todo 앱

2020-05-10

이승진

**학습목표**

계산속성

v-model modifier

todo 앱 구현

**목차**

[1. 계산속성 (computed property) 2](#_Toc40097488)

[1) 계산식 출력 2](#_Toc40097489)

[2) 메소드 리턴값 출력 4](#_Toc40097490)

[3) 계산속성 getter 구현 5](#_Toc40097491)

[4) 계산속성 getter setter 구현 9](#_Toc40097492)

[2. v-model modifier 11](#_Toc40097493)

[1) v-model.trim 11](#_Toc40097494)

[2) v-model.lazy 13](#_Toc40097495)

[3) v-model.number 14](#_Toc40097496)

[3. todo 앱1 16](#_Toc40097497)

[1) todo10.html 16](#_Toc40097498)

[2) todo11.html 17](#_Toc40097499)

[3) todo12.html 19](#_Toc40097500)

[4) todo1.css 21](#_Toc40097501)

[5) todo13.html 22](#_Toc40097502)

[6) todo14.html 24](#_Toc40097503)

[7) todo15.html 26](#_Toc40097504)

[8) todo16.html 28](#_Toc40097505)

[4. todo 앱2 30](#_Toc40097506)

[1) todo2.css 30](#_Toc40097507)

[2) todo20.html 31](#_Toc40097508)

[3) todo21.html 34](#_Toc40097509)

# 계산속성 (computed property)

## 계산식 출력

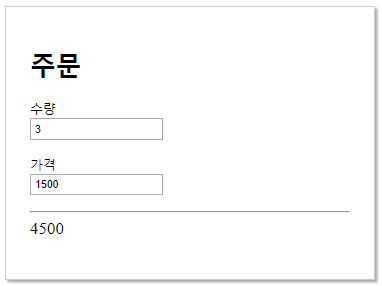
### computed0.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#app div { margin-bottom: 20px; }      label { display: block; }      input[type=text] { padding: 4px; }      div.total { font-size: 16pt; }    </style>  </head>  <body>    <div id="app">      <h1>주문</h1>      <div>        <label>수량</label>        <input type="text" v-model="quantity" />      </div>      <div>        <label>가격</label>        <input type="text" v-model="price" />      </div>      <hr />      <div class="total">        {{ quantity \* price }}      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          quantity: 1,          price: 0        }      })     </script>  </body>  </html> |

{{ quantity \* price }}

quantity 속성값과 price 속성값이 곱셈값이 출력된다.

두 속성 중에 하나가 변경되면, 곱셈값도 다시 계산되어 출력된다.



## 메소드 리턴값 출력

### computed1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#app div { margin-bottom: 20px; }      label { display: block; }      input[type=text] { padding: 4px; }      div.total { font-size: 16pt; }    </style>  </head>  <body>    <div id="app">      <h1>주문</h1>      <div>        <label>수량</label>        <input type="text" v-model="quantity" />      </div>      <div>        <label>가격</label>        <input type="text" v-model="price" />      </div>      <hr />      <div class="total">        {{ total() }}      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          quantity: 1,          price: 0        },        methods: {          total() {            return this.quantity \*this.price;          }        }      })     </script>  </body>  </html> |

{{ total() }}

total() 메소드의 리턴값이 출력된다.

출력해야할 계산식이 복잡하다면, 위와 같이 메소드의 형태로 구현하는 것이 바람직할 것이다.

화면을 다시 그려야할 때마다 메소드가 호출되고, 그 리턴값이 출력된다.

## 계산속성 getter 구현

### computed2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#app div { margin-bottom: 20px; }      label { display: block; }      input[type=text] { padding: 4px; }      div.total { font-size: 16pt; }    </style>  </head>  <body>    <div id="app">      <h1>주문</h1>      <div>        <label>수량</label>        <input type="text" v-model="quantity" />      </div>      <div>        <label>가격</label>        <input type="text" v-model="price" />      </div>      <hr />      <div class="total">        {{ total }}      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          quantity: 1,          price: 0        },        computed:  {          total() {            return this.quantity \* this.price;          }        }      })     </script>  </body>  </html> |

### 계산 속성 (computed property)

computed:  {

  total() {

    return this.quantity \* this.price;

  }

}

위 코드의 구조는 아래의 코드와 동일하다.

computed:  {

  total: function() {

    return this.quantity \* this.price;

  }

}

Vue 앱에 total 계산 속성을 정의한다. (computed property)

계산 속성이란, 구현 형태는 메소드인데, 사용법은 속성과 같은 것을 말한다.

total 계산 속성은 값을 리턴하는 getter 메소드만 구현되었기 때문에,

읽기 전용 속성 형태로 사용할 수 있고, 속성에 값을 대입할 수는 없다.

