

# **Modul Pemrograman Web Lanjut**

**Semester Gasal 2022/2023**

**Kristian Adi Nugraha, S.Kom., M.T.**



Copyright © 2022 Kristian Adi Nugraha

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

*First printing, Jan 2022*

## Daftar Isi

I	Part 01: Pengenalan Vue	
<b>1</b>	<b>Pengenalan Vue</b> .....	<b>9</b>
1.1	Vue	9
1.2	Kebutuhan Skill	9
1.3	Tools yang Digunakan	10
1.4	Aplikasi Sederhana Menggunakan Vue	10
1.5	Referensi	12
<b>2</b>	<b>Data Binding</b> .....	<b>13</b>
2.1	Pengenalan Data Binding	13
2.2	Data Binding pada Style	13
2.3	Data Binding pada Class	15
2.4	Input Binding	16
2.5	Latihan	18
2.6	Referensi	18
<b>3</b>	<b>Event Handling</b> .....	<b>19</b>
3.1	Javascript Event	19
3.2	Vue Event Handling	19
3.3	Latihan Mandiri	22
3.4	Referensi	23

<b>4</b>	<b>Conditional Rendering</b> .....	<b>25</b>
4.1	Statement Control	25
4.2	v-show	27
4.3	Latihan Mandiri	28
4.4	Referensi	29
<b>5</b>	<b>List Rendering</b> .....	<b>31</b>
5.1	List	31
5.2	Loop Statement	31
5.3	Filtering Data	33
5.4	Key	34
5.5	Latihan Mandiri	35
5.6	Referensi	35

## II

## Part 02: Module System

<b>6</b>	<b>Vue Component</b> .....	<b>39</b>
6.1	Component	39
6.2	Components Properties	40
6.3	Component Methods	41
6.4	Component Model	43
6.5	Latihan Mandiri	44
6.6	Referensi	44
<b>7</b>	<b>Component Lifecycle Hooks</b> .....	<b>45</b>
7.1	create	47
7.2	mount	47
7.3	update	48
7.4	destroy	48
7.5	Latihan Mandiri	49
<b>8</b>	<b>Vue CLI</b> .....	<b>51</b>
8.1	vue create <namaproject>	51
8.2	vue serve	51
8.3	Vue Component	52
8.4	Import Components	53
8.5	Latihan Mandiri	53
<b>9</b>	<b>Building Service with Node JS</b> .....	<b>55</b>
9.1	NodeJS	55
9.2	Express Framework	56

9.3	Mengakses Data pada Express	57
<b>10</b>	<b>Fetching Data from API</b>	<b>59</b>
10.1	Get Method	59
10.2	Post Method	60
10.3	Put Method	60
10.4	Delete Method	61
10.5	Await Syntax	61
10.6	Processing Response	62
10.7	Latihan Mandiri	62
<b>11</b>	<b>Vue Router</b>	<b>63</b>
11.1	SEO Problems	63
11.2	Router	63
11.3	VueRouter	63
11.4	Router Link & View	64
11.5	Programming Route	66
11.6	Dynamic Route	67
11.7	Nested Route	68
11.8	Latihan Mandiri	70
<b>12</b>	<b>Vuex</b>	<b>71</b>
12.1	Instalasi	71
12.2	Struktur Dasar Vuex	71
12.3	Mutations	72
12.4	Actions	73
12.5	Module	73
12.6	Latihan Mandiri	74
<b>13</b>	<b>Vuefire</b>	<b>77</b>
13.1	Firebase	77
13.2	Firestore	77
13.3	Instalasi	77
13.4	Add Data	79
13.5	Get Data	79
13.6	Delete Data	80
13.7	Data Binding	80
<b>14</b>	<b>SCSS</b>	<b>81</b>
14.1	CSS	81
14.2	SCSS	82
14.3	SCSS pada Vue	82

14.4	Variable	83
14.5	Nested	84
14.6	Include	85
14.7	Extend	87
<b>15</b>	<b>Vuetify</b> .....	<b>89</b>
15.1	Material Design	89
15.2	Vuetify	89
15.3	App Bar	90
15.4	Layout	91
15.5	Card	92
15.6	List	93
15.7	Form Input	94
15.8	Grid System	95
	<b>Bibliography</b> .....	<b>97</b>
	Articles	97
	Books	97

# Part 01: Pengenalan Vue

<b>1</b>	<b>Pengenalan Vue</b> .....	<b>9</b>
1.1	Vue	
1.2	Kebutuhan Skill	
1.3	Tools yang Digunakan	
1.4	Aplikasi Sederhana Menggunakan Vue	
1.5	Referensi	
<b>2</b>	<b>Data Binding</b> .....	<b>13</b>
2.1	Pengenalan Data Binding	
2.2	Data Binding pada Style	
2.3	Data Binding pada Class	
2.4	Input Binding	
2.5	Latihan	
2.6	Referensi	
<b>3</b>	<b>Event Handling</b> .....	<b>19</b>
3.1	Javascript Event	
3.2	Vue Event Handling	
3.3	Latihan Mandiri	
3.4	Referensi	
<b>4</b>	<b>Conditional Rendering</b> .....	<b>25</b>
4.1	Statement Control	
4.2	v-show	
4.3	Latihan Mandiri	
4.4	Referensi	
<b>5</b>	<b>List Rendering</b> .....	<b>31</b>
5.1	List	
5.2	Loop Statement	
5.3	Filtering Data	
5.4	Key	
5.5	Latihan Mandiri	
5.6	Referensi	





# 1. Pengenalan Vue

## 1.1 Vue

Vue merupakan Framework berbasis Javascript yang digunakan pada antarmuka (front-end) dari sebuah aplikasi berbasis web. Vue digunakan pada aplikasi web dengan konsep Single-Page Application (SPA), yaitu aplikasi web yang secara dinamis mengubah konten pada bagian-bagian yang diperlukan saja, tanpa harus membuka (load) ulang keseluruhan halaman.

Beberapa keunggulan SPA:

1. Lebih cepat
2. UX lebih baik
3. Mendukung akses offline
4. Caching lebih optimal
5. Hemat resource

Meskipun terdapat banyak kelebihan, namun tetap terdapat beberapa kekurangan dari SPA:

1. Membutuhkan Javascript
2. Permasalahan SEO

Vue bertujuan untuk mempermudah para pengembang dalam membangun dan mengelola aplikasi berbasis web. Vue memiliki cara kerja berbeda dibandingkan dengan library Javascript seperti JQuery, di mana JQuery bekerja dengan cara memilih element-element yang akan diproses dengan menggunakan selector, sedangkan Vue memandang seluruh element dalam satu bagian sebagai sebuah kesatuan komponen. Beberapa keunggulan Vue dibandingkan dengan Javascript Framework yang lain:

1. Ringan
2. Mudah dipelajari
3. Mudah diintegrasikan

## 1.2 Kebutuhan Skill

Untuk menggunakan Vue, skill yang dibutuhkan hanya penguasaan Javascript, karena Vue sepenuhnya menggunakan Javascript. Selain itu dibutuhkan juga penguasaan HTML dan CSS

untuk mengatur bagaimana data akan ditampilkan.

### 1.3 Tools yang Digunakan

Tools yang dibutuhkan untuk membangun SPA menggunakan Vue adalah:

1. Text Editor yang mendukung plugin Vue syntax highlight, misalnya Visual Studio Code
2. Node Package Manager (NPM)
3. Vue CLI

### 1.4 Aplikasi Sederhana Menggunakan Vue

Pertama-tama kita akan membuat sebuah halaman web sederhana berisi tulisan "Hello World!" dengan menggunakan Vue.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Vue - Hello World!</title>
8   <script src="https://cdn.jsdelivr.net/npm/vue"></script>
9 </head>
10 <body>
11   <div id="app">
12     {{ message }}
13   </div>
14
15   <script>
16     var app = Vue.createApp({
17       data() {
18         return {
19           message: "Hello world"
20         }
21       }
22     });
23     app.mount("#app");
24   </script>
25 </body>
26 </html>
```

Output:

A screenshot of a web browser window. The browser's address bar is empty. The main content area of the browser displays the text "Hello World!" in a large, bold, black serif font, centered on the page.

Pada kode program di atas, kita mengimport Vue melalui CDN di alamat `https://cdn.jsdelivr.net/npm/vue`, alamat tersebut akan selalu mengembalikan versi terakhir dari Vue. Apabila kita ingin menggunakan versi tertentu, kita dapat menuliskan versi yang dikehendaki dengan cara menambahkan kode versi setelah simbol `@` seperti contoh berikut: `https://cdn.jsdelivr.net/npm/vue@2.6.14`.

Pada bagian script, kita membuat instance dari Vue dengan memilih element HTML yang akan di-render dengan menggunakan attribute `el`, dalam contoh di atas adalah element `div` dengan id `app`. Dengan demikian, seluruh isi dari element HTML yang dipilih dapat mengandung kode program dari Framework Vue, sedangkan element HTML di luar yang dipilih akan diproses seperti biasa tanpa menggunakan Vue.

Simbol dua kurung kurawal `{{ ... }}` disebut sebagai **interpolation**, seluruh kode yang ada di dalam tanda kurung kurawal akan dianggap sebagai variable atau syntax, sehingga akan di-render oleh Vue. Apabila kita ingin menggunakan sebuah teks statis bersamaan dengan sebuah variable, maka kita dapat menaruh teks tersebut di luar interpolation seperti contoh berikut:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Vue - Hello World!</title>
8   <script src="https://cdn.jsdelivr.net/npm/vue"></script>
9 </head>
10 <body>
11   <div id="app">selamat datang, {{nama}}</div>
12   <script>
13     var app = Vue.createApp({
14       data() {
15         return {
16           nama: "Kristian Adi Nugraha"
17         }
18       }
19     });
20     app.mount("#app");
21   </script>
22 </body>
23 </html>
```

Output:

selamat datang, Adi Nugraha

Pada interpolation kita tidak hanya dapat menaruh variable, namun juga single-expression seperti ternary operator atau increment seperti contoh berikut:

```
1 {{a%2 == 0 ? 'genap' : 'ganjil'}}
2
3 {{counter + 1}}
```

Selain interpolation, syntax tambahan dari Vue adalah HTML directive. Artinya, kita dapat memproses element pada HTML menggunakan Vue dengan cara menanamkan attribute dengan awalan v- di element tersebut, salah satunya adalah v-html. Pada interpolation, tag HTML yang terdapat pada variable akan ditampilkan apa adanya, tidak diproses menggunakan HTML. Jika ingin sebuah variable ditampilkan secara HTML, maka kita harus menggunakan attribute v-html pada element yang bersangkutan:

```
1 <div v-html="output"></div>
2
3 <script>
4     var app = Vue.createApp({
5         data() {
6             return {
7                 nama: "<h1>Kristian Adi Nugraha</h1>"
8             }
9         }
10    });
11    app.mount("#app");
12 </script>
```

Penjelasan syntax directive lainnya secara mendalam akan dibahas secara bertahap pada bab-bab berikutnya.

## 1.5 Referensi

<https://vuejs.org/guide/introduction.html>

## 2. Data Binding

### 2.1 Pengenalan Data Binding

Pada sebuah halaman HTML, kita dapat memanipulasi element di dalamnya dengan menggunakan Javascript. Javascript bekerja dengan cara memilih element berdasarkan id, class, atau parameter lainnya, metode tersebut dikenal dengan istilah **selector**. Data binding memiliki tujuan yang sama dengan selector, yaitu digunakan untuk memanipulasi element pada HTML. Namun data binding memiliki pendekatan yang berbeda dibandingkan selector, yaitu dengan cara mengikat HTML dan Javascript menggunakan sebuah variable (data) yang menghubungkan keduanya. Saat kode Javascript mengubah isi dari variable tersebut, maka element HTML yang terikat akan terkena dampaknya secara langsung.

### 2.2 Data Binding pada Style

Data binding pada Vue dapat dilakukan secara directive dengan menggunakan syntax **v-bind:**, kemudian diikuti oleh nama attribute yang akan dimanipulasi. Misalnya pada contoh kode di bawah ini, kita memiliki sebuah element **span** dengan tulisan berwarna **red** dan memiliki ukuran font **16pt**.

```
1 <span style="color: red; fontSize 16pt">Selamat Pagi</span>
```

Suatu ketika, kita ingin agar dapat mengubah warna dan ukuran dari **span** tersebut secara dinamis, maka kita dapat melakukannya dengan menggunakan data binding. Langkah pertama, tambahkan syntax **v-bind** di depan attribute yang ingin dimanipulasi, dalam kasus ini adalah attribute **style** sehingga menjadi **v-bind:style**. Dengan demikian, isi dari attribute **style** akan diolah menggunakan Vue sehingga kita dapat menggunakan variable di dalamnya. Langkah berikutnya adalah dengan melakukan mapping seluruh attribute yang ingin dimanipulasi dengan menggunakan format object { ... }, sehingga hasilnya menjadi seperti berikut ini (**myColor** dan **mySize** adalah variable):

---

```
1 <span v-bind:style="{color: myColor, fontSize: mySize}">Selamat Pagi</span>
```

---

Karena attribute color dan fontSize pada kode di atas diatur berdasarkan nilai variable myColor dan mySize, maka langkah berikutnya adalah mendeklarasikan kedua variable tersebut ke dalam instance Vue yang dibuat:

---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       myColor: 'red',
5       mySize: '16pt'
6     }
7   }
8 });
9 app.mount('#app');
```

---

Output:



Coba ubah isi dari variable **myColor** dan **mySize**, kemudian reload halaman tersebut. Apabila kita ingin membuat implementasi **v-bind** pada **style** menjadi lebih ringkas, maka kita dapat membungkus isi **v-bind** ke dalam sebuah variable bertipe object seperti contoh berikut:

---

```
1 <span v-bind:style="myStyle">Selamat Pagi</span>
```

---



---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       myStyle: {
5         color: 'red',
6         fontSize: '16pt'
7       }
8     }
9   }
10 });
11 app.mount(#app);
```

---

Selain object, **v-bind** dapat berisi array of object. Apabila diimplementasikan menggunakan array of object, maka jika terdapat dua attribute untuk style yang sama namun berasal dari dua

object yang berbeda, maka yang digunakan adalah attribute dari object yang paling kanan. Pada kode di bawah ini, tulisan 'Selamat Pagi' akan memiliki warna **blue** dengan ukuran font **16pt**.

```
1 <span v-bind:style="[myStyle, myNewStyle]">Selamat Pagi</span>

1 var app = Vue.createApp({
2   data(){
3     return{
4       myStyle: {
5         color: 'red',
6         fontSize: '16pt'
7       },
8       myNewStyle{
9         color: 'blue'
10      }
11    }
12  }
13 });
14 app.mount('#app');
```

Output:



Selamat Pagi

## 2.3 Data Binding pada Class

Selain dapat diterapkan pada attribute **style**, data binding juga dapat diterapkan pada attribute lain seperti **class**. Hal ini cukup berguna saat kita ingin mengatur style sebuah element namun tidak menggunakan tipe inline. Pada CSS internal maupun external, kita mengatur tampilan sebuah element berdasarkan nama class yang diasosiasikan terhadap syntax CSS yang didefinisikan sebelumnya seperti pada contoh di bawah ini:

```
1 span{
2   color: black;
3 }
4
5 .aktif{
6   color: red;
7 }
```

Kemudian kita memiliki element **span** seperti berikut:

---

```
1 <span class="aktif">Selamat Pagi</span>
```

---

Apabila kita ingin menerapkan data binding pada kode di atas, maka kita dapat menambahkan **v-bind** pada attribute **class** menjadi **v-bind:class**. Kemudian variable yang akan digunakan untuk menentukan apakah class tersebut akan ditambahkan pada sebuah element atau tidak harus bersifat truthy (boolean), dalam kode di bawah ini variable tersebut bernama **isAktif**.

