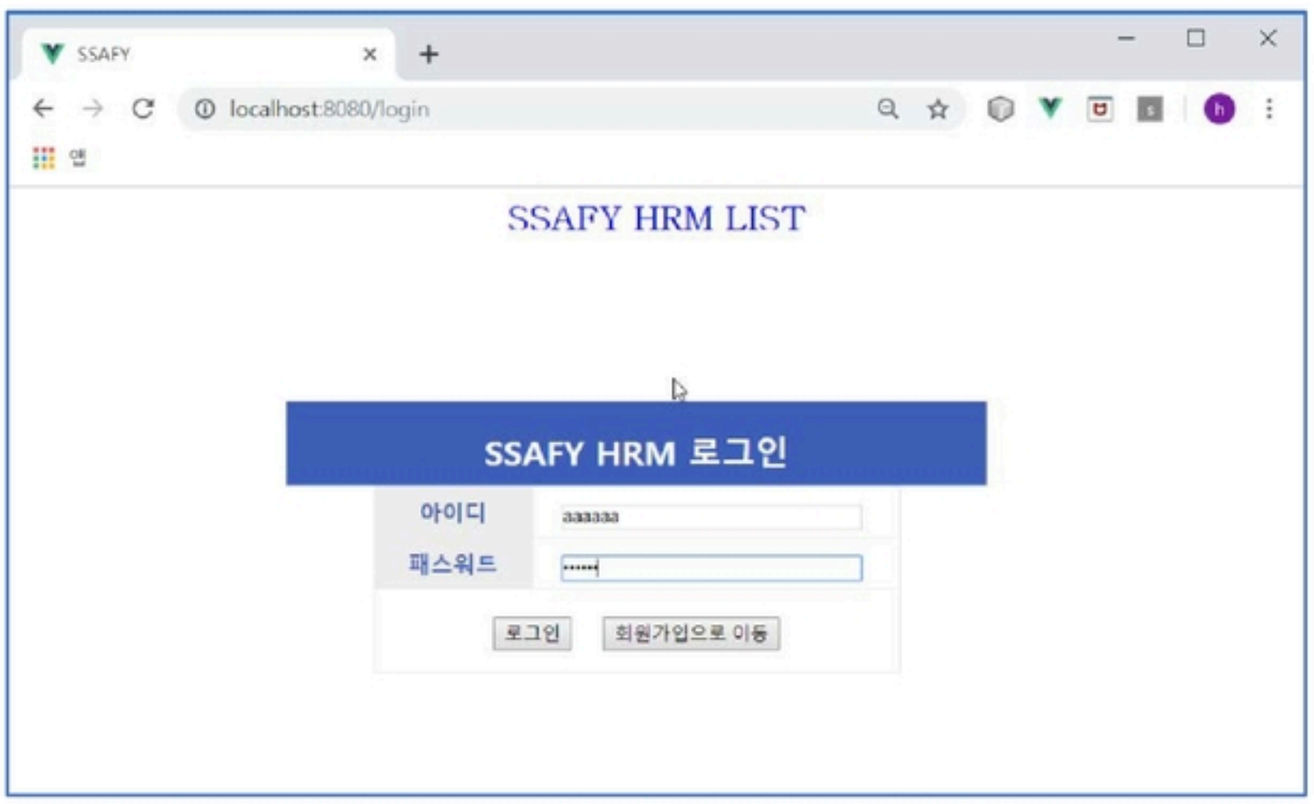
```
<router-link class="btn btn-primary" to="/customerregistry">회원가입</router-link> |
```

13.2 로그인

13.2.1 로그인

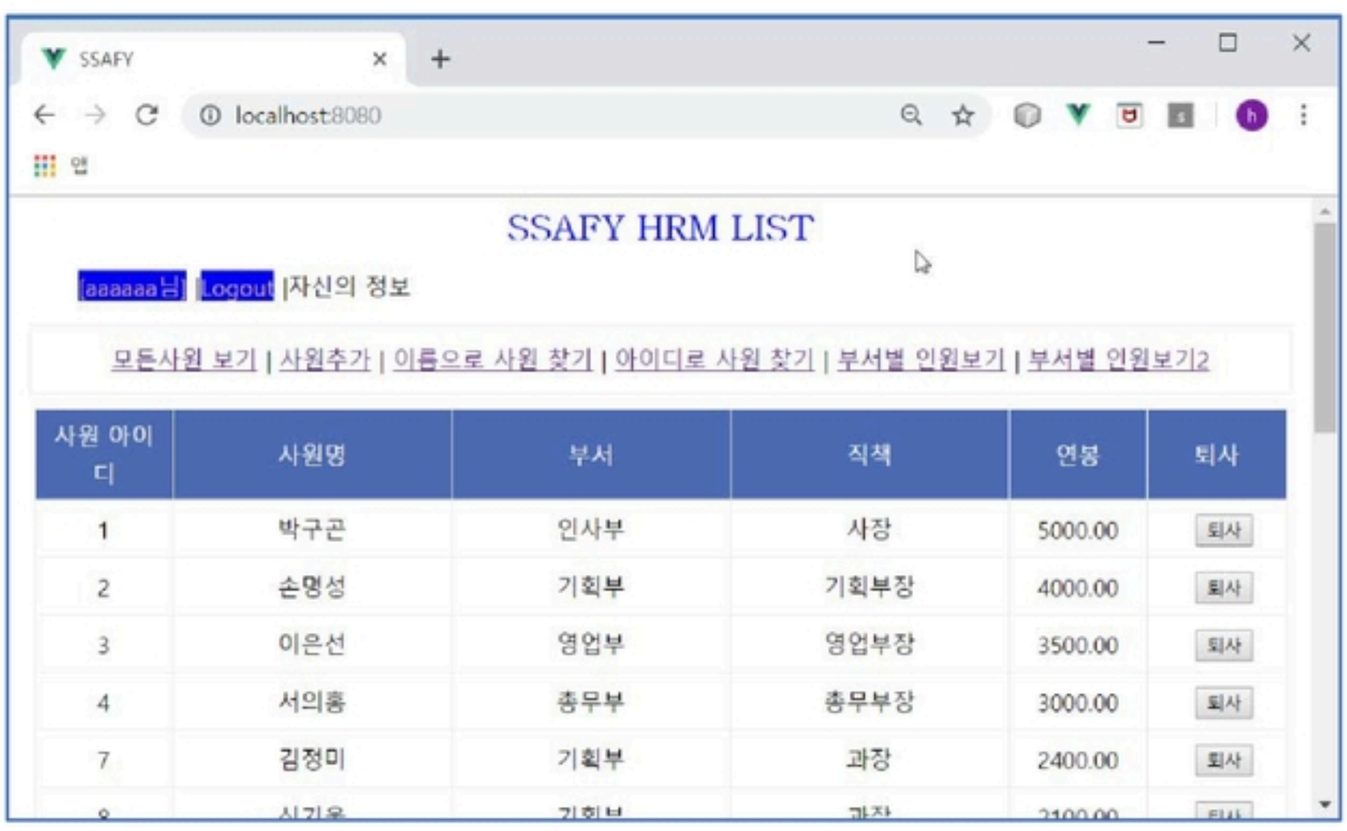
[hellovues7]를 복사한 후 복사본을 [hellovues8]로 수정하십시오 그리고 [package.json]와[package-lock.json] 의 이름을 hellovues8 로 수정하십시오.