{{ total }}

total 계산 속성의 값이 출력된다.

### 메소드 vs 계산 속성

메소드의 리턴값을 출력하는 형태로 구현하면, 화면을 다시 그릴 때 마다 메소드가 다시 호출된다.

계산 속성의 값을 출력하는 형태로 구현하면, getter 메소드의 리턴값이 캐시에 저장된 후 화면에 출력된다.

그리고 계산 속성의 값을 다시 계산할 필요가 없을 때는, 캐시에 저장된 값을 사용하여 화면을 그린다.

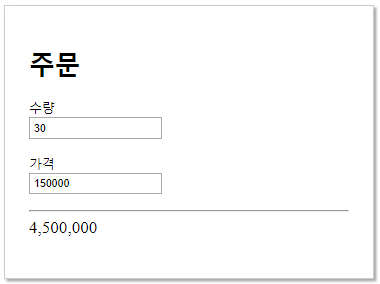
따라서 계산식의 값을 화면에 출력해야 하는 경우에, 그 계산식을 메소드로 구현하는 것보다,

계산 속성으로 구현하는 것이 바람직하다.

물론 계산식이 매우 간단하다면, {{ quantity \* price }} 형태로 구현하는 것도 좋다.

### computed3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#app div { margin-bottom: 20px; }      label { display: block; }      input[type=text] { padding: 4px; }      div.total { font-size: 16pt; }    </style>  </head>  <body>    <div id="app">      <h1>주문</h1>      <div>        <label>수량</label>        <input type="text" v-model="quantity" />      </div>      <div>        <label>가격</label>        <input type="text" v-model="price" />      </div>      <hr />      <div class="total">        {{ total }}      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          quantity: 1,          price: 0        },        computed:  {          total() {            let s = String(this.quantity \* this.price);            for (let i = 3; s.length > i; i += 4)              s = s.slice(0, -i) + "," + s.slice(-i);            return s;          }        }      })     </script>  </body>  </html> |



4,500,000 형태로 total 계산 속성값을 출력하기 위한 코드를 추가하였다.

let s = String(this.quantity \* this.price);

this.quantity \* this.price 계산식의 값을 문자열로 변환하여, 지역 변수 s에 대입한다.

s.slice(0, -i)

s 문자열의 인덱스 0 위치부터, 오른쪽 끝에서 i 번째 위치까지의 부분 문자열을 리턴한다.

s.slice(-i)

s 문자열의 오른쪽 끝에서 i 번째 위치부터, 끝까지의 부분 문자열을 리턴한다.

s = s.slice(0, -i) + "," + s.slice(-i);

s 문자열의 오른쪽 끝에서 i 번째 위치에, "," 문자열이 삽입된 새 문자열을 생성하여, 변수 s에 대입한다.

for (let i = 3; s.length > i; i += 4)

   s = s.slice(0, -i) + "," + s.slice(-i);

s 문자열의 오른쪽 끝에서 3, 7, 11, 15,.. 번째 위치에 "," 문자열을 삽입한다.

12,000,000,000

**11** 10 9 8 **7** 6 5 4 **3** 2 1 0

## 계산속성 getter setter 구현

### computed4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      div#app div { margin-bottom: 20px; }      label { display: block; }      input[type=text] { padding: 4px; }    </style>  </head>  <body>    <div id="app">      <h1>주문</h1>      <div>        <label>수량</label>        <input type="text" v-model="quantity" />      </div>      <div>        <label>가격</label>        <input type="text" v-model="price" />      </div>      <hr />      <div>        <label>합계</label>        <input type="text" v-model="total" />      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          quantity: 1,          price: 0        },        computed:  {          total : {            get() {              return this.quantity \* this.price;            },            set(value) {              this.quantity = value / this.price;            }          }        }      })     </script>  </body>  </html> |

### getter / setter 구현

total : {

  get() {

    return this.quantity \* this.price;

  },

  set(value) {

    this.quantity = value / this.price;

  }

}

위 코드의 구조는 아래 코드와 동일하다.

total : {

  get: function() {

    return this.quantity \* this.price;

  },

  set: function(value) {

    this.quantity = value / this.price;

  }

}

total 계산 속성에 getr 메소드 뿐만 아니라, set 메소드도 구현하였다.