---

```
1 <span v-bind:class="{aktif: isAktif}">Selamat Pagi</span>
```

---

Kemudian tambahkan variable tersebut ke dalam script Vue:

---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       isAktif: true
5     }
6   }
7 });
8 app.mount('#app').
```

---

Output:



Coba ubah nilai variable **isAktif** menjadi false, kemudian reload halaman web. Apabila kita tetap menginginkan terdapat class yang statis tanpa diatur oleh variable, kita dapat mencampurnya dengan attribute **class** biasa.

---

```
1 <span class="judul" v-bind:class="{aktif: isAktif}">Selamat Pagi</span>
```

---

## 2.4 Input Binding

Implementasi data binding dengan menggunakan **v-bind** bersifat satu arah, artinya element HTML hanya bertugas untuk menerima nilai dari variable yang mengikatnya. Namun nantinya akan terdapat situasi di mana kita harus mengimplementasikan data binding secara dua arah, misalnya pada element HTML yang merupakan input. Element input bertugas untuk mengembalikan isi dari variable setelah user memasukkan nilai ke dalam element tersebut. Untuk mengimplementasikan data binding secara dua arah pada input, kita dapat menggunakan **syntax v-model** berisi nama variable yang mengikat element tersebut, misalnya ketika kita ingin user menginputkan username:



---

```
1 <input v-model="username" placeholder="masukkan username ...">
```

---

Pada bagian script Vue, kita cukup menyiapkan variable seperti biasa:

---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       username: ''
5     }
6   }
7 });
8 app.mount('#app');
```

---

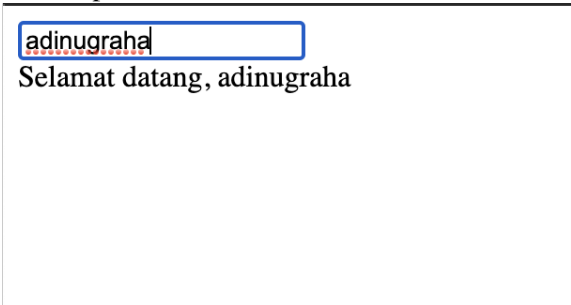
Untuk membuktikan bahwa variable tersebut bersifat dua arah, maka tambahkan sebuah element untuk menampilkan hasil dari input user:

---

```
1 <input v-model="username" placeholder="masukkan username ...">
2 <div>Selamat datang, {{username}}</div>
```

---

Output:



adinugraha  
Selamat datang, adinugraha

Pada input lain seperti checkbox, di mana kita dapat memilih lebih dari satu opsi, maka variable yang digunakan untuk data binding harus bertipe array. Selain itu, attribute value wajib didefinisikan karena isi dari array tergantung dari attribute value yang dipilih. Jika attribute value tidak didefinisikan, maka variable akan berisi tipe data boolean yang menandakan apakah checkbox tersebut dipilih atau tidak.

---

```
1 <input id="makan" value="makan" type="checkbox" v-model="hobi" />
2 <label for="makan">Makan</label>
3 <input id="tidur" value="tidur" type="checkbox" v-model="hobi" />
4 <label for="makan">Tidur</label>
5 <input id="bersepeda" value="bersepeda" type="checkbox" v-model="hobi" />
6 <label for="makan">Bersepeda</label>
7 <div>Hobi saya: {{hobi}}</div>
```

---



---

```
1 var app = Vue.createApp({
2   data(){
3     return{
```

---

```
4         hobi: []  
5     }  
6 }  
7 });  
8 app.mount(("#app"));
```

Output:

☒ Makan ☐ Tidur ☒ Bersepeda  
Hobi saya: [ "makan", "bersepeda" ]

Untuk element lain seperti **radio** memiliki implementasi yang mirip seperti pada **checkbox**, sedangkan pada **select** memiliki implementasi yang sama seperti pada input **text** biasa.

## 2.5 Latihan

Sebagai latihan, buatlah form input seperti contoh di bawah ini beserta implementasi data bindingnya! (warna tulisan menyesuaikan opsi radio button yang dipilih)

Adi 17 ☐ Merah ☒ Biru  
Perkenalkan nama saya Adi, usia 17th, warna kesukaan saya adalah biru

## 2.6 Referensi

<https://vuejs.org/guide/essentials/forms.html>

## 3. Event Handling

### 3.1 Javascript Event

Javascript pada aplikasi web seringkali diimplementasikan dengan menggunakan paradigma **event-driven programming**, artinya kode program akan berjalan saat terjadi event / kejadian tertentu. Beberapa contoh event tersebut adalah saat sebuah element di-klik (**onclick**) atau data di dalamnya berubah (**onchange**). Konsep event tersebut banyak digunakan saat aplikasi meminta pengguna memasukkan input ke dalam text field, checkbox, radio button, dan sebagainya.

### 3.2 Vue Event Handling

Pada Vue, implementasi event handling tidak jauh berbeda dibandingkan dengan Javascript biasa. Misalnya saat kita akan membuat sebuah tombol melakukan sesuatu saat di-klik, maka kita dapat membuatnya seperti ini:

```
1 <button onclick="doSomething()">Klik saya</button>
```

Sedangkan pada Vue, kita cukup menggunakan syntax **v-on:** yang diikuti dengan jenis event yang diinginkan. Untuk contoh di atas, kita dapat membuatnya dalam Vue menjadi seperti berikut:

```
1 <button v-on:click="doSomething()">Klik saya</button>
```

Selain dengan cara di atas, kita juga dapat menggunakan simbol **@** diikuti dengan nama event seperti contoh berikut:

```
1 <button @click="doSomething()">Klik saya</button>
```

Pada implementasi event handling, kita dapat langsung menuliskan baris kode program secara inline tanpa memanggil sebuah function seperti contoh di bawah ini.

---

```

1 <button v-on:click="count++">Tambah</button>
2 <span>Count: {{count}}</span>

```

---

Namun jika baris kode program cukup panjang, maka disarankan untuk menggunakan function agar lebih rapi.

Jika kita menggunakan Javascript biasa, function **doSomething()** dapat ditempatkan di mana-pun. Sedangkan pada Vue, function **doSomething()** harus dideklarasikan saat pembuatan instance Vue di dalam attribute **methods** seperti berikut:

---

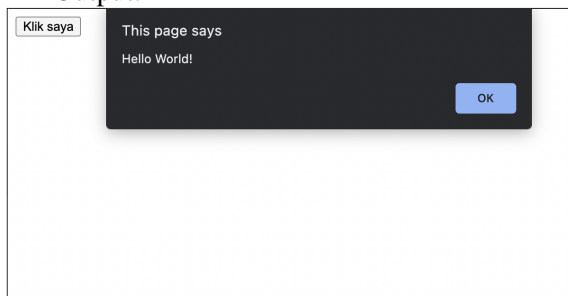
```

1 var app = Vue.createApp({
2   methods:{
3     doSomething(){
4       alert('Hello World!');
5     }
6   }
7 });
8 app.mount('#app');

```

---

Output:



Dengan mengkombinasikan event dan data binding, maka kita dapat mengirimkan parameter pada function sesuai dengan input dari user:

---

```

1 <input v-model="nama" placeholder="masukkan nama ..."/>
2 <button v-on:click="doSomething(nama)">Klik saya</button>

```

---



---

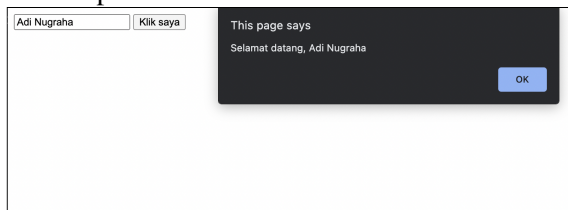
```

1 var app = Vue.createApp({
2   data(){
3     return{
4       nama: ''
5     }
6   },
7   methods:{
8     doSomething(nama){
9       alert('Selamat datang, '+nama);
10    }
11  }
12 });
13 app.mount('#app');

```

---

Output:



Sebagai alternatif lain, karena variable **nama** tersebut terdapat pada instance Vue yang sama, maka kita juga dapat mengaksesnya secara langsung di dalam method **doSomething()** dengan menggunakan keyword **this**, sehingga pada event **onclick** kita tidak perlu lagi mengirimkan parameter **nama**.

---

```
1 <input v-model="nama" placeholder="masukkan nama ..."/>
2 <button v-on:click="doSomething()">Klik saya</button>
```

---



---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       nama: ''
5     }
6   },
7   methods:{
8     doSomething(){
9       alert('Selamat datang, '+this.nama);
10    }
11  }
12 });
13 app.mount('#app');
```

---

Pada Vue, kita juga dapat mengakses variable event dari DOM jika diperlukan dengan menyebutkan parameter event pada saat pemanggilan function seperti contoh berikut:

---

```
1 <button v-on:click="doSomething('halo',\$event)">Tambah nilai</button>
2
3 var app = Vue.createApp({
4   methods:{
5     doSomething(message, event){
6       if (event) {
7         event.preventDefault()
8       }
9       alert(message);
10    }
11  }
12 });
13 app.mount('#app');
```

---

Seluruh event Javascript dapat digunakan pada Vue dengan menghilangkan kata **on** di depannya:

1. onclick
2. onload
3. onmouseover
4. onmousedown
5. onmousemove
6. onblur
7. onchange
8. onfocus
9. onselect
10. onsubmit
11. onreset
12. onkeydown
13. onkeyup
14. onkeypress

Pada key modifiers (keyup, keydown, keypress), kita dapat mentrigger event saat menekan key tertentu seperti berikut:

```

1 <input @keyup.enter="doSomething()" />
2 <input @keyup.space="doSomething()" />
3 <input @keyup.tab="doSomething()" />
4 <input @keyup.delete="doSomething()" /> <!-- Tekan Delete atau Backspace -->

```

Kita juga dapat menggunakan kombinasi dua key sekaligus seperti berikut:

```

1 <input @keyup.alt.enter="doSomething()"> <!-- Tekan Alt + Enter -->
2 <button @click.ctrl="doSomething()">Tekan Mouse Click + Ctrl</button>

```

Pada contoh di atas, jika terdapat key lain yang ditekan bersamaan, maka event tetap akan tertrigger (misalnya Shift + Click + Ctrl), jika kita ingin hanya key tersebut yang ditekan untuk mentrigger event, maka kita dapat menambahkan modifier exact:

```

1 <!-- Tetap berjalan saat tombol shift ditekan -->
2 <button @click.ctrl="doSomething()">Tekan Mouse Click + Ctrl</button>
3
4 <!-- Tidak akan berjalan saat tombol shift ditekan -->
5 <button @click.ctrl.exact="doSomething()">Tekan Mouse Click + Ctrl</button>

```

### 3.3 Latihan Mandiri

Buatlah sebuah aplikasi sederhana untuk melakukan pencarian data nama orang dengan input berupa text. Setiap user mengetikkan 1 karakter, aplikasi akan memfilter nama-nama yang mengandung susunan karakter sesuai input dari user. (Definisikan data nama secara statis, minimal 5 nama orang)

Hasil:

adi

ade

### 3.4 Referensi

<https://vuejs.org/guide/essentials/event-handling.html>





## 4. Conditional Rendering

### 4.1 Statement Control

Seperti yang kita ketahui, di setiap bahasa pemrograman terdapat mekanisme yang disebut sebagai statement control, artinya kita dapat membuat lebih dari satu kemungkinan alur program dengan mempertimbangkan kondisi yang ada. Statement control yang paling populer biasanya dibuat dengan menggunakan syntax **if-else**, selain itu terdapat beberapa syntax lain seperti **switch** atau **when**. Pada Vue, terdapat statement control yang bertujuan untuk melakukan conditional rendering, yaitu me-render element HTML berdasarkan kondisi yang ada. Terdapat tiga buah syntax yang dapat digunakan untuk conditional rendering, yaitu **v-if**, **v-else**, dan **v-else-if**. Isi di dalam attribute **v-if** dan **v-else-if** harus merupakan sebuah kondisi yang dapat mengembalikan nilai **true** atau **false**, sehingga dapat diisi dengan menggunakan variable bertipe boolean atau conditional statement. Misalnya ketika kita hendak menentukan apakah sebuah **button** akan ditampilkan atau tidak berdasarkan isi dari variable **aktif**. **Button** hanya akan ditampilkan saat variable **aktif** bernilai **true**.

```
1 <button v-if="aktif">Tombol sedang aktif</button>
```

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       aktif: true
5     }
6   }
7 });
8 app.mount('#app');
```

Apabila kita ingin menampilkan tombol lain saat variable **aktif** bernilai **false**, maka kita dapat menambahkan syntax **v-else** pada komponen yang hendak ditampilkan. Perlu dicatat bahwa syntax

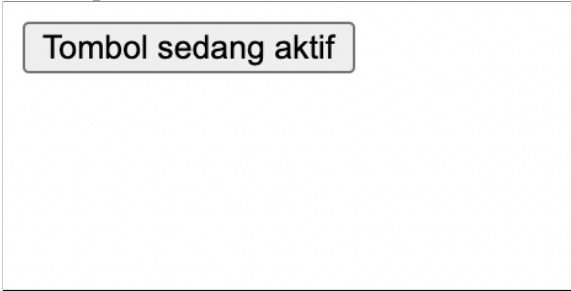
**v-else** hanya dapat dipasang pada element tepat setelah syntax **v-if** atau **v-else-if**, selebihnya syntax tersebut tidak akan berlaku.

---

```
1 <button v-if="aktif">Tombol sedang aktif</button>
2 <button v-else>Tombol sedang tidak aktif</button>
```

---

Output variable **aktif** bernilai **true**:



Tombol sedang aktif

Output variable **aktif** bernilai **false**:



Tombol sedang tidak aktif

Selain itu, apabila dibutuhkan kita juga dapat memiliki lebih dari dua kondisi dengan menggunakan syntax **v-else-if**. Sama seperti **v-else**, syntax **v-else-if** harus langsung dipasang pada element tepat setelah syntax **v-if**. Misalnya jika kita ingin menampilkan sebuah pernyataan (baik, cukup, kurang) berdasarkan isi dari variable nilai.