로그인 과정입니다. 레스트 풀은 로그인을 위한 "/signin"서비스를 제공합니다. 아이디용 회원이름과 패스워드를 입력하면 로그인 할 수 있습니다.



로그인 한 상태에서만 HRM 정보를 사용할 수 있습니다. 로그아웃 버튼을 선택하면 로그아웃 됩니다.

[그림 13.3] 로그인 화면



[그림 13.4] 로그인 후 화면

13.2.2 프라미스(Promise)

비동기는 주로 콜백을 사용합니다. 콜백의 단점을 보완하며 비동기 처리 시점을 명확하게 표현하기 위해 ES6 에서 프로미스(Promise)를 도입하였습니다.

[소스-es6/index01.html]

```
<!DOCTYPE html>
<html><head>
  <meta charset="utf-8"><title>SSAFY</title>
</head>
<body>
<script type="text/javascript">
//프로미스 생성
const authorize = function(b){
  return new Promise(function(resolve,reject){
    if(b){
      resolve("인증됨");
    }else{
      reject("인증안됨");
    }
  });
}
//프로미스 실행
authorize(true)
 .then(function(result){
        console.log(result); // result → 인증됨
 })
 .catch(function(err){
        console.log(err);// err →  인증안됨
 });
</script>
</body>
</html>
```

authorize(b) , b 에 따라 b 가 true 면 resolve function(result)가 호출되고 , b 가 false 면 reject function(err)가 호출됩니다.

[소스-es6/index02.html]

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>SSAFY</title>
</head>
<body>
<script type="text/javascript">
//프로미스 생성
const authorize = (b)=>{
  return new Promise((resolve,reject)=>{
    if(b){
      resolve("인증됨");
    }
    else{
      reject("인증안됨");
    }
  });
}
//프로미스 실행
authorize(true).then((result)=>{
  console.log(result);//인증됨
}).catch((err)=>{
  console.log(err);//인증됨
})</script>
</body>
</html>
```

'=>'는 자바의 람다와 동일한 표한 방법으로 화살표 함수(arrow function)라고도 부릅니다. const myfunction=function(b){ } 에서 function 을 삭제하면 (b){}이 되는데, function 을 삭제했다는 의미로 '=>'입력하여 const myfunction =(b)=>{ }으로 표현합니다.

'const a= function(e){ }'형식을 'const a=(e)=>{}' 간략하게 사용할 수 있습니다.

소스-api0.js]

```
import axios from 'axios'
  const apiCall = function({url, method, ...args}) {
        return new Promise(function(resolve, reject) {
            axios.post('http://localhost:8397/api/auth/signin',
                    args["data"]).then(({data}) => {
              resolve(data)
            }).catch((error) => {
              reject(error)
            });
  }

  export default apiCall
```

apiCall({url: 'auth', method: 'POST',

data: {usernameOrEmail: "cccccc", password: "cccccc"}

})로 호출되면 function({url, method, …args})이 실행되므로 args.data 또는 args["data"]가 {usernameOrEmail: "cccccc", password: "cccccc"}이 됩니다.
axios 는 레스트풀 서비스를 사용하여 해당 사용자명과 패스워드의 사용자 인증(로그인)을 받게 됩니다. 인증에 성공하면

{accessToken:"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiI0IiwiaWF0IjoxNTcwM…XYOy7KZva2LCwHnDixyQHAIX1E6K3EzmA9WSMNJklkIVthjyQ",tokenType: "Bearer"} 형식의 인증 토큰을 받습니다. 그리고 이것을 Vuex 저장소 store 에 저장하고, localStorage 에도 저장합니다. 로그아웃을 하면 store 의 토큰을 제거하고 localStorage 의 토큰도 제거합니다. [api.js]와 같이 =>를 사용하면 편리합니다.

[소스-api.js]

```
import axios from 'axios'
  const apiCall = ({url, method, ...args}) => new Promise((resolve, reject) => {
```

```
    axios.post('http://localhost:8397/api/auth/signin',
                    args["data"]).then(({data}) => {
      resolve(data)
    }).catch((error) => {
      reject(error)
    })
  })
  export default apiCall
```

13.2.3 Vuex

Vuex 는 store 라는 저장소 개념을 사용합니다. 저장소는 상태를 저장하는 컨테이너입니다. Vuex store 는 반응형으로 Vue 컴포넌트의 상태를 검색할 때 저장소의 상태가 변경되면 자동으로 변경됩니다. 그러나 저장소의 상태를 직접 변경할 수 없고, 명시적인 커밋을 통해 상태를 변경하는 변이(mutaion)를 사용합니다.

[소스-vuex/index01.html]

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"><title>vuex</title>
  <script src="https://unpkg.com/vue"></script>
  <script src="https://unpkg.com/vuex"></script>
</head><body><div id="app"><p>{{ cnt }}</p>
<p><button @click="increase">+</button><button @click="decrease">-</button></p>
</div>
<script type="text/javascript">
const store = new Vuex.Store({
  state: { cnt: 0 },
  mutations: {
    increment: state => state.cnt++,      decrement: state => state.cnt--
  }});
new Vue({
  el: '#app',
  computed: {   cnt () {return store.state.cnt}   },
  methods: {
      increase () {
        store.commit('increment')
      },
      decrease () {
         store.commit('decrement')
     }
  }
});
</script></body></html>
```

저장소 store 에 있는 상태(state)에 cnt 를 저장합니다. 버튼을 클릭하여 increase() 메서드를 호출하면 store 저장소를 이용하여 상태를 변경하기 위해 커밋이 되고 increment 변이(mutations)가 반영되어 state.cnt 가 1 씩 증가합니다. 같은 원리로 버튼을 클릭하여 decrease() 메서드를 호출하면 store 저장소를 이용하여 상태를 변경하기 위해 커밋이 되고 decrement 변이(mutations)가 반영되어 state.cnt 가 1 씩 감소합니다.