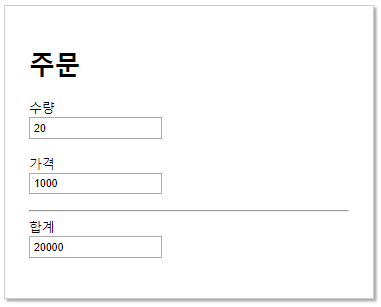
따라서 otal 계산 속성의 값을 읽을 수 있을 뿐만 아니라, 이 속성에 값을 대입할 수도 있다.

<input type="text" v-model="total" />

이 input 태그에 값을 입력하면 total 계산 속성에 그 값이 실시간으로 대입된다.

즉 total 계산 속성의 set 메소드가 즉시 호출된다.

set 메소드의 파라미터 값은, 계산 속성에 대입되는 값이다.



가격을 입력한 후 합계를 입력하면, 그 합계에 맞춰서 수량이 변경된다.

# v-model modifier

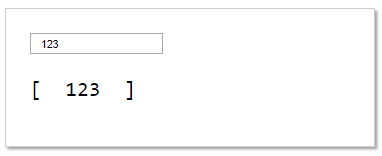
## v-model.trim

### modifier0.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=text] { padding: 4px; }      pre { font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <input type="text" v-model="message" />      <pre>[{{ message }}]</pre>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          message: ''        }      })     </script>  </body>  </html> |

html 태그를 출력할 때, 줄 바꿈 문자나 공백 문자들은, 공백 한개로 압축되어 출력된다.

그런데 pre 태그의 내용은 그렇지 않다. 줄바꿈 문자와 공백 문자들이 모두 출력된다.



입력된 문자열 앞뒤의 공백까지 message 속성에 대입되었다.

### modifier1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=text] { padding: 4px; }      pre { font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <input type="text" v-model.trim="message" />      <pre>[{{ message }}]</pre>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          message: ''        }      })     </script>  </body>  </html> |

<input type="text" v-model.trim="message" />

input 태그에 입력된 값이 실시간으로 message 속성에 대입된다.

그런데 앞뒤의 공백을 제거하고 대입된다.

## v-model.lazy

### modifier2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=text] { padding: 4px; }      pre { font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <input type="text" v-model.lazy="message" />      <pre>[{{ message }}]</pre>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          message: ''        }      })     </script>  </body>  </html> |

<input type="text" v-model.lazy="message" />

input 태그에 입력된 값이 message 속성에 대입된다.

실시간으로 대입되지 않고 늦게 대입된다.

입력을 마치고 엔터키를 누르거나, 다른 입력 칸으로 이동할 때, 그때 대입된다.

<input type="text" v-model.trim.lazy="message" />

trim modifier와 lazy modifier를 둘 다 적용한다.

즉 input 태그에 입력된 값이 message 속성에 늦게 대입되고,

앞뒤 공백을 제거하고 대입된다.

## v-model.number

### modifier3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=text] { padding: 4px; }      div#app div { font-size: 20pt; margin-top: 20px; }    </style>  </head>  <body>    <div id="app">      <input type="text" v-model="number" />      <div>{{ number + number }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          number: 0        }      })     </script>  </body>  </html> |



<input type="text" v-model="number" />

input 태그에 입력된 값은 문자열로 취급된다.

그래서 3을 입력했을 때, number 속성에 대입되는 값은 "3" 이다.

number + number 식의 "3" + "3" 이어서, 결과는 "33" 이다.

### modifier4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=text] { padding: 4px; }      div#app div { font-size: 20pt; margin-top: 20px; }    </style>  </head>  <body>    <div id="app">      <input type="text" v-model.number="number" />      <div>{{ number + number }}</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          number: 0        }      })     </script>  </body>  </html> |

<input type="text" v-model.number="number" />

input 태그에 입력된 값을 number 속성에 대입할 때, number 자료형으로 대입한다.

그래서 3을 입력했을 때, number 속성에 대입되는 값은 숫자 3 이다.

number + number 식의 결과는 6 이다.

# todo 앱1

할일 문자열 목록을 관리하는 todo 앱 구현하기.