---

```
1 <div id="app">
2   Ujian anda dinyatakan:
3   <span v-if="nilai > 60">Baik</span>
4   <span v-else-if="nilai > 30">Cukup</span>
5   <span v-else>Kurang</span>
6 </div>
```

---

---

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       nilai: 50
5     }
6   }
7 });
8 app.mount('#app');
```

---

Output:

Ujian anda dinyatakan: Cukup

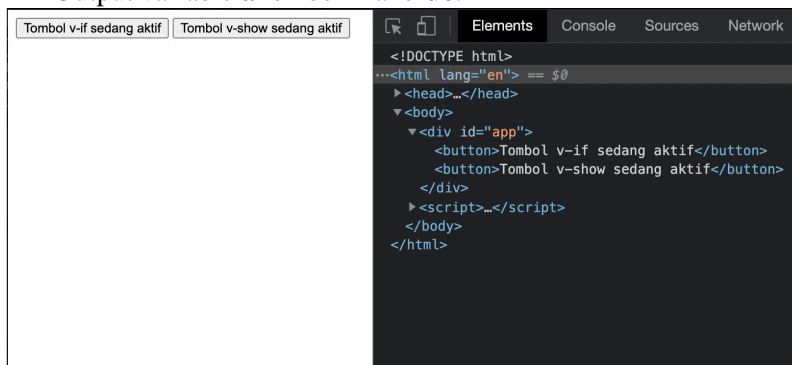
## 4.2 v-show

Selain **v-if**, terdapat syntax lain yang dapat digunakan untuk conditional rendering yaitu **v-show**. Perbedaan utama antara **v-if** dan **v-show** adalah pada **v-if**, jika kondisi bernilai **false** maka element tidak akan di-render, sedangkan pada **v-show** element tetap akan di-render namun disembunyikan dengan menggunakan CSS. Untuk membuktikan pertanyaan tersebut, bandingkan kode dari implementasi **v-if** dan **v-show**.

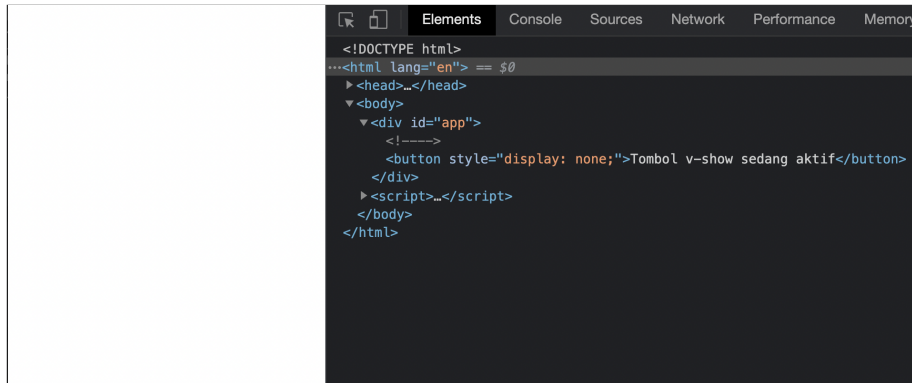
```
1 <button v-if="aktif">Tombol v-if sedang aktif</button>
2 <button v-show="aktif">Tombol v-show sedang aktif</button>
```

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       aktif: true
5     }
6   }
7 });
8 app.mount('#app');
```

Output variable **aktif** bernilai **true**:



Output variable **aktif** bernilai **false**:



Perhatikan source code di atas, saat variable **aktif** bernilai **true**, kedua button sama-sama muncul pada HTML. Sedangkan saat variable **aktif** bernilai **false**, hanya **button** dari **v-show** yang terdapat pada HTML, namun dengan kondisi CSS **display:none** agar element tidak muncul pada tampilan. Syntax **v-show** akan berguna saat element akan sering dimunculkan dan disembunyikan secara bergantian, karena mekanisme pemrosesan yang lebih ringan dengan menggunakan CSS. Sedangkan pada **v-if**, setiap kali kondisi berganti akan dilakukan rendering ulang terhadap tampilan sehingga proses akan lebih berat jika dibandingkan dengan **v-show**, oleh karena itu, **v-if** akan optimal jika element akan di-render sebanyak satu kali saja. Kekurangan dari **v-show** adalah kita tidak dapat memiliki kondisi alternatif, karena syntax **v-show** tidak mendukung penggunaan kombinasi syntax dengan syntax **v-else** maupun **v-else-if**.

### 4.3 Latihan Mandiri

Buatlah sebuah halaman login sederhana, di mana halaman tersebut hanya meminta input berupa username saja. Kemudian saat tombol login ditekan, halaman akan berubah menjadi halaman logout seperti contoh di bawah ini:

Kondisi halaman sebelum login:

Kondisi halaman setelah login:

Selamat datang adinugraha

Logout

#### 4.4 Referensi

<https://vuejs.org/v2/guide/conditional.html>



## 5. List Rendering

### 5.1 List

List merupakan sebuah komponen yang di dalamnya terdapat banyak item dengan tampilan yang serupa. List biasanya digunakan untuk menampilkan data record yang didapatkan dari database, array, atau struktur data lainnya. Beberapa contoh dari list yang sering kita jumpai adalah daftar barang hasil pencarian di marketplace atau isi chat pada sebuah room.

### 5.2 Loop Statement

Pada Vue, kita dapat me-render sebuah list dengan menggunakan syntax loop statement bernama **v-for**. Sama seperti syntax **for** di bahasa pemrograman pada umumnya, **v-for** akan melakukan iterasi berdasarkan variable array yang dipasang di dalamnya, kemudian melakukan setiap iterasinya dengan menciptakan element HTML sesuai nilai yang terdapat pada setiap item. Contohnya jika kita ingin menampilkan daftar belanja yang datanya didapatkan dari sebuah array:

```
1 <ul>
2   <li v-for="item in items">{{ item }}</li>
3 </ul>
```

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       items: ["Tomat", "Wortel", " Kol"]
5     }
6   }
7 });
8 app.mount('#app');
```

Output:

- Tomat
- Wortel
- Kol

Apabila pada setiap iterasi kita membutuhkan lebih dari 1 variable, maka kita dapat membuatnya menjadi object:

```
1 <ul>
2   <li v-for="item in items">{{ item.jumlah+'x '+item.nama }}</li>
3 </ul>
```

```
1 var app = Vue.createApp({
2   data(){
3     return{
4       items: [
5         {jumlah: 3, nama: 'Tomat'},
6         {jumlah: 5, nama: 'Wortel'},
7         {jumlah: 1, nama: 'Kol'},
8       ]
9     }
10  }
11 });
12 app.mount('#app');
```

Output:

- 3x Tomat
- 5x Wortel
- 1x Kol

Kita dapat melakukan operasi manipulasi pada array seperti **push()**, **pop()**, dan lain-lain, kemudian syntax **v-for** akan me-render ulang element di dalamnya apabila terdapat perubahan data pada array. Untuk mencobanya, silakan tambahkan input untuk jumlah dan nama barang ke dalam daftar belanja.



---

```

1 <div id="app">
2   <input type="text" v-model="jumlah" placeholder="jumlah ..." />
3   <input type="text" v-model="nama" placeholder="nama ..." />
4   <button v-on:click="items.push({jumlah, nama})">Tambahkan</button>
5   <ul>
6     <li v-for="item in items">{{item.jumlah+'x '+item.nama}}</li>
7   </ul>
8 </div>

```

---

```

1 var app = Vue.createApp({
2   data(){
3     return{
4       jumlah: '',
5       nama: '',
6       items: [
7         {jumlah: 3, nama: 'Tomat'},
8         {jumlah: 5, nama: 'Wortel'},
9         {jumlah: 1, nama: 'Kol'},
10      ]
11    }
12  }
13 });
14 app.mount('#app');

```

---

Output:

- 3x Tomat
- 5x Wortel
- 1x Kol
- 2x Kentang

## 5.3 Filtering Data

Pada data yang akan ditampilkan, kita dapat melakukan filtering agar hanya beberapa data saja yang ditampilkan berdasarkan kriteria yang ada. Cara yang paling efisien untuk melakukan filterisasi adalah dengan membuat function di attribute **computed** pada instance Vue. Misalnya kita hanya ingin menampilkan bilangan genap saja dari sebuah array:

---

```

1 var app = Vue.createApp({
2   data(){
3     return{
4       numbers: [1,2,3,4,5,6]

```

---

```

5     }
6   },
7   computed: {
8     evenNumbers(){
9       return this.numbers.filter((n)=>n%2==0);
10    }
11  }
12 });
13 app.mount('#app');
```

Pada HTML, kita cukup memanggil `evenNumbers` seperti memanggil variable biasa:

```

1 <ul>
2   <li v-for="number in evenNumbers">{{number}}</li>
3 </ul>
```

Output:

- 2
- 4
- 6

Apa bedanya cara di atas dibandingkan membuat function biasa di attribute **methods**? Secara umum **computed** juga berisi function, namun function yang terdapat pada **computed** hanya akan dijalankan ulang saat terdapat perubahan nilai pada dependency reactive yang ada di dalamnya, sedangkan function pada **methods** akan selalu dijalankan kembali saat element HTML di-render ulang.

## 5.4 Key

Pada element yang dibangun melalui setiap iterasi menggunakan **v-for**, sebaiknya memiliki attribute **:key** di dalamnya. **:key** merupakan sebuah syntax yang membedakan antara item satu dengan yang lainnya, sehingga harus bernilai unik. Apabila tidak menggunakan **:key**, maka list akan selalu di-render ulang apabila terdapat perubahan data pada **array**. Sebaliknya, jika terdapat attribute **:key**, maka list hanya akan disusun ulang menggunakan item yang sudah ada berdasarkan susunan key yang baru. Isi dari **:key** biasanya merupakan sebuah id (primary key) yang ada pada table database atau bisa berupa nilai unik lainnya.

```

1 <ul>
2   <li v-for="item in listItem" :key="item.id">{{item.name}}</li>
3 </ul>
```

## 5.5 Latihan Mandiri

Buatlah sebuah halaman login sederhana untuk menampilkan daftar percakapan chat suatu grup. Buatlah beberapa contoh data dummy, di mana setiap item memiliki attribute **username** pengirim dan **message** dari chat. Buatlah satu buah input text untuk menentukan user siapa yang sedang login, kemudian buatlah tampilan user tersebut berbeda dibanding dengan yang lain.

Output:

Login as:

adinugraha: halo  
ekonugroho: halo juga  
aguskurniawan: gimana kabarnya?  
ekonugroho: saya sehat  
adinugraha: saya juga sehat  
aguskurniawan: sama dong

Kemudian, buatlah sebuah array yang berisi daftar **username** beserta nomor **handphone** masing-masing user. Apabila salah satu chat di-klik, maka aplikasi akan menampilkan data username beserta nomor handphonenya di bagian paling bawah seperti contoh berikut:

Output:

Login as:

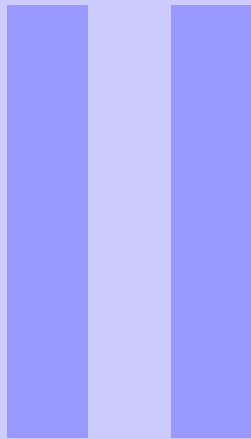
adinugraha: halo  
ekonugroho: halo juga  
aguskurniawan: gimana kabarnya?  
ekonugroho: saya sehat  
adinugraha: saya juga sehat  
aguskurniawan: sama dong

**Username: ekonugroho, HP: 45678**

## 5.6 Referensi

<https://vuejs.org/v2/guide/list.html>





## Part 02: Module System

<b>9</b>	<b>Building Service with Node JS</b>	<b>55</b>
9.1	NodeJS	
9.2	Express Framework	
9.3	Mengakses Data pada Express	
<b>10</b>	<b>Fetching Data from API</b>	<b>59</b>
10.1	Get Method	
10.2	Post Method	
10.3	Put Method	
10.4	Delete Method	
10.5	Await Syntax	
10.6	Processing Response	
10.7	Latihan Mandiri	
<b>11</b>	<b>Vue Router</b>	<b>63</b>
11.1	SEO Problems	
11.2	Router	
11.3	VueRouter	
11.4	Router Link & View	
11.5	Programming Route	
11.6	Dynamic Route	
11.7	Nested Route	
11.8	Latihan Mandiri	
<b>12</b>	<b>Vuex</b>	<b>71</b>
12.1	Instalasi	
12.2	Struktur Dasar Vuex	
12.3	Mutations	
12.4	Actions	
12.5	Module	
12.6	Latihan Mandiri	
<b>13</b>	<b>Vuefire</b>	<b>77</b>
13.1	Firebase	
13.2	Firestore	
13.3	Instalasi	
13.4	Add Data	
13.5	Get Data	
13.6	Delete Data	
13.7	Data Binding	
<b>14</b>	<b>SCSS</b>	<b>81</b>
14.1	CSS	
14.2	SCSS	
14.3	SCSS pada Vue	
14.4	Variable	
14.5	Nested	
14.6	Include	
14.7	Extend	
<b>15</b>	<b>Vuetify</b>	<b>89</b>
15.1	Material Design	
15.2	Vuetify	
15.3	App Bar	
15.4	Layout	
15.5	Card	
15.6	List	
15.7	Form Input	
15.8	Grid System	
	<b>Bibliography</b>	<b>97</b>
	Articles	
	Books	



## 6. Vue Component

### 6.1 Component

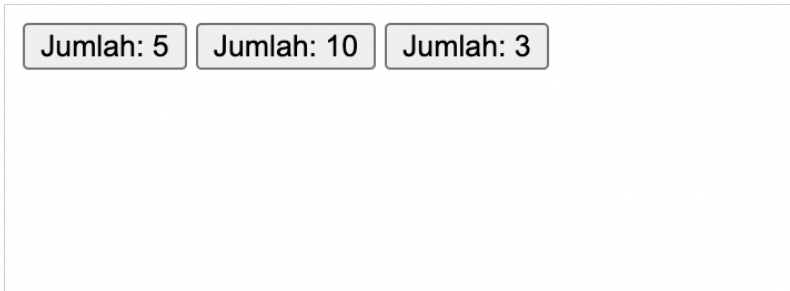
Component merupakan Vue instance yang dapat digunakan berulang-kali (re-use). Setiap component yang hendak digunakan berulang kali harus diregistrasikan terlebih dahulu dengan cara memanggil method **component()** pada class **Vue**. Setiap component yang akan diregistrasikan harus memiliki nama, parameter script, dan template HTML seperti contoh berikut:

```
1 Vue.component('button-counter', {  
2   data(){  
3     return {  
4       count: 0  
5     }  
6   },  
7   template: '<button v-on:click="count++">Jumlah: {{count}}</button>'  
8 });  
9  
10 new Vue({el: '#app'});
```

Kemudian, coba gunakan element **button-counter** pada HTML:

```
1 <div id="app">  
2   <button-counter></button-counter>  
3   <button-counter></button-counter>  
4   <button-counter></button-counter>  
5 </div>
```

Output:



Pada contoh di atas, dapat terlihat bahwa setiap **button-counter** memiliki lifecycle masing-masing, ditandai dengan nilai **count** yang berbeda-beda untuk setiap element. Cara di atas akan meregistrasikan component secara global, apabila sebuah component hanya akan digunakan pada satu instance saja, maka kita dapat meregistrasikan component tersebut secara local.