[소스-vuex/index02.html]

```
<div id="app">
  <p>{{ cnt }}</p>
  <p>
    <button @click="increase">+</button><button @click="decrease">-</button>
  </p>
</div>
<script type="text/javascript">
const INCREMENT = 'increment'
const DECREMENT = 'decrement'
const store = new Vuex.Store({
  state: {cnt: 0},
  mutations: {
        [INCREMENT]: state => state.cnt++,
        [DECREMENT]: state => state.cnt--
  }});
new Vue({
  el: '#app',
  computed: {cnt () {return store.state.cnt} },
  methods: {
    increase () {
      store.commit(INCREMENT)
    },
    decrease () {
      store.commit(DECREMENT)
    }
  }});

</script>
```

Store 에 저장되는 상태가 많다면 관리하기 힘들 것입니다. 이름을 혼동하지 않고 사용하기 위해 const INCREMENT = 'increment'과 같이 이름(상수변수)을 등록해서 커밋과 변이에서 사용하면 편리합니다.

actions 에서는 사용자 인증과 사용자 정보를 요청하기 위해 상수 변수를 등록하여 사용합니다. 사용자 인증을 요청할 때는 AUTH_REQUEST, 인증에 성공하면 AUTH_SUCCESS 를 사용합니다.

[소스-actions/auth.js]

```
export const AUTH_REQUEST = 'AUTH_REQUEST'
export const AUTH_SUCCESS = 'AUTH_SUCCESS'
export const AUTH_ERROR = 'AUTH_ERROR'
export const AUTH_LOGOUT = 'AUTH_LOGOUT'
```

사용자 정보를 요청할 때 USER_REQUEST 를, 요청에 성공하면 USER_SUCCESS 을 사용합니다.

[소스-actions/user.js]

```
export const USER_REQUEST = 'USER_REQUEST'
export const USER_SUCCESS = 'USER_SUCCESS'
export const USER_ERROR = 'USER_ERROR'
```

modules는 state, getters, actions, mutations를 선언합니다.

13.2.3 Vuex 구조

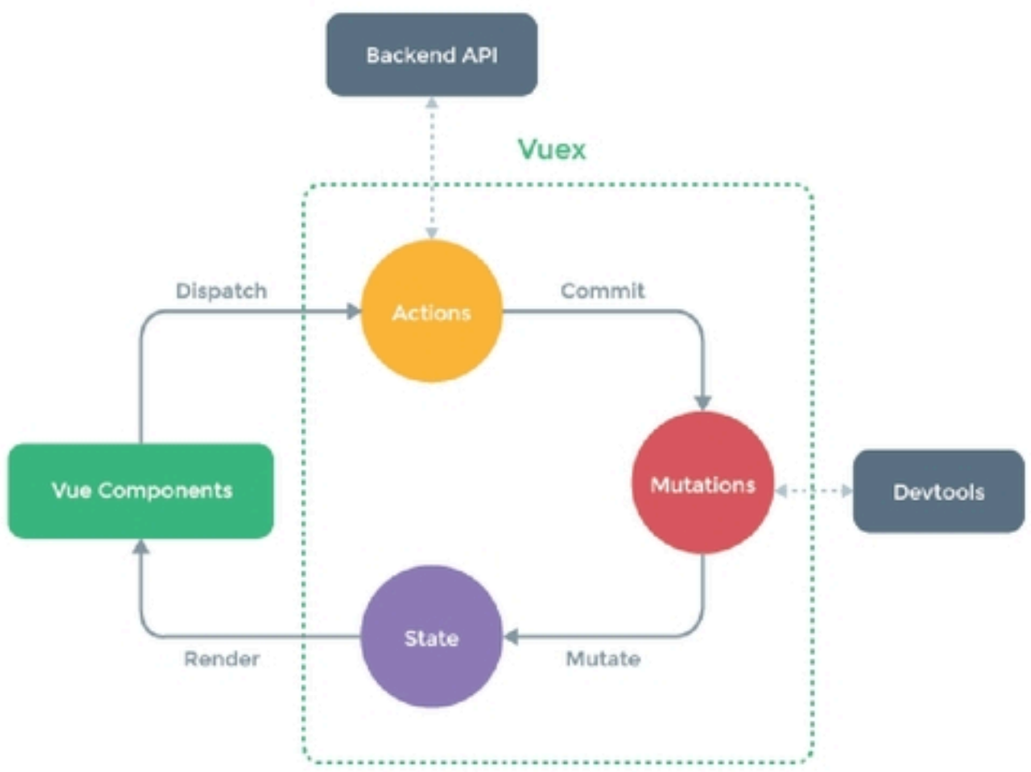
아이디와 패스워드를 입력하여 로그인을 하겠습니다. 아이디와 패스워드를 입력하고 로그인 버튼을 클릭하면 메서드에서 Vuex store를 이용하여 this.$store.dispatch(AUTH_REQUEST, {아이디, 패스워드}) 형식으로 Actions에 디스패치합니다.

Actions에서는 [AUTH_REQUEST]에 해당하는 커밋을 실행시킵니다. 이 커밋은 Mutations로 하여금 현재의 상태 state.status가 'loading'중으로 상태를 변경시키도록 합니다.

그리고 다시 Actions에서 레스트풀 서비스를 호출하여 아이디와 패스워드에 해당하는 사용자의 인증을 받습니다. 인증되면 인증 토큰을 받습니다.

인증되면 [AUTH_SUCCESS]해당하는 커밋을 실행시킵니다. 이 커밋은 Mutations로 하여금 현재의 상태 state.status가 인증되어 'success' 로 상태를 변경시키도록 하고, state.token에 인증토큰 값도 저장합니다.