## todo10.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      h1 { border-bottom: 1px solid gray; }      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }    </style>  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="todo in todoList">          {{ todo }}        </li>      </ul>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [ "과제", "시험공부", "코딩훈련" ]        }      })     </script>  </body>  </html> |



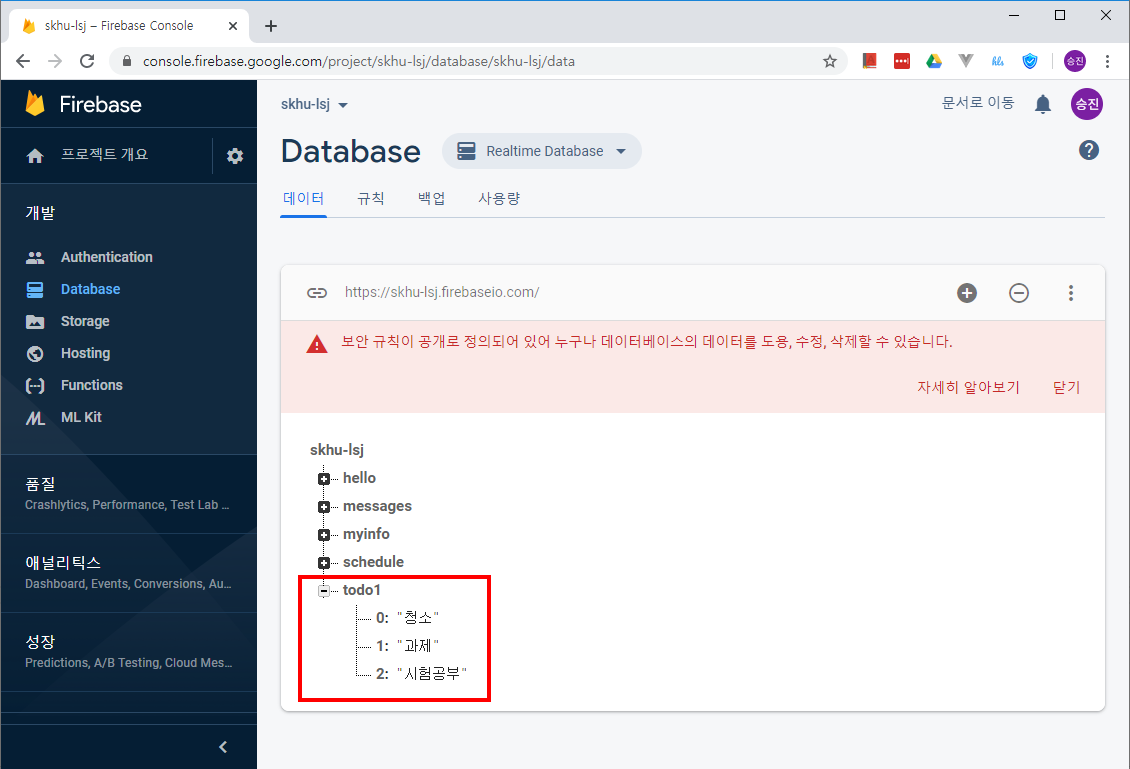
## todo11.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      h1 { border-bottom: 1px solid gray; }      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      li { margin-bottom: 10px; }    </style>  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="todo in todoList">          {{ todo }}        </li>      </ul>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: []        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |

firebase DB 기능을 추가하였다.

app.todoList  = snapshot.val();

firebase DB의 "todo1" 데이터 항목의 값을 앱의 todoList 속성에 대입한다.

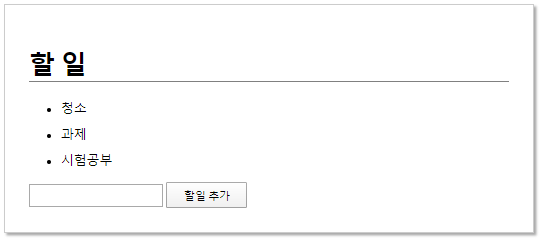


firebase realtime database에 todo1 데이터 항목을 위와 같이 만들자.