```

1  new Vue({
2    el: '#app',
3    components: {
4      'button-counter': {
5        data(){
6          return {
7            count: 0
8          }
9        },
10       template: '<button v-on:click="count++">Jumlah: {{count}}</button>'
11     }
12   }
13 });

```

## 6.2 Components Properties

Pada beberapa situasi, setiap komponen memiliki nilai attribute atau behaviour yang berbeda antara satu dengan yang lainnya. Agar dapat memberikan parameter unik untuk setiap component, terdapat attribute **props** yang mendefinisikan property apa saja yang dapat dikirimkan ke dalam component yang hendak dibuat.

```

1  Vue.component('button-counter', {
2    props: ['nama', 'ipk'],
3    template: '<div>{{nama}}: {{ipk}}</div>'
4  });
5
6  new Vue({el: '#app'});

```

```

1  <div id="app">
2    <mahasiswa nama="Adi Nugraha" ipk="4.00"></mahasiswa>
3    <mahasiswa nama="Budi Gunawan" ipk="3.00"></mahasiswa>
4    <mahasiswa nama="Eko Kurniawan" ipk="3.50"></mahasiswa>
5  </div>

```



Output:

```
Adi Nugraha: 4.00
Budi Gunawan: 3.00
Eko Kurniawan: 3.50
```

Apabila data mahasiswa disimpan dalam sebuah array, maka kita dapat menggunakan kombinasi **v-for** dan **v-bind** untuk menanamkan data ke dalam component.

```
1 Vue.component('mahasiswa', {
2   props: ['nama','ipk'],
3   template: '<div>{{nama}}: {{ipk}}</div>'
4 });
5
6 new Vue({
7   el: '#app',
8   data(){
9     return{
10      mahasiswa: [
11        {nama: "Adi Nugraha", ipk: 4.00},
12        {nama: "Budi Gunawan", ipk: 3.00},
13        {nama: "Eko Kurniawan", ipk: 3.50},
14      ]
15    }
16  }
17 });
```

```
1 <div id="app">
2   <mahasiswa v-for="m in mahasiswa" v-bind:nama="m.nama" v-bind:ipk="m.ipk"></mahasiswa>
3 </div>
```

## 6.3 Component Methods

Method pada component dapat ditambahkan melalui attribute **methods** seperti instance Vue pada umumnya. Pemanggilan event dapat dilakukan dengan menuliskan syntax **v-on** pada element yang bersangkutan:

```
1 Vue.component('mahasiswa', {
2   props: ['nama','ipk'],
3   methods:{
4     info(nama, ipk){
5       alert('Nama saya: '+nama+', IPK saya: '+ipk)
```

```

6      }
7    },
8    template: '<div>{{nama}}: {{ipk}} <button v-on:click="info(nama, ipk)">Info</button></div>',
9  });
10
11  new Vue({
12    el: '#app',
13    data(){
14      return{
15        mahasiswa: [
16          {nama: "Adi Nugraha", ipk: 4.00},
17          {nama: "Budi Gunawan", ipk: 3.00},
18          {nama: "Eko Kurniawan", ipk: 3.50},
19        ]
20      }
21    }
22  });

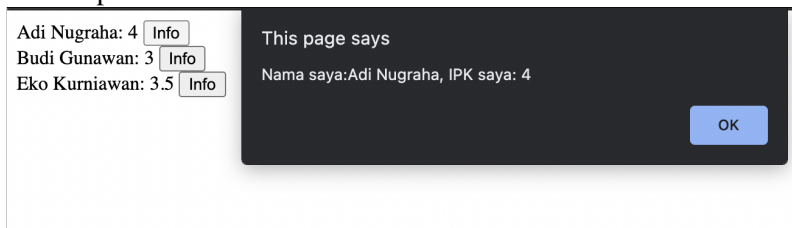
```

```

1  <div id="app">
2    <mahasiswa v-for="m in mahasiswa" v-bind:nama="m.nama" v-bind:ipk="m.ipk"></mahasiswa>
3  </div>

```

Output:



Bagaimana jika method yang akan dipanggil adalah milik parent instance Vue? Untuk melakukannya, kita harus membuat custom event yang dipanggil dari child. Misalnya terdapat method pada parent bernama parentInfo() seperti berikut:

```

1  new Vue({
2    el: '#app',
3    data(){
4      return{
5        mahasiswa: [
6          {nama: "Adi Nugraha", ipk: 4.00},
7          {nama: "Budi Gunawan", ipk: 3.00},
8          {nama: "Eko Kurniawan", ipk: 3.50},
9        ]
10     }
11   },
12   methods: {
13     parentInfo(nama){
14       alert(nama+ ' memanggil method milik parent');

```

```

15     }
16   }
17 });

```

Kemudian pada saat registrasi component, tambahkan event `v-on:click` pada button dengan memanggil method `$emit()` di dalamnya. Method `$emit()` bertujuan agar button dapat memerintahkan element `<mahasiswa>` sebagai perantara untuk memanggil method `parentInfo()`.

```

1 Vue.component('mahasiswa', {
2   props: ['nama', 'ipk'],
3   methods: {
4     info(nama, ipk) {
5       alert('Nama saya: '+nama+', IPK saya: '+ipk)
6     }
7   },
8   template: '<div>{{nama}}: {{ipk}} '+
9     '<button v-on:click="$emit('parent-info', nama)'>Info</button></div>'
10 });

```

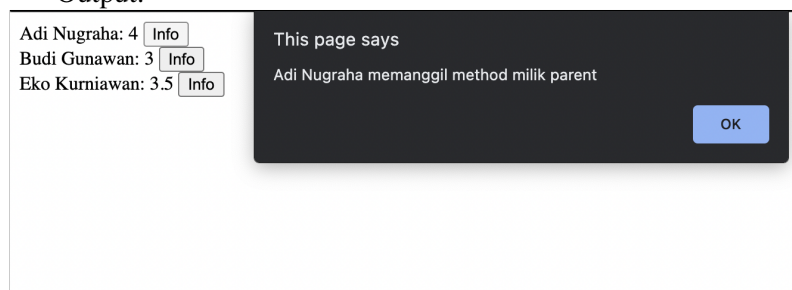
Pada HTML, tambahkan custom event `parent-info` pada component yang dibuat. Method `$emit` harus memanggil custom event menggunakan nama yang tertera pada component, yaitu **parent-info**. Apabila pemanggilan method harus mengirimkan parameter, maka parameter tersebut harus ditampung ke dalam sebuah variable bernama `$event`. Saat pemanggilan custom event dari child komponen melalui `$emit()` parameter pertama adalah nama custom event, sedangkan parameter kedua adalah data yang hendak dikirimkan.

```

1 <div id="app">
2   <mahasiswa
3     v-for="m in mahasiswa"
4     v-bind:nama="m.nama"
5     v-bind:ipk="m.ipk"
6     v-on:parent-info="parentInfo($event)">
7   </mahasiswa>
8 </div>

```

Output:



## 6.4 Component Model

Penggunaan **v-model** untuk keperluan input pada component memiliki mekanisme khusus. Jika kita menambahkan syntax **v-model** seperti berikut:

---

```
1 <custom-input v-model="keywords"></custom-input>
```

---

Maka akan dikonversi menjadi:

---

```
1 <custom-input
2   v-bind:value="keywords"
3   v-on:input="keywords = $event"
4 >
5 </custom-input>
```

---

Maka pada template component, kita dapat memodifikasinya menjadi seperti berikut:

---

```
1 Vue.component('custom-input', {
2   props: ['value'],
3   template: '<input v-bind:value="value"'+
4     ' v-on:input="$emit('input')', $event.target.value"/>'
5 });
```

---

## 6.5 Latihan Mandiri

Buatlah sebuah halaman web untuk melakukan pembelian. User dapat menambahkan barang ke dalam cart melalui tombol yang tersedia. Isi dari cart akan ter-update secara real time apabila jumlah item di dalamnya berubah. Setiap barang memiliki attribute berupa nama dan harga, buatlah minimal 3 data barang dummy.

Output:

Flashdisk: Rp 50000

Buku Tulis: Rp 5000

Pensil: Rp 3000

### Isi Cart:

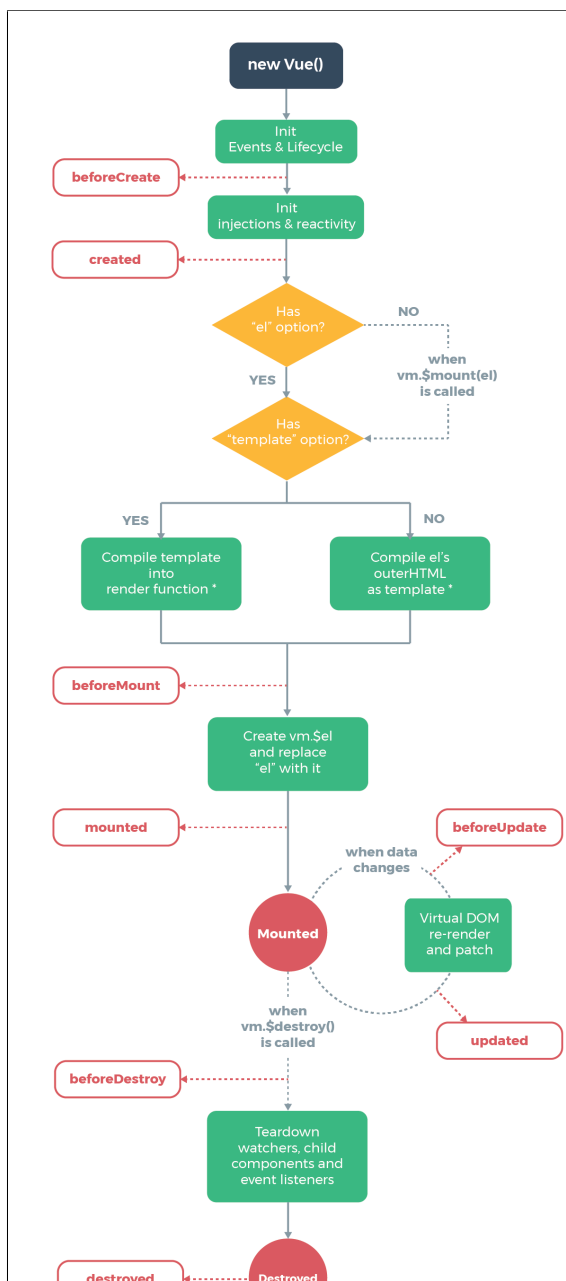
3x Flashdisk = Rp 9000  
 1x Buku Tulis = Rp 3000  
 5x Pensil = Rp 15000  
 Total: Rp 27000

## 6.6 Referensi

<https://vuejs.org/v2/guide/components.html>



## 7. Component Lifecycle Hooks



Saat kita membuat instance dari Vue dengan menggunakan `new Vue()`, maka terdapat beberapa tahap yang akan dilalui seperti yang ditunjukkan pada diagram di bawah.

Di antara setiap fase, kita dapat menyisipkan kode program untuk dieksekusi pada fase-fase tertentu, ditandai dengan blok berwarna merah dari diagram di atas. Terdapat 4 fase utama dari lifecycle Vue yaitu `create`, `mount`, `update`, dan `destroy`.

## 7.1 create

Fase ini adalah fase di mana pertama kali instance dari Vue akan dibuat. Fase ini dibagi menjadi 2 bagian yaitu **beforeCreate** dan **created**. Pada **beforeCreate**, state sama sekali belum diinisialisasi oleh Vue, sedangkan pada **created** data dan event telah selesai diinisialisasi namun belum terpasang pada template. Perhatikan contoh di bawah ini:

```
1 new Vue({
2   el: '#app',
3   data(){
4     return{
5       message: 'Hello World!'
6     }
7   },
8   beforeCreate(){
9     console.log(this.message);
10  },
11  created(){
12    console.log(this.message);
13  }
14 });
```

Karena kedua lifecycle di atas digunakan sebelum template dipasang, sehingga sering diimplementasikan untuk mengambil data dari API dan menyimpannya pada variable (atau state) yang telah disiapkan agar dapat dipergunakan pada tahap berikutnya.

## 7.2 mount

Fase ini adalah fase di mana template pada Vue akan di-render. Fase ini terbagi menjadi 2 bagian, yaitu **beforeMount** yang merupakan sesaat sebelum template di-render dan **mount** yaitu sesaat setelah template di-render. Perhatikan contoh di bawah ini:

```
1 new Vue({
2   el: '#app',
3   template: '<div id="hello">{{message}}</div>',
4   data(){
5     return{
6       message: 'Hello World!'
7     }
8   },
9   beforeMount(){
10    console.log(document.getElementById("hello"));
11  }
```

```
11     },
12     mounted(){
13         console.log(document.getElementById("hello"));
14     }
15 });
```

---

Pada contoh di atas dapat terlihat bahwa pada **beforeMount**, DOM belum dibentuk sama sekali sedangkan pada **mounted** DOM sudah terbentuk. Fase **mounted** dapat digunakan untuk mengakses seluruh komponen yang terdapat pada aplikasi, seolah merupakan kode yang pertama kali dijalankan saat seluruh dokumen telah siap. Fase ini banyak digunakan untuk memodifikasi template atau mengintegrasikan library-library lain ke dalam Vue.

### 7.3 update

Fase ini akan dipanggil apabila template akan di-render ulang setelah sebuah dependency reactive mengalami perubahan. Terdapat 2 fase yaitu **beforeUpdate** yang merupakan sesaat sebelum template di-render ulang dan **updated** yaitu sesaat setelah template di-render ulang. Perhatikan contoh di bawah ini:

```
1 new Vue({
2     el: '#app',
3     template: '<div id="hello">{{message}}'+
4               '<span id="update" v-if="message == \'Updated!\'>Done!</span>'+
5               '<button v-on:click="message=\'Updated!\'>update</button></div>',
6     data(){
7         return{
8             message: 'Hello World!'
9         }
10    },
11    beforeUpdate(){
12        console.log(document.getElementById("update"));
13    },
14    updated(){
15        console.log(document.getElementById("update"));
16    }
17 });
```

---

Fase ini paling tepat digunakan ketika kita ingin mengakses kembali DOM terbaru setelah di-render ulang.

### 7.4 destroy

Fase ini merupakan fase terakhir yang akan dipanggil saat DOM akan dihilangkan. Terdapat 2 fase yaitu **beforeDestroy** yaitu sesaat sebelum DOM dihilangkan, dan **destroyed** yaitu setelah seluruh DOM dihilangkan. Perhatikan contoh kode di bawah ini:

```
1 new Vue({
2     el: '#app',
```



```
3     template: '<div>Hello World!<button v-on:click="destroy()">update</button></div>',
4     data(){
5         return{
6             message: 'Hello World!'
7         }
8     },
9     beforeDestroy(){
10         console.log(this.message);
11     },
12     destroyed(){
13         console.log(this.message);
14     },
15     methods: {
16         destroy(){
17             this.$destroy();
18         }
19     }
20 });
```

---

Saat menekan tombol yang kedua kalinya, tidak terdapat output apapun karena instance Vue sudah dihancurkan.

## 7.5 Latihan Mandiri

Buatlah sebuah aplikasi yang mengimplementasikan seluruh fase (8) yang ada pada penjelasan di atas. Setiap pasang fase (before dan after) harus berisi code yang sama, namun memiliki output yang berbeda untuk menandakan bahwa fase tersebut telah selesai dieksekusi.



## 8. Vue CLI

Vue memiliki command line interface (CLI) khusus dengan perintah **vue** yang bisa dilakukan di terminal/console. Tujuan dari penggunaan Vue CLI adalah agar proses pengelolaan aplikasi Vue menjadi lebih mudah, mulai dari tahap inisialisasi project hingga deployment. Berikut adalah beberapa perintah dasar yang harus dipelajari dengan menggunakan Vue CLI.

### 8.1 **vue create <namaproject>**

Perintah **vue create** digunakan untuk menginisialisasi project. Perintah tersebut harus diikuti dengan nama project yang akan dibangun.