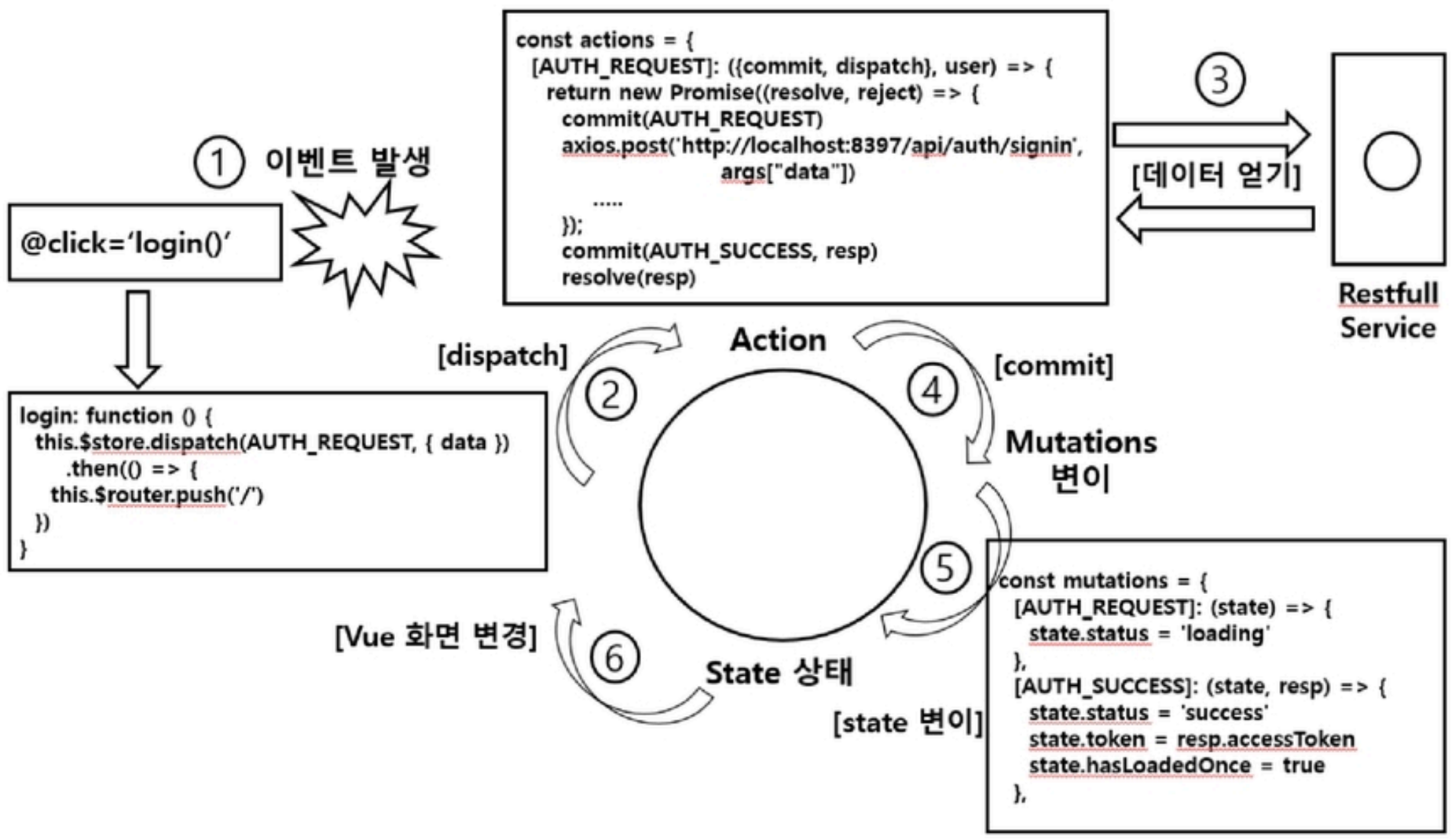
resolve(data)를 통해서 Vue 화면에 결과를 반영합니다. 로그인 화면에서 사원정보 리스트 화면으로 이동합니다. 직접 상태를 변경하는 것이 아니라 등록한 행위(Actions)로 디스패치하면, 해당 Actions는 커밋을 실행시켜 Mutations로 하여금 상태(State)를 변경시키는 것입니다.



[그림 13.1] Vuex 의 Action, Mutation, State 의 관계 1
(참고 https://vuex.vuejs.org/kr/)

로그인 과정을 정리한 그림입니다.

1) (1) 아이디와 패스워드를 클릭하여 로그인 이벤트를 발생시킵니다.
2) (2) login()메서드는 this.$store.dispatch(AUTH_REQUEST, {아이디, 패스워드})형식으로 Actions 에 디스패치합니다.
3) (4) Actions 에서는 [AUTH_REQUEST]에 해당하는 커밋을 실행시킵니다.
4) (5) 이 커밋은 Mutations 로 하여금 현재의 상태 state.status 가 'loading'중으로 상태를 변경시키도록 합니다.
5) (3) Actions 에서 레스트풀 서비스를 호출하여 아이디와 패스워드에 해당하는 사용자의 인증을 받습니다.
6) (4) 인증되면 인증 토큰을 받습니다. 인증되면 [AUTH_SUCCESS]해당하는 커밋을 실행시킵니다.
7) (5) 이 커밋은 Mutations 로 하여금 현재의 상태 state.status 가 인증되어 'success' 로 상태를 변경시키도록 하고, state.token 에 인증토큰 값도 저장합니다.
8) resolve(resp)는 상태에서 Vue 화면으로 이동하여 this.$router.push('/')가 실행됩니다. 사원 목록 화면으로 라우팅됩니다.



[그림 13.1] Vuex 의 Action, Mutation, State 의 관계 2

[소스- store/modules/auth.js]

```
import { AUTH_REQUEST, AUTH_ERROR, AUTH_SUCCESS, AUTH_LOGOUT } from
'../actions/auth'
import { USER_REQUEST } from '../actions/user'
import apiCall from '../../utils/api'
const state = { token: localStorage.getItem('user-token') || '', status: '', hasLoadedOnce: false }
const getters = {
  isAuthenticated: state => !!state.token,
  authStatus: state => state.status
}
const actions = {
  [AUTH_REQUEST]: ({commit, dispatch}, user) => {
    return new Promise((resolve, reject) => {
      commit(AUTH_REQUEST)
      apiCall({url: 'auth', data: user, method: 'POST'})
      .then(resp => {
        localStorage.setItem('user-token', resp.accessToken)
        if (resp.accessToken!=null || resp.accessToken !='') {
            alert("안녕하세요  "+user.usernameOrEmail);
        }
        commit(AUTH_SUCCESS,resp)
        dispatch(USER_REQUEST,user)
        resolve(resp)
      })
      .catch(err => {
        alert("로그인에 실패하였습니다.");
        commit(AUTH_ERROR, err)
        localStorage.removeItem('user-token')
        reject(err)
      })
    })
  },
```