## todo12.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      h1 { border-bottom: 1px solid gray; }      div { padding: 30px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      li { margin-bottom: 10px; }      input[type=text] { padding: 5px;  }      button { padding: 0.4em 1.5em; }    </style>  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="todo in todoList">          {{ todo }}        </li>      </ul>      <input type="text" v-model="todo" />      <button type="button" v-on:click="addTodo">할일 추가</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          todo: ""        },        methods: {          addTodo: function() {            this.todoList.push(this.todo);          }        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |



할일 추가 기능 구현

<input type="text" v-model="todo" />

이 input 태그에 입력한 값의 앱의 todo 속성에 대입된다.

<button type="button" v-on:click="addTodo">할일 추가</button>

할일 추가 버튼을 클릭하면 addTodo 메소드가 호출된다.

addTodo: function() {

  this.todoList.push(this.todo);

}

할일 추가 버튼을 클릭하면 이 메소드가 호출된다.

앱의 todoList 속성값은 배열이다.

이 배열의 끝에 todo 값이 추가된다.

### 참고

태그 부분에서 앱의 속성을 사용할 때는 this를 생략한다.

앱의 메소드에서 앱의 속성을 사용할 때는 this를 생략할 수 없다.

### 개선할 점

addTodo 메소드의 구현은, 앱의 todoList 배열에만 항목을 추가할 뿐, DB에 저장하지는 않는다.

firebase DB의 데이터 항목에 저장하는 기능도 구현하자.

## todo1.css

|  |
| --- |
| h1 { border-bottom: 1px solid gray; }  div { padding: 30px; margin: 30px auto; max-width: 600px;    border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }  li { margin-bottom: 10px; }  input[type=text] { padding: 5px;  }  button { padding: 0.4em 1.5em; } |

CSS 서식 부분을 todo1.css 파일로 추출하였다.

이렇체 추출된 css 파일을 html 파일에서 사용하기 위한 태그는 다음과 같다.

  <link rel="stylesheet" type="text/css" href="todo1.css">

## todo13.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo1.css">  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="todo in todoList">          {{ todo }}        </li>      </ul>      <input type="text" v-model="todo" />      <button type="button" v-on:click="addTodo">할일 추가</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          todo: ""        },        methods: {          addTodo: function() {            let s = this.todo.trim();            if (s) {              this.todoList.push(s);              this.todo1Ref.set(this.todoList);            }            this.todo = "";          }        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |

addTodo: function() {

  let s = this.todo.trim();

  if (s) {

    this.todoList.push(s);

    this.todo1Ref.set(this.todoList);

  }

  this.todo = "";

}

let s = this.todo.trim();

this.todo 값은 input 태그에 입력된 값이다.

이 값에서 앞뒤의 공백을 제거한 새 문자열을 만들고, 이 새 문자열을 s에 대입한다.

if (s) {

변수 s의 값이 null 이거나 빈문자열이면 false로 취급되고

그렇지 않으면 true로 취급된다.

즉 input 태그에 입력된 문자열이 있을 때에만 if 문이 true가 된다.

this.todoList.push(s);

this.todoList 목록에 s 값을 추가한다.

this.todo1Ref.set(this.todoList);

this.todoList 값을 firebase DB의 todo1 데이터 항목에 저장한다.

this.todo = "";

앱의 todo 속성 값이 변경되면, 그 변경된 값이 input 태그에 즉시 채워진다.

즉 input 태그가 비워진다.

### 참고

app.todo1Ref = firebase.database().ref("todo1");

firebase DB의 todo1 데이터 항목에 대한 원격 참조 객체를,

Vue 앱의 todo1Ref 속성에 대입했다.

그래서 Vue 앱의 메소드에서 this.todo1Ref 값을 사용할 수 있다.

Vue 앱 객체 외부에서 이 속성을 사용할 때는 app.todo1Ref 이고,

Vue 앱 메소드 내부에서 이 속성을 사용할 때는 this.todo1Ref 이다.

여기서 app은 Vue 앱 객체가 대입된 전역 변수이다.

var app = new Vue(...)

### 개선할 점

할일 삭제 기능을 구현하자.

## todo14.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo1.css">  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="(todo, index) in todoList">          <input type="checkbox" v-model="checked[index]" />          {{ todo }}        </li>      </ul>      <input type="text" v-model="todo" />      <button type="button" v-on:click="addTodo">할일 추가</button>      <button type="button" v-on:click="deleteTodo">삭제</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          checked: [],          todo: ""        },        methods: {          addTodo: function() {            let s = this.todo.trim();            if (s) {              this.todoList.push(s);              this.todo1Ref.set(this.todoList);              this.todo = "";            }          },          deleteTodo: function() {            let list = [];            for (let i = 0; i < this.todoList.length; ++i)              if (!this.checked[i])                list.push(this.todoList[i]);            this.todo1Ref.set(list);            this.checked = [];          }        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |



checked: []

앱에 checked 배열 속성을 만든다.