```
1 vue create my_project
```

Perlu diketahui ketika kita membuat project dengan Vue CLI, kita akan membangun aplikasi dengan module system karena project tidak hanya akan melibatkan package Vue saja, namun juga terdapat package webpack untuk mengolah seluruh file komponen (Vue) agar dapat dibaca sebagai HTML, CSS, dan JS serta package Babel agar kode Javascript pada project dapat mendukung seluruh browser termasuk versi-versi lama. Di dalam direktori project, kita akan banyak berurusan pada direktori **src** karena seluruh source code terdapat di sana. Di dalam direktori **src**, akan terdapat 2 file utama yaitu **main.js** dan **App.vue**. Entry point dari aplikasi secara default adalah **main.js**, di mana **main.js** akan memanggil komponen dari **App.vue** sebagai landing page-nya. Oleh karena itu, pengerjaan aplikasi yang akan dibuat dapat dimulai dari file **App.vue**.

### 8.2 **vue serve**

Apabila kita telah memiliki project berbasis Vue, maka untuk menjalankannya kita dapat masuk ke dalam direktori **src** lalu memanggil perintah **vue serve** pada terminal.

```
1 vue serve
```

Perintah tersebut akan memanggil virtual server yang dapat diakses pada web browser melalui alamat **localhost:8080**. Perlu diketahui apabila kita membangun project Vue dengan modular system, maka kita tidak dapat lagi membuka file source code secara langsung pada browser, karena pada dasarnya web browser tidak akan dapat mengenali file dengan ekstensi **.vue**. File tersebut harus diproses dulu oleh package Webpack agar dikonversi menjadi file HTML, CSS, dan JS biasa, sehingga dapat dikenali oleh browser. Selain itu, sebagian besar package menerapkan aturan Same-Origin Policy, di mana seluruh file-file dalam satu project harus berasal dari domain yang sama. Dengan demikian, proses tersebut hanya dapat dilakukan jika kita menggunakan virtual server.

### 8.3 Vue Component

Apabila pada bab sebelumnya kita mempelajari tentang registrasi component dengan cara langsung melalui Class atau instance Vue (baik local maupun global), pada bagian ini kita akan mempelajari bagaimana membuat component pada file terpisah dengan ekstensi **.vue**. Setiap file vue memiliki struktur isi sebagai berikut:

```
1 <template>
2 ...
3 </template>
4
5 <script>
6 ...
7 </script>
8
9 <style lang="scss" scoped>
10 ...
11 </style>
```

Sehingga jika kita memiliki component sebagai berikut:

```
1 Vue.component('hello-component', {
2   props: ['message'],
3   template: '<div>{{message}}</div>'
4 });
```

Maka versi dalam bentuk file vue-nya adalah:

```
1 <template>
2   <div>{{message}}</div>
3 </template>
4
5 <script>
6 export default {
7   name: 'HelloComponent',
8   props: ['message']
9 }
10 </script>
```

```
11
12 <style lang="scss" scoped></style>
```

---

Apabila kita menggunakan module system, maka hampir setiap script akan diawali dengan **export default** yang menandakan bahwa component ini akan di-export agar dapat digunakan oleh file lain dengan cara import.

## 8.4 Import Components

Pada umumnya, jika kita ingin menggunakan Javascript dari file atau library lain, maka kita harus meng-import dengan menggunakan tag `<script>` kemudian mencantumkan nama file yang dikehendaki. Namun dengan menggunakan ES6, kita dapat melakukan import component melalui perintah `import`.

```
1 <script>
2 import HelloComponent from './HelloComponent.vue'
3
4 new Vue({
5   el: '#app',
6   components: {
7     hello: HelloComponent
8   }
9 });
10 </script>
11
12 ...
13
14 <hello></hello>
```

---

Setiap komponen yang diimport harus memiliki alias (pada contoh di atas adalah **MyComponent**) agar memudahkan pemanggilan di script yang bersangkutan. Pada path setelah syntax form biasanya diawali dengan simbol `'./'` yang menandakan direktori src, diikuti dengan lokasi di mana file yang lain berada.

## 8.5 Latihan Mandiri

Buatlah sebuah halaman untuk menampilkan daftar nama orang dengan menggunakan project berbasis module system. Template list untuk nama orang dibuat dalam komponen terpisah dan akan diimport pada file `App.vue`.

Output:

- Adi
- Budi
- Eko
- Fajar
- Tono

## 9. Building Service with Node JS

Single-Page Application (SPA) umumnya menggunakan konsep client-server architecture, dikarenakan proses render data ke dalam view dilakukan oleh client. Dengan demikian, server hanya perlu menyediakan data saja tanpa perlu memproses layout tampilan untuk client. Karena server hanya menyediakan data, maka server dapat digunakan oleh bermacam-macam jenis client lain seperti aplikasi mobile atau desktop.

Server yang hanya menyediakan data saja biasanya disebut sebagai Web Service API. Web Service memiliki beberapa varian, salah satunya yang cukup populer adalah REST. REST cukup populer karena mudah untuk diimplementasikan dan tidak bergantung pada tools tertentu. Pada bab ini kita akan membangun Web Service REST dengan menggunakan NodeJS.

### 9.1 NodeJS

Agar dapat menggunakan NodeJS, pertama-tama install NodeJS terlebih dahulu <https://nodejs.org/>. Kemudian, buat file baru dengan ekstensi .js, untuk endpoint biasanya bernama index.js atau app.js. Kemudian, tambahkan baris program berikut di file tersebut.

```
1 const http = require('http');  
2 const hostname = '127.0.0.1';  
3 const port = 3000;
```

Baris pertama **require('http')** digunakan untuk meng-import library **http**, karena kita akan membangun server dengan protokol **http**. Kemudian **hostname** dan **port** merupakan variable yang digunakan sebagai parameter alamat saat server dijalankan. Kemudian, tambahkan baris berikut ini:

```
1 const server = http.createServer((req, res) => {  
2   res.statusCode = 200;  
3   res.setHeader('Content-Type', 'text/plain');
```

```
4   res.end('Hello World');  
5 });
```

Pada kode di atas, saat ada yang mengirim request ke server, maka server akan mengembalikan pesan **Hello World**. Kemudian di bagian akhir, tambahkan kode program berikut:

```
1 server.listen(port, hostname, () => {  
2   console.log(`Server running at http://${hostname}:${port}/`);  
3 });
```

Kode program di atas berfungsi untuk menjalankan server pada port dan hostname yang telah ditentukan sebelumnya. Secara keseluruhan, kode program menjadi seperti berikut ini:

```
1 const http = require('http');  
2 const hostname = '127.0.0.1';  
3 const port = 3000;  
4  
5 const server = http.createServer((req, res) => {  
6   res.statusCode = 200;  
7   res.setHeader('Content-Type', 'text/plain');  
8   res.end('Hello World');  
9 });  
10  
11 server.listen(port, hostname, () => {  
12   console.log(`Server running at http://${hostname}:${port}/`);  
13 });
```

Untuk menjalankan server, gunakan perintah **node index.js** pada terminal.

## 9.2 Express Framework

Untuk memudahkan membangun server menggunakan NodeJS, kita dapat menggunakan framework Express. Express dapat langsung memisahkan request berdasarkan endpoint dan method-nya masing-masing. Agar dapat menggunakan Express, tambahkan library-nya terlebih dahulu dengan perintah **npm install express**. Kemudian, coba tuliskan kode program berikut:

```
1 const express = require('express')  
2 const app = express()  
3 const port = 3000  
4  
5 app.get('/', (req, res) => {  
6   res.send('Hello World!')  
7 })  
8  
9 app.listen(port, () => {  
10   console.log('Example app listening on port ${port}')11 })
```



Jalankan program di atas dengan perintah **node index.js**, kemudian akses alamatnya pada **localhost:3000**.

Jika diperhatikan, kode program di atas tidak jauh berbeda dengan NodeJS biasa tanpa framework. Hanya saja, pada bagian **createServer** di NodeJS digantikan oleh perintah **get**. Jika pada **createServer** di NodeJS digunakan untuk handle segala jenis request, maka pada Express method **get('/')** hanya akan menerima request yang bertipe **get** dan memiliki entry point **/**. Kita dapat menambahkan entry point lain dengan method atau entry point yang berbeda seperti.

---

```
1 app.get('/', (req, res) => {
2   res.send('Hello World!')
3 })
4
5 app.get('/selamat', (req, res) => {
6   res.send('Selamat Pagi')
7 })
8
9 app.post('/', (req, res) => {
10  res.send('Post: Hello World!')
11 })
12
13 app.put('/', (req, res) => {
14  res.send('Put: Hello World!')
15 })
16
17 app.delete('/', (req, res) => {
18  res.send('Delete: Hello World!')
19 })
```

---

Misalnya pada kode **app.get('/selamat')**, kita dapat mengaksesnya melalui alamat **localhost:3000/selamat**. Untuk perintah yang lain, kita dapat mengaksesnya pada alamat **localhost:3000** kemudian sesuaikan method-nya masing-masing.

## 9.3 Mengakses Data pada Express

Jika kita ingin mengakses data pada body request, kita dapat memanfaatkan parameter **req** yang ada pada setiap perintah. Jika data tersebut terletak pada url, misalnya **localhost:3000/user/nama=adi&umur=17**, maka kita dapat mengakses dengan cara:

---

```
1 app.get('/user', (req, res) => {
2   const nama = req.query.nama;
3   const umur = req.query.umur;
4
5   res.send('nama = '+nama+', umur = '+umur);
6 })
```

---

Namun jika data terletak di dalam body, misalnya pada request dengan method **post**, maka kita dapat mengaksesnya dengan cara:

```
1 app.post('/user', (req, res) => {  
2   const nama = req.body.nama;  
3   const umur = req.body.umur;  
4  
5   res.send('nama = '+nama+', umur = '+umur);  
6 })
```

---

Untuk menggunakan metode di atas, set url ke **localhost:3000/user**, kemudian set body ke **x-www-form-urlencoded**, kemudian masukkan key berupa nama dan umur serta value menyesuaikan key-nya.

## 10. Fetching Data from API

API memiliki peran penting dalam membangun Single-Page Application, karena berisi seluruh data yang dibutuhkan oleh aplikasi. Pengaksesan API pada Vue dapat dilakukan dengan menggunakan library Axios. Sebelum digunakan, kita harus menambahkan library Axios dengan perintah :

```
1 npm install --save axios vue-axios
```

Kemudian meng-import dan menggunakannya dengan cara:

```
1 import Vue from 'vue'
2 import axios from 'axios'
3 import VueAxios from 'vue-axios'
4
5 Vue.use(VueAxios, axios)
```

### 10.1 Get Method

Bentuk paling dasar untuk melakukan request terhadap sebuah API adalah dengan menggunakan get. Get merupakan jenis request yang biasa digunakan untuk mengambil data dari sebuah service.

```
1 Vue.axios
2   .get('https://myapi.com/service?data=value')
3   .then(response => (this.info = response))
```

Pada kode di atas, hasil kembalian dari API akan dapat diakses melalui variable response. Kembalian dari API biasanya memiliki format JSON. Untuk menguji request terhadap API, kita dapat menggunakan situs webhook.site.

Sebagai contoh, kita dapat mengirimkan sebuah request beserta beberapa parameter kepada webhook.site.

```
1 Vue.axios
2   .get('https://webhook.site/<id>?message=helloworld&type=greetings')
3   .then(response => (this.info = response))
```

Kemudian periksa hasilnya melalui URL yang telah disediakan oleh webhook.site.

## 10.2 Post Method

Post merupakan method request yang digunakan untuk mengirimkan data, biasanya bertujuan untuk menyimpan data ke server. Post memiliki bentuk dasar yang serupa dengan get, namun parameter yang ingin disisipkan tidak dituliskan pada URL, melainkan di dalam body request:

```
1 Vue.axios
2   .post('https://myapi.com/service',{
3     data1: value1,
4     data2: value2
5   })
6   .then(response => (this.info = response))
```

Sebagai contoh, kita dapat mengirimkan sebuah request dengan beberapa data kepada webhook.site.

```
1 Vue.axios
2   .post('https://webhook.site/<id>',{
3     message: hello world!,
4     type: greetings
5   })
6   .then(response => (this.info = response))
```

Kemudian periksa hasilnya melalui URL yang telah disediakan oleh webhook.site.

## 10.3 Put Method

Put merupakan method request yang digunakan untuk meng-update data. Put memiliki bentuk yang mirip dengan Post.

```
1 Vue.axios
2   .put('https://myapi.com/service',{
3     data1: value1,
4     data2: value2
5   })
6   .then(response => (this.info = response))
```

Sebagai contoh, kita dapat mengirimkan sebuah request dengan beberapa data kepada webhook.site.

```
1 Vue.axios
2   .put('https://webhook.site/<id>',{
3     message: hello world!,
4     type: greetings
5   })
6   .then(response => (this.info = response))
```

Kemudian periksa hasilnya melalui URL yang telah disediakan oleh webhook.site.

## 10.4 Delete Method

Delete merupakan method request yang digunakan untuk menghapus sebuah data pada server. Delete juga memiliki bentuk yang mirip dengan Post.

```
1 Vue.axios
2   .delete('https://myapi.com/service',{
3     data1: value1,
4     data2: value2
5   })
6   .then(response => (this.info = response))
```

Sebagai contoh, kita dapat mengirimkan sebuah request dengan beberapa data kepada webhook.site.