[소스- store/modules/auth.js]

```
[AUTH_LOGOUT]: ({commit, dispatch}) => {
    return new Promise((resolve, reject) => {
      commit(AUTH_LOGOUT)
      localStorage.removeItem('user-token')
      resolve()
    })
}
}
const mutations = {
  [AUTH_REQUEST]: (state) => {
    state.status = 'loading'
  },
  [AUTH_SUCCESS]: (state, resp) => {
    state.status = 'success'
    state.token = resp.accessToken
    state.hasLoadedOnce = true
  },
  [AUTH_ERROR]: (state) => {
    state.status = 'error'
    state.hasLoadedOnce = true
  },
  [AUTH_LOGOUT]: (state) => {
    state.token = ''
  }
}
export default {
  state,
  getters,
  actions,
  mutations,
}
```

[소스- store/modules/user.js]

```
import { USER_REQUEST, USER_ERROR, USER_SUCCESS } from '../actions/user'
import apiCall2 from '../../utils/api2'
import Vue from 'vue'
import { AUTH_LOGOUT } from '../actions/auth'
const state = { status: '', profile: {} }
const getters = {
  getProfile: state => state.profile,
  isProfileLoaded: state => !!state.profile.name
}
const actions = {
  [USER_REQUEST]: ({commit, dispatch},args) => {
    apiCall2( args.usernameOrEmail)
      .then(resp => {
        commit(USER_SUCCESS, resp)
      })
      .catch(resp => {
        commit(USER_ERROR)
        dispatch(AUTH_LOGOUT)
      })

  },
}
```

[소스-store/modules/user.js]

```
const mutations = {
  [USER_REQUEST]: (state,resp) => {
    state.status = 'loading'
  },
  [USER_SUCCESS]: (state, resp) => {
    state.status = 'success'
    Vue.set(state, 'profile', resp)
  },
  [USER_ERROR]: (state) => {
    state.status = 'error'
  },
  [AUTH_LOGOUT]: (state) => {
    state.profile = {}
  }
}

export default {
  state,
  getters,
  actions,
  mutations,
}
```

Vuex 의 Store 에 user 와 auth 의 state, getters, actions, mutations 를 선언합니다.

[소스-store/index.js]

```
import Vue from 'vue'
import Vuex from 'vuex'
import user from './modules/user'
import auth from './modules/auth'
Vue.use(Vuex)

const debug = process.env.NODE_ENV !== 'production'

export default new Vuex.Store({
  modules: {
    user,
    auth,
  },
  strict: debug,
})
```

상태 값을 반영해서 사용할 수 있는 isAuthenticated 를 이용합니다. 로그인 되었다면 인증되어 isAuthenticated 가 true 가 되어 로그아웃 버튼이 보이고 사용자 정보가 보입니다. 그리고 여러 라우팅 경로가 보입니다. 로그인을 하지 않으면 반대가 되어 보이지 않습니다.

```
const getters = {
  isAuthenticated: state => !!state.token,
  authStatus: state => state.status
}
```

isAuthenticated 는 store 에 저장된 인증토큰이 있는가 없는가(!!)에 따라 자동으로 true, false 가 됩니다. 인증되었는가 안되었는가만 사용하기 때문에 getters 로 사용하면 편리합니다.
같은 원리로 user 의 isProfileLoaded 는 사용자 기본 정보가 profile 에 저장되어 있는가 없는가(!!)입니다. 사용자 정보가 레스트풀에서 얻어서 store 에 저장했다면 true 가 됩니다.

```
const getters = {
  getProfile: state => state.profile,
  isProfileLoaded: state => !!state.profile.name
```

[소스-App.vue]

```
<template>
    <div id="app">
    <div class='headtitle'><h2>SSAFY HRM LIST</h2></div>

<div id="top_menu_wrap">
        <div id="_top_menu">
                <ul class="navi">
      <li v-if="isAuthenticated && isProfileLoaded">
        <span class="logout">[{{name}}님]</span> |
      </li>
      <li v-if="isAuthenticated" @click="logout">
        <span class="logout">Logout</span> |
      </li>
      <li v-if="isAuthenticated" @click="aboutme">
         <span class="aboutme">자신의 정보</span>
      </li>
      <li v-if="!isAuthenticated">
      <!-- <router-link to="/login">로그인</router-link> -->
          <login/>
      </li>
                </ul>
        </div>
</div>
```

인증되었다면 isAuthenticated 가 true 가 되어 v-if 부분이 보입니다. 각 화면의 경로가 보입니다.
경로를 선택하면 라우터 뷰에 해당 화면이 출력됩니다.

[소스-App.vue]

```
<div class='search_box' v-if="isAuthenticated">
 <nav>
  <router-link class="btn btn-primary" to="/">모든사원 보기</router-link> |
  <router-link class="btn btn-primary" to="/add">사원추가</router-link> |
  <router-link class="btn btn-primary" to="/search">이름으로 사원 찾기</router-link> |
  <router-link class="btn btn-primary" to="/searchbyid">아이디로 사원 찾기</router-link> |
  <router-link class="btn btn-primary" to="/chartcustomer">부서별 인원보기</router-link> |
  <router-link class="btn btn-primary" to="/chartpiecustomer">부서별 인원보기2</router-link>
 </nav>
</div>
<div>
        <router-view v-if="isAuthenticated" />

</div>
    </div>

</template>

<script>
  import { mapGetters, mapState } from 'vuex'
  import Login from './components/Login'
  import {AUTH_LOGOUT} from './store/actions/auth'
```

긴 경로와 추적을 편리하게 하기 위해 computed 속성에 필요한 형태의 상태 값을 선언합니다.
자동으로 반영된 상태값을 편리하게 사용하도록 getMapper 를 선언합니다.