<input type="checkbox" v-model="checked[index]" />

이 체크박스를 체크를 체크하면, checked 배열의 index 항목에 true 값이 저장된다.

javascript 배열의 원소에 처음으로 값을 대입할 때, 배열의 크기가 자동으로 증가한다.

<button type="button" v-on:click="deleteTodo">삭제</button>

이 버튼을 클릭하면 deleteTodo 메소드가 호출된다.

deleteTodo: function() {

  let list = [];

  for (let i = 0; i < this.todoList.length; ++i)

    if (!this.checked[i])

      list.push(this.todoList[i]);

  this.todo1Ref.set(list);

  this.checked = [];

}

let list = [];

빈 배열을 만든다.

for (let i = 0; i < this.todoList.length; ++i)

  if (!this.checked[i])

    list.push(this.todoList[i]);

todoList 배열의 항목 중에서 체크가 되지 않은 항목들만 list 배열에 채운다.

this.todo1Ref.set(list);

list 배열을 DB의 todo1 항목에 저장한다.

DB의 todo1 항목의 값이 변경되면, 그 값이 다시 앱에 전달될 것이다.

그리고 아래의 코드가 실행되어 앱의 todoList 속성값이 변경될 것이다.

app.todoList  = snapshot.val();

this.checked = [];

checked 배열을 초기화한다.

v-model="checked[index]" 여기서 checked[index] 값이 undefined가 된다.

undefined는 false로 취급되므로, 체크박스들의 체크가 다 꺼지게 된다.

## todo15.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo1.css">    <style> ul { list-style-type: none; } </style>  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="(todo, index) in todoList">          <input type="checkbox" v-model="checked[index]" />          {{ todo }}        </li>      </ul>      <input type="text" v-model="todo" />      <button type="button" v-on:click="addTodo" v-bind:disabled="!todo.trim()">할일 추가</button>      <button type="button" v-on:click="deleteTodo" v-show="checkedCount > 0">삭제</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          checked: [],          todo: ""        },        computed: {          checkedCount: function() {            let count = 0;            for (let i = 0; i < this.checked.length; ++i)              if (this.checked[i]) ++count;            return count;          }        },        methods: {          addTodo: function() {            let s = this.todo.trim();            if (s) {              this.todoList.push(s);              this.todo1Ref.set(this.todoList);              this.todo = "";            }          },          deleteTodo: function() {            let list = [];            for (let i = 0; i < this.todoList.length; ++i)              if (!this.checked[i])                list.push(this.todoList[i]);            this.todo1Ref.set(list);            this.checked = [];          }        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |

<style> ul { list-style-type: none; } </style>

ul 태그 앞 부분의 점을 그리지 않는다.

 체크 박스 앞 부분의 점을 그리지 않는다.

<button type="button" v-on:click="addTodo" v-bind:disabled="!todo.trim()">할일 추가</button>

input 태그에 입력된 값이 없으면, 이 버튼은 비활성화 된다.

<button type="button" v-on:click="deleteTodo" v-show="checkedCount > 0">삭제</button>

체크된 항목이 없으면, 이 버튼은 보이지 않는다.

computed: {

  checkedCount: function() {

    let count = 0;

    for (let i = 0; i < this.checked.length; ++i)

      if (this.checked[i]) ++count;

    return count;

  }

}

checkedCount 속성을 계산 속성으로 구현하였다.

화면 출력과 관련된 값을 계산하는 것은, 메소드로 구현하는 것 보다, 계산 속성으로 구현하는 것이 효율적이다.

화면에 출력하는 값 계산을 메소드로 구현하면, 화면을 다시 그릴 때 마다 메소드가 다시 호출되지만,

계산 속성의 get 메소드의 리턴 값은 캐시에 저장되기 때문에, 화면이 다시 그려질 때 마다 get 메소드가 다시 호출되지는 않는다.