```
1 Vue.axios
2   .delete('https://webhook.site/<id>',{
3     id: 1
4   })
5   .then(response => (this.info = response))
```

Kemudian periksa hasilnya melalui URL yang telah disediakan oleh webhook.site.

## 10.5 Await Syntax

Sebuah request biasanya dijalankan secara asinkron, artinya berjalan pada thread yang terpisah. Kode program pada baris berikutnya tidak perlu menunggu hingga request selesai dijalankan, oleh karena itu terdapat method then yang akan meng-handle program saat response diterima. Bagaimana jika terdapat suatu kondisi kita hanya akan menjalankan baris program berikutnya saat data telah siap? Untuk mengakomodasi kebutuhan tersebut, kita menggunakan syntax await yang dituliskan sebelum pemanggilan request dilakukan.

```
1 await Vue.axios
2   .get('https://webhook.site/<id>?message=helloworld&type=greetings')
3   .then(response => (this.info = response))
```

## 10.6 Processing Response

Hasil kembalian dari API biasanya memiliki format JSON. Pada Axios, hasil kembalian akan di-convert otomatis ke dalam bentuk object, sehingga kita dapat langsung menggunakannya tanpa perlu melakukan parsing. Sehingga apabila kita memiliki kembalian data seperti contoh berikut:

```
1 {  
2   nama: {  
3     depan: 'Adi',  
4     belakang: 'Nugraha'  
5   },  
6   hobi: 'makan'  
7 }
```

Maka kita dapat mengakses data di atas dengan cara:

```
1 var hasil = 'Nama Lengkap'+response.nama.depan+' '+response.nama.belakang+', Hobi: '+response.hobi
```

## 10.7 Latihan Mandiri

Buatlah sebuah aplikasi ramalan cuaca yang menampilkan kondisi cuaca saat ini sesuai nama kota yang diinputkan oleh user. Secara default, aplikasi akan menampilkan kondisi ramalan untuk kota 'Jakarta'. Seluruh nama kota yang pernah di-request akan tetap dimunculkan sebagai list dan dapat dilihat detilnya apabila di-klik. Gunakan API dari [openweathermap.com](https://openweathermap.org/) untuk mengambil data ramalan cuaca.

## 11. Vue Router

### 11.1 SEO Problems

Seperti yang kita ketahui, membangun sebuah Single-Page Application (SPA) memiliki beberapa kekurangan, salah satunya adalah permasalahan SEO yang kurang optimal. SPA memiliki konsep satu halaman sebagai entry point, kemudian saat berpindah halaman cukup mengganti bagian yang diperlukan saja tanpa harus membuka halaman web yang baru dengan tujuan agar mampu memberikan respon yang cepat terhadap user. Dengan demikian, secara teknis user tidak akan pernah berpindah halaman dan alamat URL tidak pernah berubah. Hal ini menjadikan SPA sulit untuk di-index oleh mesin pencari.

### 11.2 Router

Salah satu solusi yang dapat digunakan untuk mengatasi permasalahan tersebut adalah dengan menggunakan router, yaitu sebuah komponen yang dapat mengatur proses penggantian komponen terkait dengan path pada URL. Dengan demikian, path URL dapat berubah secara dinamis menyesuaikan komponen yang sedang terpasang pada aplikasi.

### 11.3 VueRouter

Vue memiliki router khusus bernama VueRouter yang bertugas untuk melakukan routing atau mapping terhadap view. Untuk menggunakan VueRouter, mula-mula kita harus menambahkan packagenya ke dalam project yang kita bangun:

---

```
1 npm install vue-router
```

---

Kemudian kita dapat meng-import dengan cara berikut:

---

```
1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
```

---

```

3
4 Vue.use(VueRouter)

```

Atau jika tidak menggunakan module system, maka file VueRouter dapat diimport secara langsung:

```

1 <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

```

## 11.4 Router Link & View

Terdapat dua element dasar terkait untuk melakukan routing, yaitu **<router-link>** dan **<router-view>**. Element **<router-link>** akan di-render menjadi element **<a>** dan wajib memiliki attribute **to** yang akan menunjukkan path mana link tersebut mengarah. Saat user meng-klik element **<a>** tersebut, maka komponen yang bersangkutan akan ditampilkan di dalam element **<router-view>**.

```

1 <router-link to="/home">Menuju Home</router-link>
2 <router-link to="/about">Menuju About</router-link>
3
4 <router-view></router-view>

```

Pada contoh di atas, apabila link 'Menuju Home' di-klik, maka aplikasi akan me-render component yang diasosiasikan dengan path '/home' ke dalam element **<router-view>**. Untuk menentukan component yang akan di-render, kita dapat melakukan konfigurasi router melalui kode program berikut:

```

1 const routes = [
2   { path: '/home', component: HomeComponent },
3   { path: '/about', component: AboutComponent }
4 ];
5
6 const router = new VueRouter({
7   routes
8 });
9
10 new Vue({
11   router
12 }).$mount('#app')

```

Untuk mencoba menggunakan router, mula-mula buatlah tiga buah component di dalam direktori **components** dengan isi yang berbeda seperti contoh berikut:

```

1 <template>
2   <div>
3     Ini halaman satu
4   </div>
5 </template>
6

```



```
7 <script>
8   export default {
9     name: 'Satu'
10  }
11 </script>
```

---

Buatlah juga untuk component bernama Dua dan Tiga (isi menyesuaikan).

Kemudian di dalam App.vue yang berperan sebagai entry page, kita akan memasukkan element **router-link** dan **router-view** di dalamnya.

```
1 <template>
2   <div id="app">
3     <router-link to="/">Satu</router-link> |
4     <router-link to="/dua">Dua</router-link> |
5     <router-link to="/tiga">Tiga</router-link>
6
7     <router-view />
8   </div>
9 </template>
```

---

Kemudian untuk langkah terakhir, kita harus mendefinisikan komponen untuk masing-masing path. Terdapat tiga buah path yaitu '/' sebagai landing page, sedangkan '/dua' dan '/tiga' merupakan link untuk menuju ke masing-masing halaman. Buka file router/index.js, kemudian masukkan kode seperti berikut:

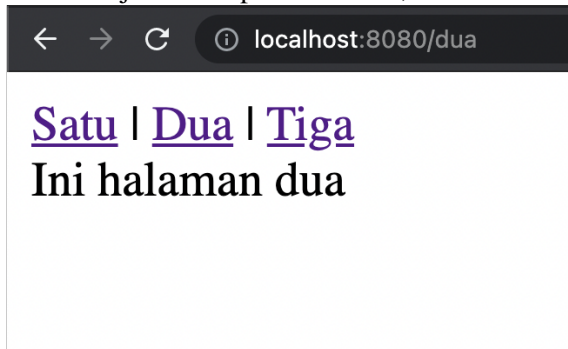
```
1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
3 import Satu from '../components/Satu.vue'
4 import Dua from '../components/Dua.vue'
5 import Tiga from '../components/Tiga.vue'
6
7 Vue.use(VueRouter)
8
9 const routes = [
10   {
11     path: '/',
12     name: 'Satu',
13     component: Satu
14   },
15   {
16     path: '/dua',
17     name: 'Dua',
18     component: Dua
19   },
20   {
21     path: '/tiga',
22     name: 'Tiga',
23     component: Tiga
```

```

24   },
25 ]
26
27 const router = new VueRouter({
28   mode: 'history',
29   base: process.env.BASE_URL,
30   routes
31 })
32
33 export default router

```

Coba jalankan aplikasi di atas, kemudian amati perubahan pada URL-nya. Output:



## 11.5 Programming Route

Selain menggunakan link, kita juga dapat mengatur routing secara programming. Misalnya ketika kita ingin mundur ke halaman sebelumnya, kita dapat melakukannya dengan cara memanggil perintah **this.\$router.go(-1)**. Untuk mencobanya, kita akan membuat sebuah button back pada template di App.vue.

```

1 <button v-on:click="back()">Back</button>

```

---

```

1 export default({
2   methods:{
3     back(){
4       this.$router.go(-1);
5     }
6   }
7 })

```

Selain itu, kita juga dapat mengarahkan aplikasi ke halaman tertentu dengan perintah **this.\$router.push('path')**. Misalnya jika kita ingin membuat sebuah tombol yang akan selalu mengarah ke landing page:

```

1 <button v-on:click="home()">Go to Home</button>

```

---

```

1 export default({
2   methods:{

```

---

```

3     home(){
4         this.$router.push('/');
5     }
6 }
7 })

```

---

## 11.6 Dynamic Route

Selain menggunakan routing statis, kita juga dapat melakukan routing dinamis, yaitu routing dengan mengirimkan parameter. Cara ini akan cukup berguna untuk situasi di mana kita akan menampilkan master/detail data. Misalnya kita memiliki array of object yang berisi nama orang beserta id-nya pada App.vue, kemudian kita ingin membuat mekanisme master dan detail data dari array tersebut. Mula-mula tambahkan routing baru pada router yang dapat menerima parameter. Parameter pada routing ditandai dengan simbol titik dua (:) pada path.

---

```

1 {
2     path: '/penduduk/:id/:nama',
3     name: 'Penduduk',
4     component: Penduduk
5 }

```

---

Buat juga komponen untuk menampilkan hasil dari route tersebut dengan nama Penduduk.vue

---

```

1 <template>
2     <div>
3         Id: {{$route.params.id}}, Nama: {{$route.params.nama}}
4     </div>
5 </template>
6
7 <script>
8     export default {
9         name: 'Penduduk',
10        props: ['id', 'nama']
11    }
12 </script>

```

---

Pada kode di atas dapat terlihat bahwa route menerima dua buah parameter yaitu id dan nama, ditandai dengan tanda titik dua di depannya. Kemudian pada App.vue tambahkan beberapa data dummy, serta tambahkan sebuah method yang dapat dipanggil menuju route yang bersangkutan.

---

```

1 export default({
2     data(){
3         return{
4             listNama: [
5                 {id: 1, nama: 'Adi Nugraha'},
6                 {id: 3, nama: 'Bambang Pamungkas'},
7                 {id: 10, nama: 'Kunto Aji'},
8                 {id: 12, nama: 'Ariel Noah'},

```

```

9      {id: 5, nama: 'Charlie Van Houten'},
10    ]
11  },
12  },
13  methods:{
14    go(id, nama){
15      this.$router.push('/penduduk/'+id+'/'+nama);
16    }
17  }
18 })

```

Method `go()` nantinya akan di-trigger oleh sebuah button seperti pada contoh berikut:

```

1 <div v-for="item in listNama" :key="item.id" v-on:click="go(item.id, item.nama)">{{item.id}}

```

Jalankan aplikasi, kemudian apabila salah satu id di-klik, maka data detail akan ditampilkan seperti contoh di bawah ini: Output:

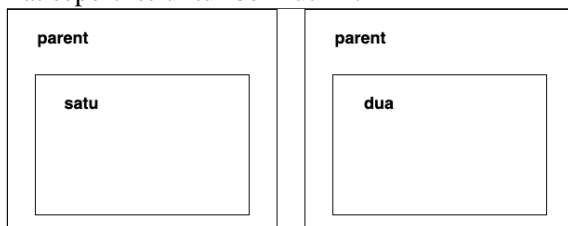
```

1
3
10
12
5
Id: 3, Nama: Bambang Pamungkas

```

## 11.7 Nested Route

Selain dapat mengirimkan parameter, routing pada VueRouter juga dapat dibuat secara bertingkat seperti struktur berikut ini:



Maka secara routing, kita dapat membuatnya seperti berikut:

```

1 {
2   path: '/parent',
3   name: 'Parent',
4   component: Parent,
5   children: [
6     {
7       path: 'satu',
8       component: Satu
9     },
10    {

```

```

11         path: 'dua',
12         component: Dua
13     }
14 ]
15 }

```

Pada template component Parent harus terdapat element `<router-view>` agar dapat menampilkan component milik children yang dimiliki seperti contoh berikut:

```

1 <template>
2   <div>
3     Ini Parent
4     <router-view/>
5   </div>
6 </template>
7
8 <script>
9   export default {
10     name: 'Parent'
11   }
12 </script>

```

Kemudian pada App.vue, kita dapat menambahkan `<router-link>` seperti berikut:

```

1 <router-link to="/parent">Parent</router-link> |
2 <router-link to="/parent/satu">Parent - Satu</router-link> |
3 <router-link to="/parent/dua">Parent - Dua</router-link>
4 <router-view/>

```

Output /parent:

[Parent](#) | [Parent - Satu](#) | [Parent - Dua](#)  
Ini Parent

Output /parent/satu:

[Parent](#) | [Parent - Satu](#) | [Parent - Dua](#)  
Ini Parent  
Ini halaman satu

Output /parent/dua:

[Parent](#) | [Parent - Satu](#) | [Parent - Dua](#)  
Ini Parent  
Ini halaman dua

## 11.8 Latihan Mandiri

Buatlah sebuah SPA tentang data penduduk yang terdiri dari dua buah halaman, pada halaman pertama user dapat menambahkan data penduduk yang terdiri dari nama dan usia, sedangkan pada halaman kedua akan terdapat satu buah input text di mana user dapat mencari nama penduduk berdasarkan keyword yang dimasukkan.

Output:

[Add Penduduk](#) | [Search Penduduk](#) |

Nama	Usia	Tambahkan
------	------	-----------

Output:

[Add Penduduk](#) | [Search Penduduk](#) |

adi 17

bambang 19

## 12. Vuex

Vuex merupakan state management library yang dapat digunakan untuk menyimpan state atau kondisi dari sebuah aplikasi. Karena aplikasi SPA merupakan aplikasi berbasis modul, maka modul dapat muncul dan hilang kapanpun, sementara itu di dalam modul terdapat data-data yang harus disimpan. State management bertugas untuk mengelola seluruh data agar dapat disimpan secara terpusat, sehingga dapat digunakan oleh modul apapun.

### 12.1 Instalasi

Untuk menggunakan Vuex, kita dapat menambahkan package dengan cara:

```
1 npm install vuex --save
```

Kemudian kita dapat meng-import dengan cara:

```
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3
4 Vue.use(Vuex)
```

Kita juga dapat menambahkan file Javascript secara langsung dengan cara:

```
1 <script src="https://unpkg.com/vuex"></script>
```

### 12.2 Struktur Dasar Vuex

Sebuah instance pada Vuex disebut dengan store. Sebagai contoh, kita dapat membuat sebuah instance sederhana seperti contoh berikut:

```
1 const store = new Vuex.Store({
2   state: {
3     count: 0
4   },
5   mutations: {
6     increment (state) {
7       state.count++
8     }
9   }
10 })
```

Pada contoh di atas, terdapat beberapa attribute dalam pembuatan store yaitu **state** dan **mutations**. **State** berisi deklarasi variable yang akan dikelola oleh Vuex, sedangkan **mutations** berisi method atau function yang dapat memanipulasi variable pada **state**. Kemudian, apabila kita ingin memanggil method `increment` yang terdapat pada store, kita dapat memanggilnya dengan method `commit` seperti berikut:

```
1 store.commit('increment');
```

Untuk mengakses variable di dalam store, kita dapat melakukannya dengan cara:

```
1 console.log(store.state.count)
```

Karena instance dari Vuex disimpan ke dalam sebuah variable bernama `store`, maka kita dapat memasukkan variable tersebut ke dalam instance vue agar dapat digunakan di semua modul dengan cara:

```
1 new Vue({
2   el: '#app',
3   store: store
4 })
```

Di dalam instance vue, kita dapat memanggil variable store dengan menambahkan tanda `$` di depan kata store seperti: **`this.$store`**. Dengan cara di atas, seluruh komponen yang menjadi child dari instance vue juga akan dapat mengakses variable `$store`.

## 12.3 Mutations

Sebuah instance store dapat memiliki **mutations**, di mana di dalamnya akan terdapat deklarasi function terkait dengan **state**. Pada contoh di atas, kita mengetahui bahwa function pada **mutations** harus menerima parameter berupa **state**. Namun selain itu, function juga dapat menerima parameter tambahan seperti contoh berikut:

```
1 mutations: {
2   increment (state, n) {
3     state.count += n
4   }
5 }
```



Kemudian function **increment()** dapat dipanggil dengan cara:

```
1 store.commit('increment', 10)
```

## 12.4 Actions

Selain **mutations**, instance store juga dapat memiliki attribute bernama **actions**. Pada dasarnya **actions** serupa dengan **mutations**, namun **mutations** hanya dapat dilakukan untuk proses yang bersifat sinkronus, sedangkan **actions** dapat dilakukan secara asinkronus. Attribute **actions** tidak memanipulasi **state** secara langsung, melainkan memanggil function yang terdapat pada **mutations**.