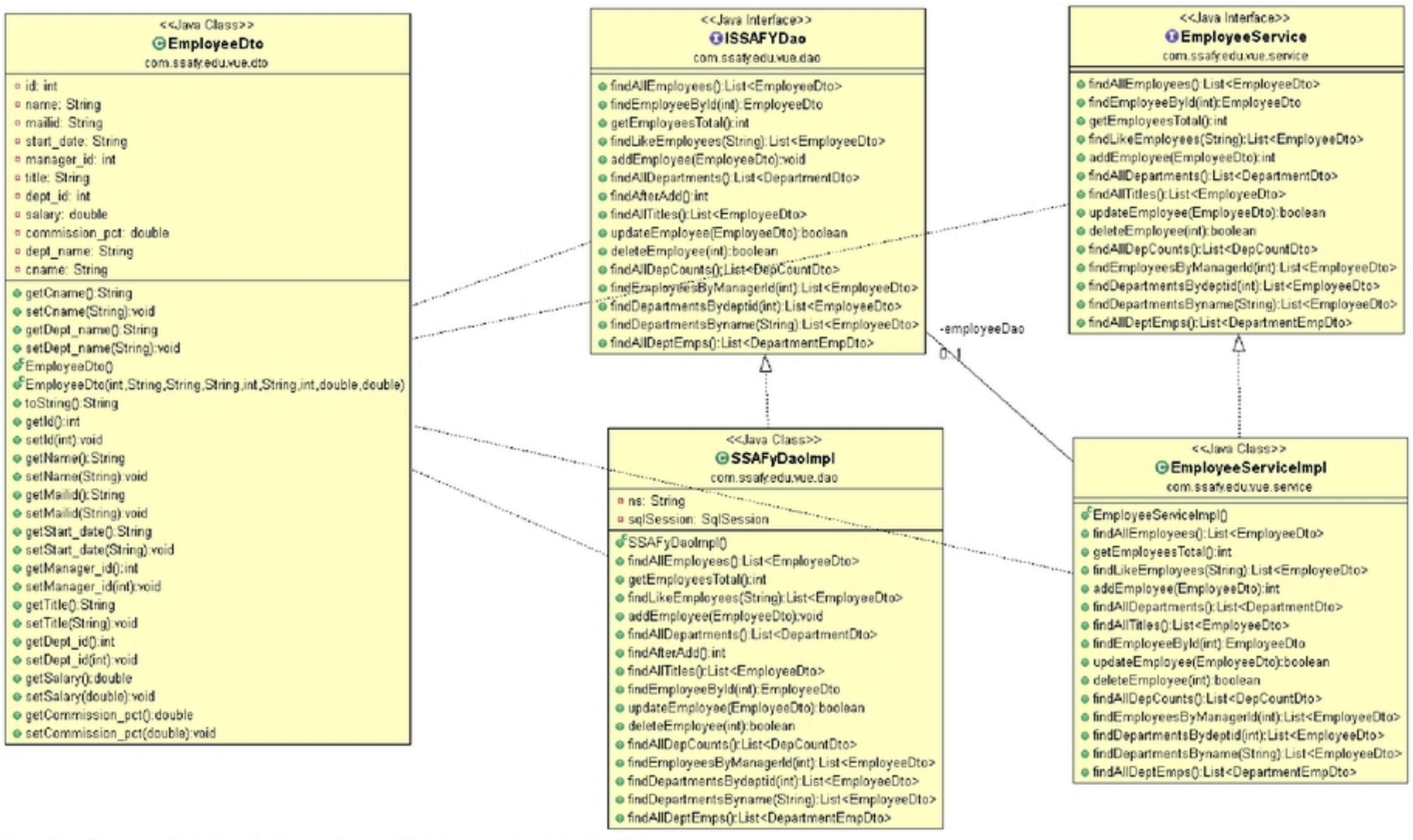
로그아웃을 클릭하면 this.$store.dispatch(AUTH_LOGOUT)처럼 AUTH_LOGOUT 로 디스패치 됩니다.
그러면 커밋이 실행되고 AUTH_LOGOUT 에 해당하는 상태로 변이됩니다.

다시 말하면 store 에 있는 인증토큰 값이 삭제되고, status 가 ''이 되며 사용자 profile 에 제거됩니다.
그 후 로그인하는 화면으로 라우팅됩니다.

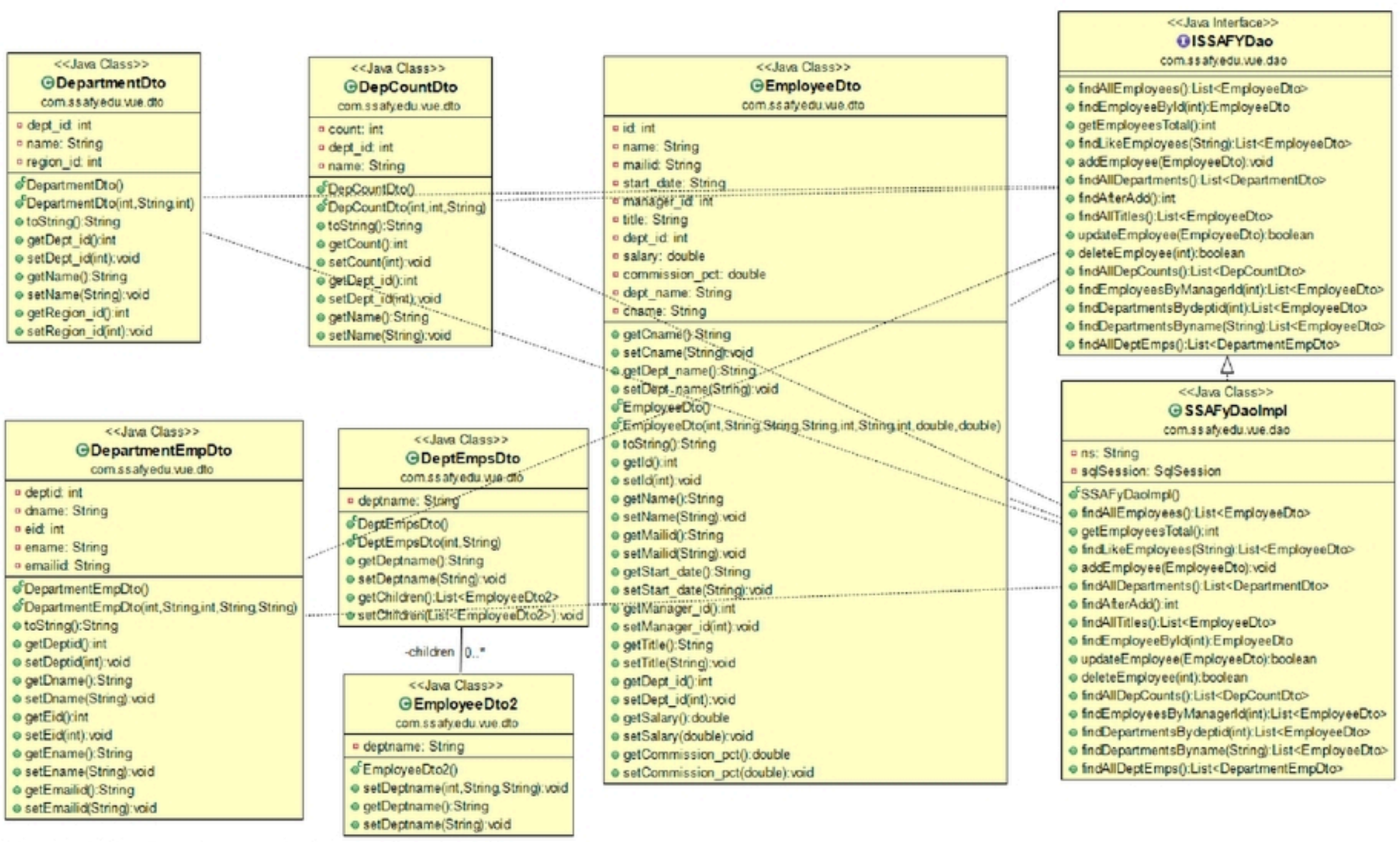
[소스-App.vue]

```
export default {
  name: "app",
  components: {
      Login
    },
    methods: {
      logout: function () {
        alert('로그아웃 합니다. 로그인 하십시오.')
        this.$store.dispatch(AUTH_LOGOUT).then(() => this.$router.push('/login'))
      }
  },
   aboutme: function () {
     alert('본인의 정보를 보입니다. 구현하세요.~~.')
     //this.$router.push('/')
  }
},
computed: {
  ...mapGetters(['isAuthenticated', 'isProfileLoaded','getProfile']),
  ...mapState({
     authLoading: state => state.auth.status === 'loading',
     name: state => `${state.user.profile.name}`,
  }),
     loading: function () {
       return this.authStatus === 'loading' && !this.isAuthenticated
     }
  }
};
</script>
```