## todo16.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo1.css">    <style> ul { list-style-type: none; } </style>  </head>  <body>    <div id="app">      <h1>할 일</h1>      <ul>        <li v-for="(todo, index) in todoList">          <input type="checkbox" v-model="checked[index]" />          {{ todo }}        </li>      </ul>      <input type="text" v-model.trim="todo" />      <button type="button" v-on:click="addTodo" v-bind:disabled="!todo">할일 추가</button>      <button type="button" v-on:click="deleteTodo" v-show="checkedCount > 0">삭제</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          checked: [],          todo: ""        },        computed: {          checkedCount: function() {            let count = 0;            for (let i = 0; i < this.checked.length; ++i)              if (this.checked[i]) ++count;            return count;          }        },        methods: {          addTodo: function() {            this.todoList.push(this.todo);            this.todo1Ref.set(this.todoList);            this.todo = "";          },          deleteTodo: function() {            let list = [];            for (let i = 0; i < this.todoList.length; ++i)              if (!this.checked[i])                list.push(this.todoList[i]);            this.todo1Ref.set(list);            this.checked = [];          }        }      })      window.onload = function() {        app.todo1Ref = firebase.database().ref("todo1");        app.todo1Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val();        });      }     </script>  </body>  </html> |

<input type="text" v-model.trim="todo" />

input 태그에 입력된 값이, 앞뒤 공백이 제거되고, todo 속성에 대입된다.

<button type="button" v-on:click="addTodo" v-bind:disabled="!todo">

todo 속성의 값이 없으면 이 버튼이 비활성화된다.

없으면 == undefined, null, 빈 문자열

addTodo: function() {

  this.todoList.push(this.todo);

  this.todo1Ref.set(this.todoList);

  this.todo = "";

}

todo 속성값이 있을 때만 버튼이 활성화 되어, 이 메소드가 호출될 수 있으므로,

todo 속성값이 있는지 검사하는 코드를 생략할 수 있다.

# todo 앱2

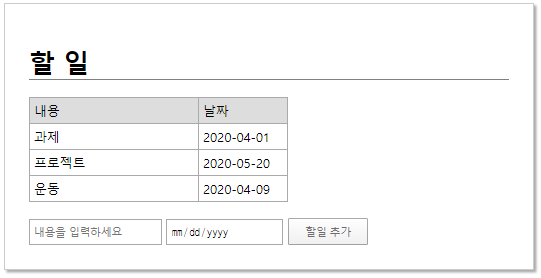
할일 객체 목록을 관리하는 todo 앱 구현하기

## todo2.css

|  |
| --- |
| h1 { border-bottom: 1px solid gray; }  div { padding: 30px; margin: 30px auto; max-width: 600px;    border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa;  }  li { margin-bottom: 10px; }  input[type=text], input[type=date] { height: 2.4em; padding: 4px; }  button { padding: 0.4em 1.5em; }  table { border-collapse: collapse; margin-bottom: 20px; }  tr:nth-child(1) { background-color: #ddd; }  td { padding: 5px;  border: 1px solid #aaa; }  td.checkbox { width: 50px; }  td.description { min-width: 200px; }  td.date { min-width: 100px; } |

## todo20.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo2.css">  </head>  <body>    <div id="app">      <h1>할 일</h1>      <table>        <tr>          <td>내용</td>          <td>날짜</td>        </tr>        <tr v-for="todo in todoList">          <td class="description">{{ todo.description }}</td>          <td class="date">{{ todo.due\_date }}</td>        </tr>      </table>      <input type="text" v-model.trim="todo.description" placeholder="내용을 입력하세요" />      <input type="date" v-model="todo.due\_date" />      <button type="button" v-on:click="addTodo"              v-bind:disabled="!todo.description">할일 추가</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          todo: {}        },        methods: {          addTodo: function() {            this.todoList.push(this.todo);            this.todo2Ref.set(this.todoList);            this.todo = {};          }        }      })      window.onload = function() {        app.todo2Ref = firebase.database().ref("todo2");        app.todo2Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val() || [];        });      }     </script>  </body>  </html> |



app.todo2Ref = firebase.database().ref("todo2");

firebase DB 데이터 항목 "todo2"에 대한 원격 참조 객체를 구한다.

이 데이터 항목을 처음 사용하는 것이라서, 아직 저장된 값이 없을 것이다.

따라서 위 화면에서 할 일 목록도 출력되지 않을 것이다.

app.todoList  = snapshot.val() || [];

snapshot.val() 이 값은 DB에서 전달된 todo2 데이터 항목의 값이다.

firebase DB에 todo2 항목을 아직 만들지 않았다면, snapshot.val() 값은 null 이다.