```
1 store = new Vuex.Store({
2   state: {
3     count: 0
4   },
5   mutations: {
6     increment (state) {
7       state.count++
8     }
9   },
10  actions: {
11    increment (context) {
12      context.commit('increment')
13    }
14  }
15 })
```

Function pada actions di trigger dengan menggunakan perintah `dispatch()`:

```
1 store.dispatch('increment');
```

## 12.5 Module

Agar dapat digunakan secara lebih ringkas, Vuex dapat dibuat menjadi sebuah modul dengan template seperti berikut ini:

```
1 const moduleA = {
2   state: () => ({ ... }),
3   mutations: { ... },
4   actions: { ... },
5   getters: { ... }
6 }
7
8 const moduleB = {
9   state: () => ({ ... }),
10  mutations: { ... },
11  actions: { ... },
```

```
12   getters: { ... }  
13 }
```

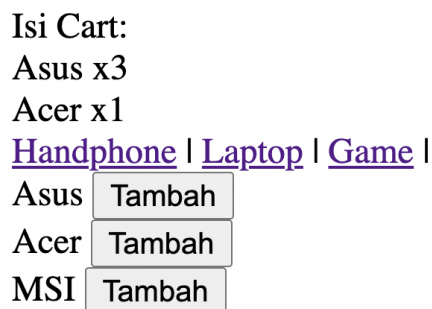
Setelah membuat dalam bentuk module, module tersebut harus dimasukkan ke dalam instance store seperti berikut:

```
1  const store = new Vuex.Store({  
2    modules: {  
3      a: moduleA,  
4      b: moduleB  
5    }  
6  });  
7  
8  store.state.a; //mengakses state modul a  
9  store.state.b; //mengakses state modul b
```

## 12.6 Latihan Mandiri

Buatlah sebuah aplikasi SPA tentang marketplace yang terdiri dari 3 halaman, di mana masing-masing halaman akan menampilkan daftar item yang berbeda. Pada bagian header aplikasi, akan terdapat keranjang belanja yang berisi daftar item apa saja yang telah ditambahkan oleh user ke dalamnya.

Output:



Isi Cart:  
Asus x3  
Acer x1  
[Handphone](#) | [Laptop](#) | [Game](#) |  
Asus   
Acer   
MSI

Output:

Isi Cart:

Asus x3

Acer x1

Xbox One x3

Samsung Galaxy S30 x1

IPhone 15 x2

[Handphone](#) | [Laptop](#) | [Game](#) |

IPhone 15

Samsung Galaxy S30



## 13. Vuefire

### 13.1 Firebase

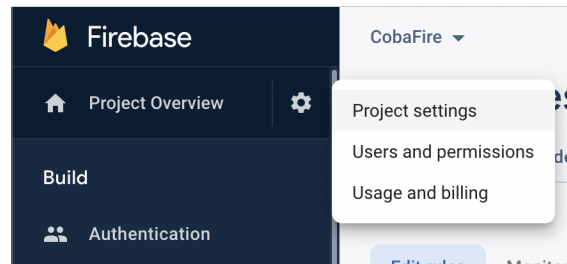
Firebase merupakan salah satu layanan berbasis awan (cloud) yang dapat digunakan sebagai backend dari sebuah aplikasi, atau disebut juga sebagai Backend as a Service (BaaS). Dengan memanfaatkan Firebase, maka kita dapat langsung memiliki sebuah backend tanpa harus membuat dari nol. Layanan yang terdapat pada Firebase meliputi otentikasi, database, storage, machine learning, dan lain-lain.

### 13.2 Firestore

Firestore merupakan salah satu produk dari Firebase yang dapat digunakan sebagai database. Sebetulnya, selain Firestore terdapat database lain bernama Realtime Database. Namun Realtime Database lebih berfokus pada data yang ingin digunakan secara realtime, sedangkan Firestore berfokus pada penyimpanan data yang kompleks dan memiliki scalability yang baik.

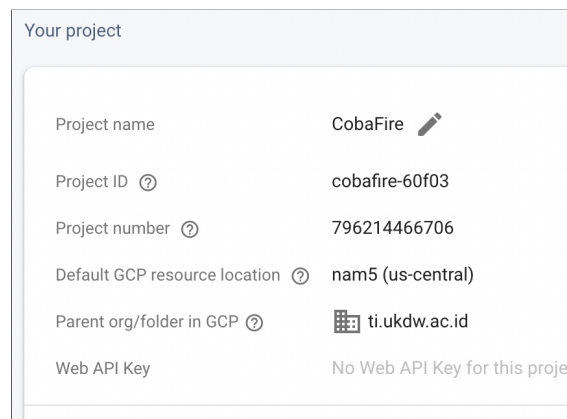
### 13.3 Instalasi

Pertama-tama, kunjungi halaman <https://console.firebase.google.com/> kemudian lakukan login. Setelah login, buat sebuah project baru, gunakan pengaturan default jika bingung. Setelah masuk ke halaman dashboard, tekan tombol dengan simbol gerigi, kemudian pilih menu Project Settings.



Gambar 13.1: Menu Project Settings

Di dalam menu Project Settings akan terdapat Project ID yang akan kita perlukan saat konfigurasi.



Gambar 13.2: Project ID

Apabila ingin menggunakan Firestore, maka buka menu Firestore kemudian tekan tombol Create Database, lalu pilih mode test.

Langkah berikutnya, buat sebuah project Vue, kemudian di dalam folder project tambahkan library Firebase dengan menggunakan perintah:

---

```
1 vue add vuefire
```

---

Di dalam folder src akan terdapat file baru bernama db.js, di dalamnya akan berisi konfigurasi Firestore yang akan kita gunakan. Pada bagian atribut projectId, masukkan Project ID dari Firebase yang telah kita buat sebelumnya:

---

```
1 import firebase from 'firebase/compat/app'
2 import 'firebase/compat/firestore'
3
4 export const db = firebase
5   .initializeApp({ projectId: 'PROJECT_ID' })
6   .firestore()
```

---

Catatan: pada bagian import 'firebase/app', jika project tidak support, maka kita dapat menggunakan tambahan kata 'compat' setelah direktori 'firebase' pada path seperti 'firebase/compat/app'.

Firestore memiliki konsep database sebagai sekumpulan dokumen, sehingga terdapat dua buah istilah yang harus dipahami yaitu **collection** dan **document**. Collection adalah sekumpulan document, sedangkan document itu sendiri merupakan satu entitas data. Apabila dibandingkan dengan database relational, maka collection dapat dianggap sebagai sebuah table dan document adalah record di dalam table tersebut.

### 13.4 Add Data

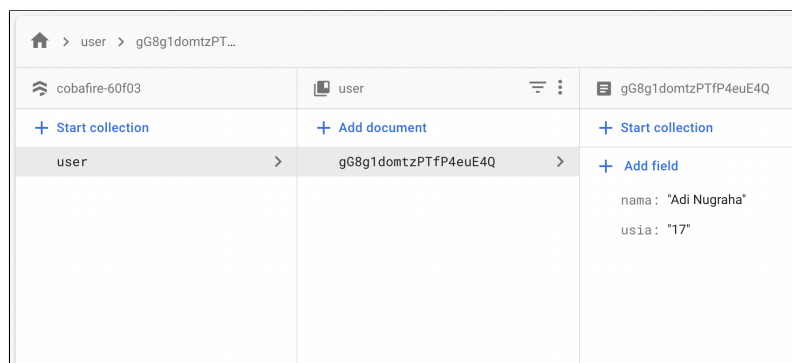
Untuk menambah data pada Firestore, kita dapat meng-import database yang berasal dari file db.js, kemudian memanggil fungsi-fungsi di dalamnya. Berikut adalah cara meng-import database dari file db.js.

```
1 import {db} from './db'
```

Jika ingin menambahkan data pada collection **user** dengan atribut **nama** dan **usia**, maka dapat dilakukan dengan cara berikut ini:

```
1 db.collection('user').add({nama: 'Adi Nugraha', usia: 17});
```

Apabila kita cek pada menu Firestore dalam Firebase, maka akan terdapat data tersebut:



Gambar 13.3: Database Firestore

Jika diperhatikan, setiap document akan memiliki id yang bersifat unik. Dengan menggunakan cara di atas, id akan diberikan dengan metode hash di mana id akan memiliki karakter acak. Apabila kita ingin menambah data dengan id yang kita tentukan sendiri, maka dapat dilakukan dengan cara berikut ini:

```
1 db.collection('user').doc('123').set({nama: 'Adi Nugraha', usia: 17});
```

Cara di atas juga dapat digunakan untuk meng-update data.

### 13.5 Get Data

Untuk mengambil data dari Firestore, kita dapat menggunakan fungsi `get()`. Apabila kita hendak mengambil data dari sebuah collection, maka dapat dilakukan dengan cara berikut:

```
1 db.collection('user')
2   .get()
3   .then(querySnapshot => {
4     const documents = querySnapshot.docs.map(doc => doc.data())
5   })
```

---

Namun jika kita ingin mengambil data dari sebuah document, maka dapat dilakukan dengan cara:

```
1 db.collection('documents')
2   .doc(documentId)
3   .get()
4   .then(snapshot => {
5     const document = snapshot.data()
6   })
```

---

### 13.6 Delete Data

Jika ingin menghapus data, maka kita dapat menggunakan fungsi `delete()` seperti pada contoh berikut ini:

```
1 db.collection('user').doc('123').delete();
```

---

### 13.7 Data Binding

Pada Vue, kita dapat melakukan binding antara sebuah variable lokal dengan data pada Firestore, dengan demikian apabila terdapat perubahan nilai pada data tersebut di Firestore, maka variable lokal yang terikat akan ikut berubah seketika. Proses binding dapat dilakukan dengan menambahkan atribut `firestore` pada instance Vue seperti contoh berikut:

```
1 export default {
2   data() {
3     return {
4       user: [],
5     }
6   },
7
8   firestore: {
9     user: db.collection('user'),
10   },
11 }
```

---



## 14. SCSS

### 14.1 CSS

CSS merupakan komponen pada sebuah website yang digunakan untuk mengatur bagaimana dokumen HTML akan ditampilkan. Salah satu kekurangan dari teknologi CSS saat ini adalah CSS tidak dapat memproses kode-kode yang bersifat logika, sehingga akan banyak sekali kode program yang redundan. Salah satu contohnya jika sebuah website hanya memiliki satu warna utama, dan akan digunakan di berbagai macam komponen, maka kode programnya adalah sebagai berikut:

```
1 p{  
2   color: #E32088  
3 }  
4  
5 div{  
6   background-color: #E32088;  
7 }
```

Jika suatu saat terdapat perubahan warna pada atribut CSS, maka kita harus mengubah kode sebanyak warna yang telah dituliskan sebelumnya. Pada contoh di atas, jika kita ingin mengubah warna dari #E32088 menjadi #FA161E, maka kita harus mengubah dua baris kode pada CSS:

```
1 p{  
2   color: #FA161E  
3 }  
4  
5 div{  
6   background-color: #FA161E;  
7 }
```

Bayangkan jika ternyata tidak hanya dua baris kode yang harus diubah, melainkan terdapat puluhan atau ratusan baris atribut warna yang harus diubah.

## 14.2 SCSS

SCSS merupakan singkatan dari Sassy CSS, adalah syntax yang memungkinkan sebuah CSS dibangun menggunakan proses logika seperti pada bahasa pemrograman pada umumnya. SCSS nantinya akan dicompile menggunakan Sass (Syntactically Awesome Style Sheet) dengan output akhir berupa file CSS standar. Karena dapat memproses logika di dalamnya, maka di dalamnya kita dapat membuat variable, melakukan import, dan lain sebagainya. Misalnya pada contoh sebelumnya, jika website kita hanya memiliki satu warna utama, maka kita dapat menuliskan kode SCSS-nya sebagai berikut:

```
1 $color: #E32088
2
3 p{
4     color: $color
5 }
6
7 div{
8     background-color: $color;
9 }
```

Pada kode program di atas, jika kita ingin mengganti warna utama dari website, maka kita cukup mengubah satu baris kode saja yaitu di bagian deklarasi \$color paling atas.

## 14.3 SCSS pada Vue

Untuk mengimplementasikan SCSS pada Vue, pertama-tama kita harus menambahkan beberapa modules sebagai berikut:

```
1 npm install sass sass-loader@10 webpack webpack-cli webpack-dev-server
```

Module sass dan sass-loader digunakan untuk memproses syntax SCSS. Karena kode SCSS harus di-compile terlebih dahulu menjadi CSS sebelum digunakan, maka kita membutuhkan preprocessor, di sini module webpack dibutuhkan.

Setelah module-module tersebut berhasil di-install, langkah berikutnya adalah membuat file webpack.config.js pada direktori root dengan isi sebagai berikut:

```
1 const path = require('path')
2
3 module.exports = {
4     entry: './src/main.js',
5     output: {
6         path: path.resolve(__dirname, './dist'),
7         filename: 'bundle.js'
8     },
9     devServer: {
```

---

```

10     contentBase: path.resolve(__dirname, 'public')
11   },
12   module: {
13     rules: [
14       {
15         test: /\.scss$/,
16         use: [
17           'vue-style-loader',
18           'css-loader',
19           'sass-loader'
20         ]
21       }
22     ]
23   },
24 }

```

---

Dengan demikian, kita dapat menggunakan syntax SCSS pada file Vue dengan cara seperti berikut:

---

```

1 <style lang="scss">
2 $primary: red;
3
4 #app {
5   color: $primary;
6 }
7 </style>

```

---

## 14.4 Variable

Seperti pada penjelasan sebelumnya, pada SCSS kita dapat membuat variable dengan cara menggunakan simbol \$ (dollar) diikuti dengan nama variable-nya.

---

```

1 $myColor: red;
2 $myFont: Helvetica, sans-serif;
3 $mySize: 18px;
4
5 div{
6   color: $myColor;
7   font-family: $myFont;
8   font-size: $mySize;
9 }

```

---

Tipe data yang dapat digunakan pada variable SCSS adalah strings, numbers, colors, booleans, lists, dan nulls.

Pada SCSS juga berlaku sistem variable global dan local (scope). Kita dapat mendefinisikan dua variable dengan nama yang sama, namun berbeda scope. Variable dengan scope lebih dalam tidak dapat diakses oleh scope yang berada di luar:

```
1 $myColor: red;
2
3 h1{
4     $myColor: blue;
5     color: $myColor;
6 }
7
8 div{
9     color: $myColor;
10 }
```

Pada kode di atas, konten di dalam h1 akan memiliki warna blue, sedangkan konten di dalam div akan memiliki warna red. Kecuali jika kita menambahkan tag !global di akhir variable, maka variable tersebut akan menjadi variable global. Pada contoh di bawah, kode seluruh elemen akan memiliki warna blue.