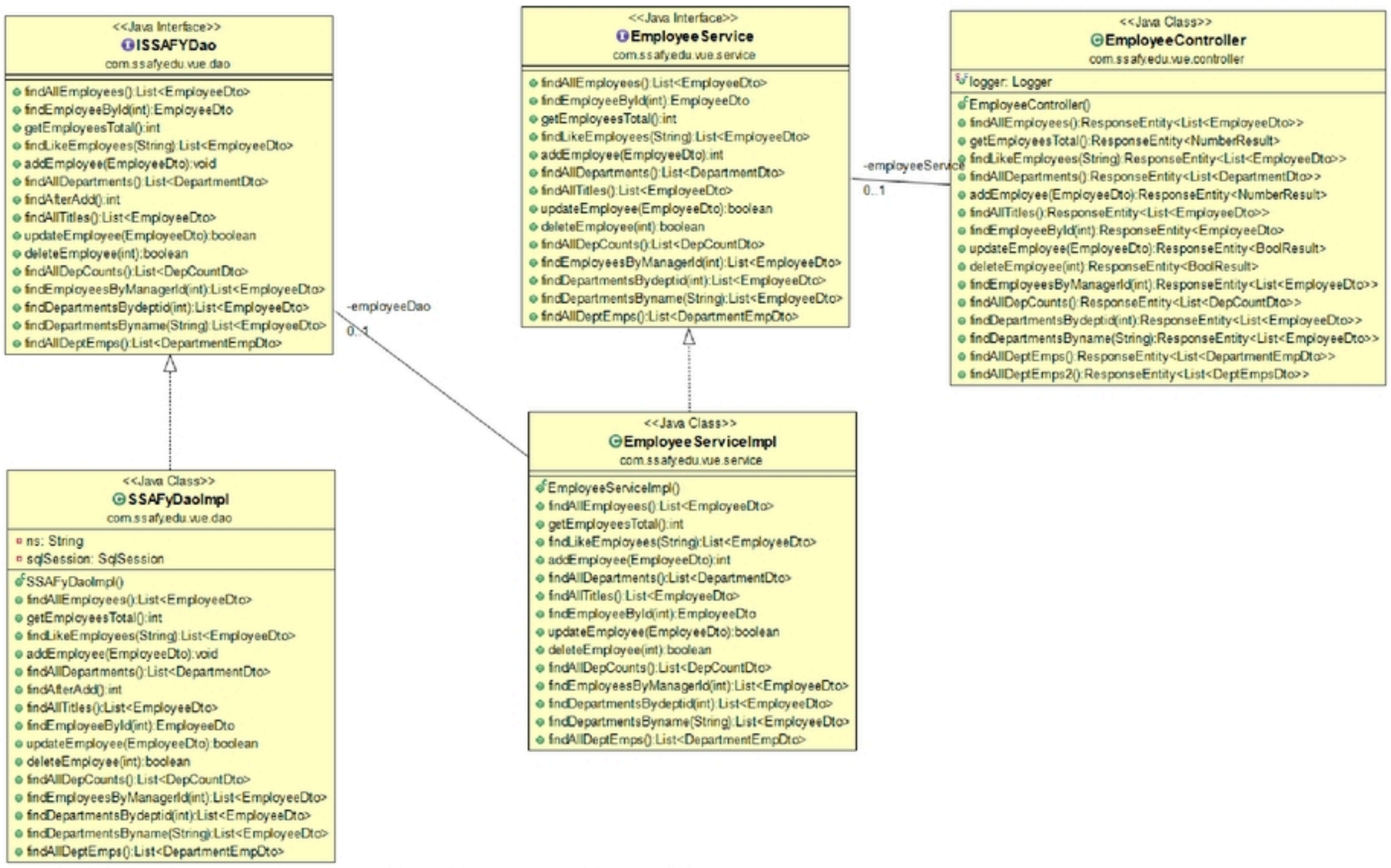
SSAFY HRM 관련 클래스 다이어그램



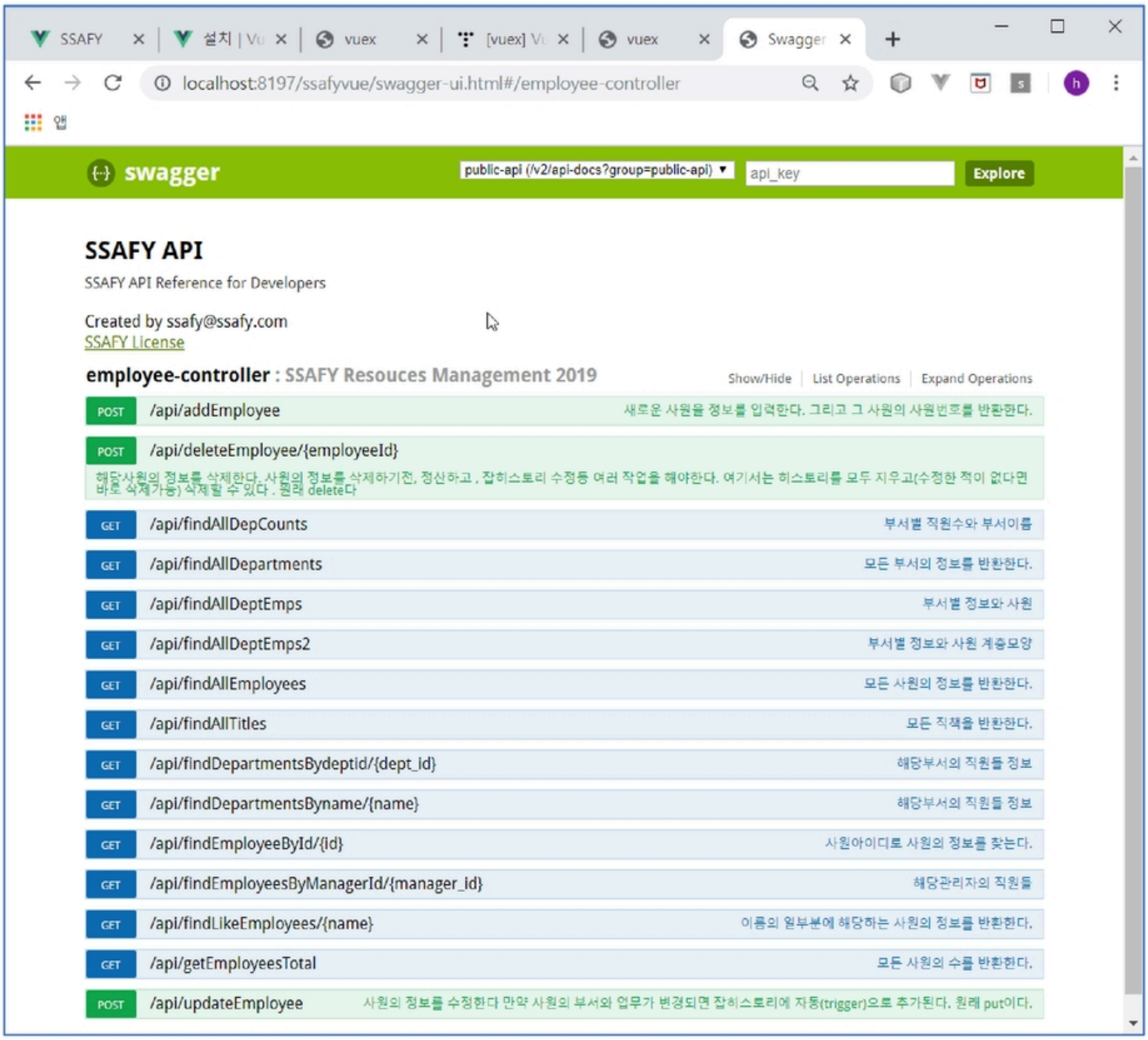
[그림-참고 1] DAO, Service 클래스 다이어그램