이 값이 null 이 아니면 그 값이 app.todoList에 대입된다.

이 값이 null 이면, app.todoList에 대입되는 값은 [] 이다.

data: {

  todoList: [],

  todo: {}

},

todoList 속성값은 할 일 목록이다. 이 속성 값은 배열이다.

todo 속성은, input 태그에 입력된 값이 실시간으로 대입되는 속성이다.

toto 속성 값은 객체이다.

<input type="text" v-model.trim="todo.description" placeholder="내용을 입력하세요" />

이 태그에 입력된 값은 todo 객체의 description 속성에 대입된다.

javascript에서는 속성에 처음 값을 대입할 때 그 속성이 생성된다.

<input type="date" v-model="todo.due\_date" />

이 태그에 입력된 값은 todo 객체의 due\_date 속성에 대입된다.

<button type="button" v-on:click="addTodo" v-bind:disabled="!todo.description">할일추가</button>

todo.description 값이 없으면 이 버튼은 비활성화된다.

이 버튼을 클릭하면 addTodo 메소드가 호출된다.

addTodo: function() {

  this.todoList.push(this.todo);

  this.todo2Ref.set(this.todoList);

  this.todo = {};

}

this.todoList.push(this.todo);

todoList 목록 끝에 todo 값을 추가한다.

this.todo2Ref.set(this.todoList);

todoList 목록을 firebase DB의 todo2 항목에 저장한다.

this.todo = {};

todo 객체를 초기화 한다.

이 객체와 연결된 input 태그의 내용도 초기화 된다.

## todo21.html

|  |
| --- |
| <html lang="ko">  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <link rel="stylesheet" type="text/css" href="todo2.css">  </head>  <body>    <div id="app">      <h1>할 일</h1>      <table>        <tr>          <td></td>          <td>내용</td>          <td>날짜</td>        </tr>        <tr v-for="todo in todoList">          <td class="checkbox"><input type="checkbox" v-model="todo.checked" /></td>          <td class="description">{{ todo.description }}</td>          <td class="date">{{ todo.due\_date }}</td>        </tr>      </table>      <input type="text" v-model.trim="todo.description" placeholder="내용을 입력하세요" />      <input type="date" v-model="todo.due\_date" />      <button type="button" v-on:click="addTodo"                            v-bind:disabled="!todo.description">할일 추가</button>      <button type="button" v-on:click="deleteTodo" v-show="checkedCount > 0">삭제</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          todoList: [],          todo: {}        },        computed: {          checkedCount: function() {            let count = 0;            for (let i = 0; i < this.todoList.length; ++i)              if (this.todoList[i].checked) ++count;            return count;          }        },        methods: {          addTodo: function() {            this.todoList.push(this.todo);            this.todo2Ref.set(this.todoList);            this.todo = {};          },          deleteTodo: function() {            let list = [];            for (let i = 0; i < this.todoList.length; ++i)              if (!this.todoList[i].checked)                list.push(this.todoList[i]);            this.todo2Ref.set(list);          }        }      })      window.onload = function() {        app.todo2Ref = firebase.database().ref("todo2");        app.todo2Ref.on('value', (snapshot) => {          app.todoList  = snapshot.val() || [];        });      }     </script>  </body>  </html> |

<input type="checkbox" v-model="todo.checked" />

이 체크박스가 체크되면 todo 객체의 checked 속성값이 true가 된다.

javascript에서는 속성에 처음 값을 대입할 때 그 속성이 생성된다.

deleteTodo: function() {

  let list = [];

  for (let i = 0; i < this.todoList.length; ++i)

    if (!this.todoList[i].checked)

      list.push(this.todoList[i]);

  this.todo2Ref.set(list);

}

let list = [];

list 배열을 생성한다.

for (let i = 0; i < this.todoList.length; ++i)

  if (!this.todoList[i].checked)

    list.push(this.todoList[i]);

todoList 배열의 항목 중에서 checked 속성값이 false 인 것만 list 배열에 추가한다.

this.todo2Ref.set(list);

list 배열을 firebase DB의 todo2 항목에 저장한다.

app.todoList  = snapshot.val() || [];

firebase DB의 todo2 항목의 값이 변경되면, 그 값이 앱으로 전달되고, 위 코드가 실행된다.

그 변경된 값이 앱의 todoList 속성에 대입된다.