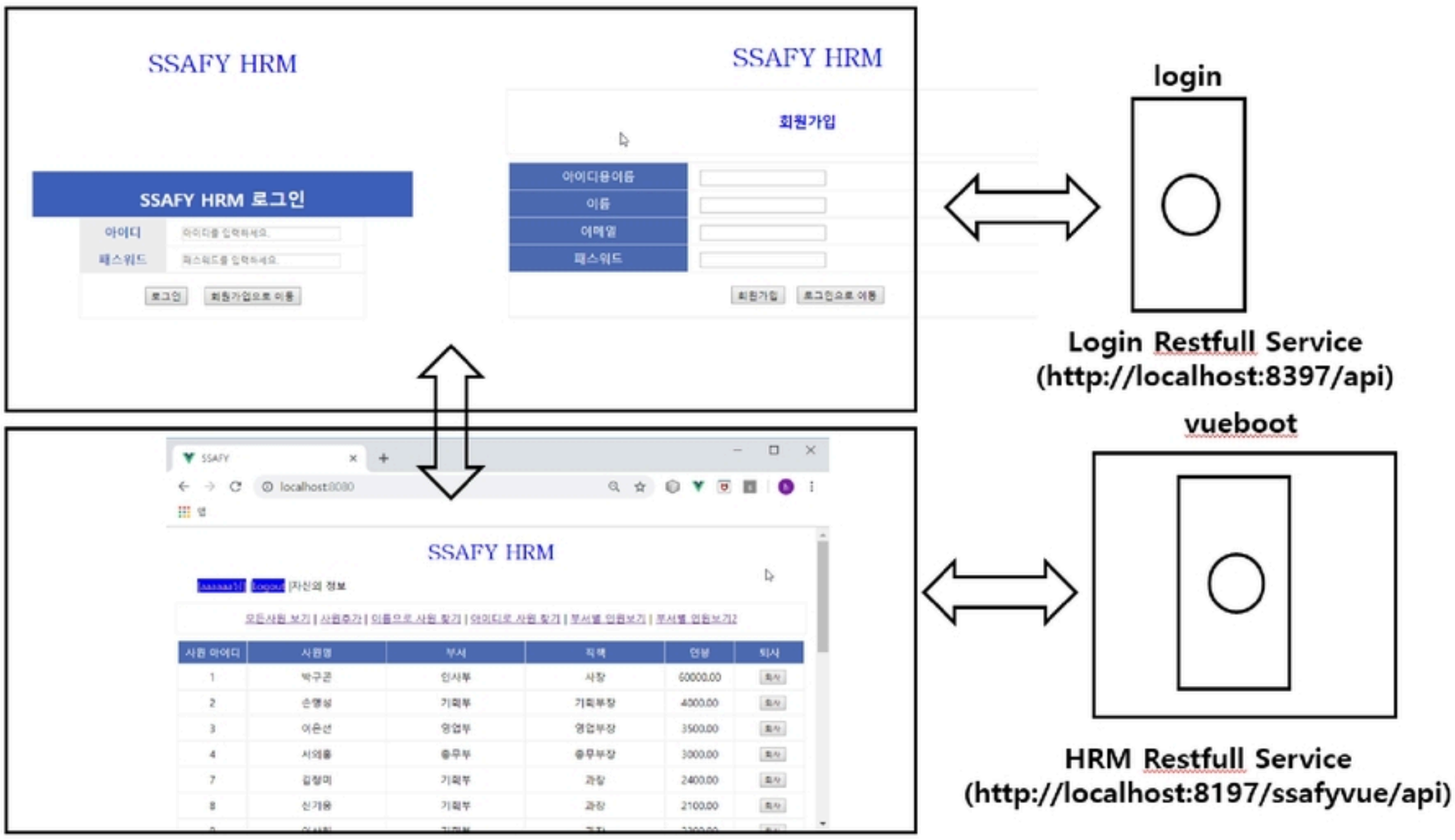
[그림-참고 2] DTO 클래스 다이어그램



[그림-참고 3] Model-Controller 클래스 다이어그램



[그림-참고 4] SWAGGER API



[그림-참고 5] 뷰와 부트 연동(레스트풀 이용)