```
1 $myColor: red;
2
3 h1{
4     $myColor: blue !global;
5     color: $myColor;
6 }
7
8 div{
9     color: $myColor;
10 }
```

## 14.5 Nested

Pada SCSS kita dapat membuat syntax bertingkat, sehingga kita dapat memisahkan mana kode yang akan diimplementasikan di seluruh selector, dan mana kode yang hanya terdapat di selector tertentu.

```
1 div{
2     font-size: 18pt;
3
4     .first{
5         color: red;
6     }
7
8     .second{
9         color: blue
10    }
11 }
```

Kode SCSS di atas setara dengan kode CSS berikut:

---

```
1  div .first{
2      font-size: 18pt;
3      color: red;
4  }
5
6  div .second{
7      font-size: 18pt;
8      color: blue;
9  }
```

---

Selain selector, property juga dapat dibuat secara nested seperti contoh berikut:

---

```
1  font: {
2      family: Helvetica, sans-serif;
3      size: 18px;
4      weight: bold;
5  }
6
7  text: {
8      align: center;
9      transform: lowercase;
10     overflow: hidden;
11 }
```

---

Kode program di atas setara dengan kode CSS berikut:

---

```
1  font-family: Helvetica, sans-serif;
2  font-size: 18px;
3  font-weight: bold;
4
5  text-align: center;
6  text-transform: lowercase;
7  text-overflow: hidden;
```

---

## 14.6 Include

Pada SCSS, kita dapat membuat satu set syntax, kemudian menggunakannya di banyak selector. Untuk mendeklarasikan set syntax, kita dapat menggunakan syntax `@mixin` dan mengimplementasikannya dengan menggunakan `@include` seperti contoh berikut:

---

```
1  @mixin warning{
2      color: red;
3      font-weight: bold;
4      font-size: 32pt;
5  }
6
7  div{
```

---

```

8     @include warning;
9     font-family: Arial;
10  }
11
12  span{
13     @include warning;
14     font-family: Helvetica;
15  }

```

---

Kode program di atas setara dengan kode CSS berikut:

---

```

1  div{
2     color: red;
3     font-weight: bold;
4     font-size: 32pt;
5     font-family: Arial;
6  }
7
8  span{
9     color: red;
10    font-weight: bold;
11    font-size: 32pt;
12    font-family: Helvetica;
13  }

```

---

@mixin juga dapat bertindak seperti sebuah function yang dapat menerima parameter apabila dibutuhkan property yang berbeda. Kode program SCSS sebelumnya akan lebih efisien jika diubah menjadi seperti berikut:

---

```

1  @mixin warning($font){
2     color: red;
3     font-weight: bold;
4     font-size: 32pt;
5     font-family: $font;
6  }
7
8  div{
9     @include warning(Arial);
10  }
11
12  span{
13     @include warning(Helvetica);
14  }

```

---

Parameter tersebut juga dapat memiliki default value apabila saat pemanggilan tidak mengirimkan parameter:

---

```

1  @mixin warning($font: sans-serif){
2     color: red;

```

---

```

3     font-weight: bold;
4     font-size: 32pt;
5     font-family: $font;
6 }
7
8 div{
9     @include warning(Arial);
10 }
11
12 span{
13     @include warning(Helvetica);
14 }
15
16 h1{
17     @include warning;
18 }

```

---

## 14.7 Extend

Pada SCSS, kita dapat menggunakan property yang dimiliki oleh selector lain agar dicopy ke dalam sebuah selector dengan menggunakan syntax `@extend` seperti pada contoh berikut:

---

```

1 .base{
2     color: black;
3     font-size: 12pt;
4     font-family: Arial;
5 }
6
7 .first{
8     @extend .base;
9     background-color: blue;
10 }
11
12 .second{
13     @extend .base;
14     background-color: red;
15 }

```

---

Kode program di atas setara dengan kode CSS berikut:

---

```

1 .base, .first, .second{
2     color: black;
3     font-size: 12pt;
4     font-family: Arial;
5 }
6
7 .first{
8     background-color: blue;

```

```
9   }  
10  
11  .second{  
12      background-color: red;  
13  }
```

---



## 15. Vuetify

### 15.1 Material Design

Material Design adalah panduan desain (design guidelines) yang diterbitkan oleh Google pertama kali tahun 2014. Material Design diterapkan pada seluruh platform milik Google, mulai dari website, mobile, hingga di dalam sistem operasi Android. Apabila terdapat developer yang ingin mengembangkan aplikasi dengan menggunakan Material Design, maka panduan tersebut dapat diakses melalui <https://material.io/>.

### 15.2 Vuetify

Vuetify merupakan library Vue yang berfungsi untuk menerapkan Material Design ke dalam project Vue. Dengan menggunakan Vuetify, maka kita dapat membangun aplikasi berbasis Vue menggunakan tampilan Material Design dengan mudah dan cepat. Untuk menggunakan Vuetify, pertama-tama kita dapat menambahkan Vuetify ke dalam project Vue dengan cara:

---

```
1 vue add vuetify
```

---

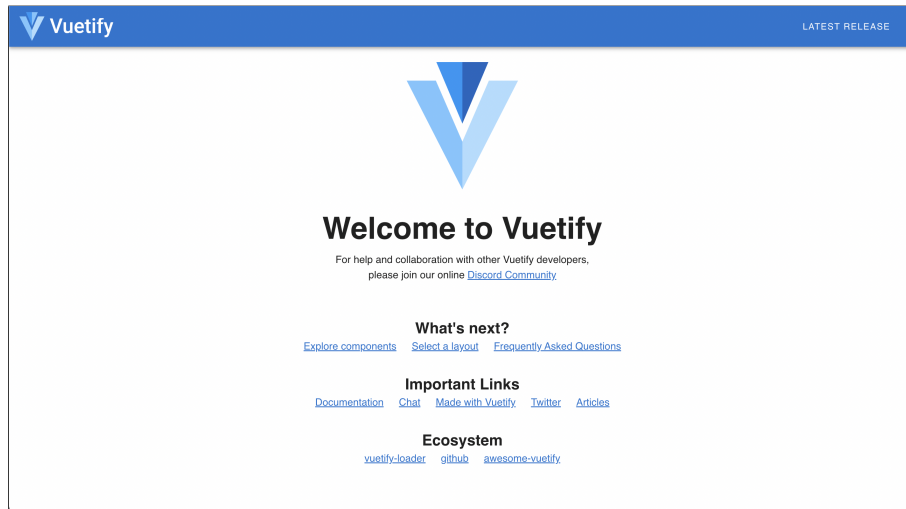
Jika tidak ada preferensi khusus, pilih settingan default / recommended. Kemudian, lakukan pengecekan pada file `src/plugins/vuetify.js`, pastikan isinya seperti berikut:

---

```
1 import Vue from 'vue';
2 import Vuetify from 'vuetify'
3 import 'vuetify/dist/vuetify.min.css'
4
5 Vue.use(Vuetify);
6
7 export default new Vuetify({});
```

---

Jika sesuai, maka tampilan halaman defaultnya adalah sebagai berikut:



Gambar 15.1: Tampilan Default Vuetify

Pada Vuetify, root aplikasi dapat menggunakan `<v-app>` untuk menggantikan `<div id="app">` yang biasa digunakan sebelumnya.

```
1 <template>
2   <v-app>
3     <!-- content -->
4   </v-app>
5 </template>
```

### 15.3 App Bar

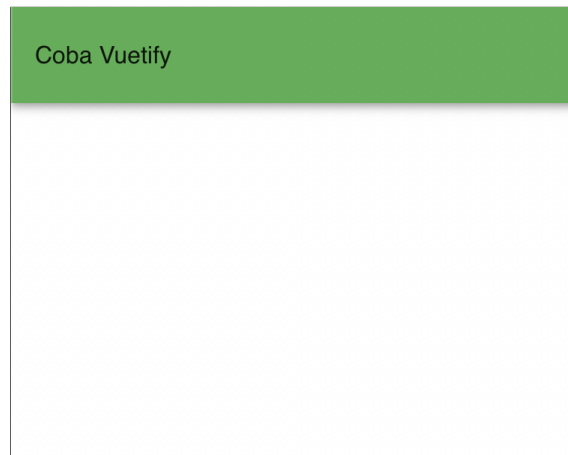
Pada Material Design, biasanya terdapat navigasi di bagian atas layar yang disebut sebagai App Bar. Untuk membuat AppBar, kita dapat menggunakan syntax berikut:

```
1 <v-app-bar app>Coba Vuetify</v-app-bar>
```

Jika ingin melakukan modifikasi seperti menambahkan shadow atau mengganti warna, kita dapat menambahkan atribut berikut:

```
1 <v-app-bar
2   app
3   color="green"
4   elevation="4"
5   >
6   Coba Vuetify
7 </v-app-bar>
```

Maka tampilan akhir akan menjadi seperti berikut:



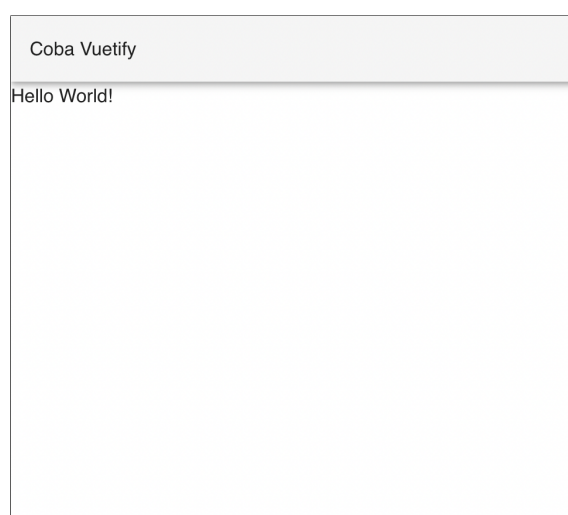
Gambar 15.2: AppBar Custom

## 15.4 Layout

Pada Vuetify, seluruh konten dari website harus dimasukkan ke dalam element `<v-main>` sebagai root. Namun karena App Bar tidak termasuk konten dari website, maka element `<v-app-bar>` tetap diletakkan di luar `<v-main>`

```
1 <v-app>
2   <v-app-bar app>Coba Vuetify</v-app-bar>
3   <v-main>
4     Hello World!
5   </v-main>
6 </v-app>
```

Maka tampilan akhir akan menjadi seperti berikut:

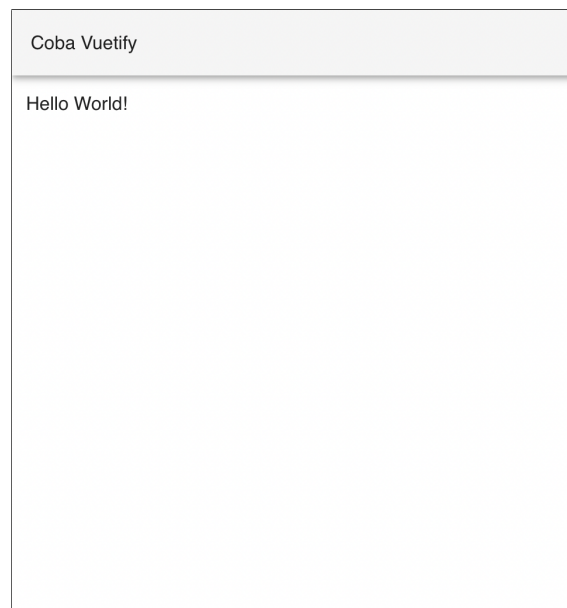


Gambar 15.3: Tampilan Main

Agar sedikit lebih rapi, maka tambahkan `<v-container>` di dalam `<v-main>`:

```
1 <v-app>
2   <v-app-bar app>Coba Vuetify</v-app-bar>
3   <v-main>
4     <v-container>
5       Hello World!
6     </v-container>
7   </v-main>
8 </v-app>
```

Maka tampilan akan menjadi seperti berikut:



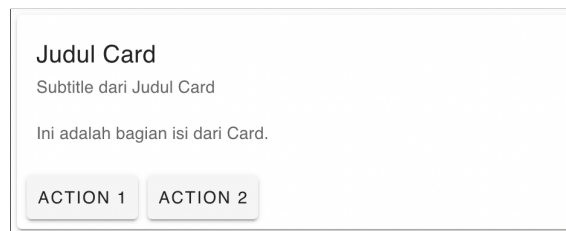
Gambar 15.4: Tampilan Container

## 15.5 Card

Card merupakan container yang banyak digunakan untuk membungkus konten. Card dapat digunakan dengan menggunakan elemen `<v-card>`, kemudian di dalamnya terdapat 4 sub-komponen yang dapat didefinisikan: `v-card-title`, `v-card-subtitle`, `v-card-text` dan `v-card-actions`.

```
1 <v-card
2   elevation="2"
3   >
4     <v-card-title>Judul Card</v-card-title>
5     <v-card-subtitle>Subtitle dari Judul Card</v-card-subtitle>
6     <v-card-text>Ini adalah bagian isi dari Card.</v-card-text>
7     <v-card-actions>
8       <v-btn>Action 1</v-btn>
9       <v-btn>Action 2</v-btn>
10    </v-card-actions>
11  </v-card>
```

Maka tampilan akan menjadi seperti berikut:



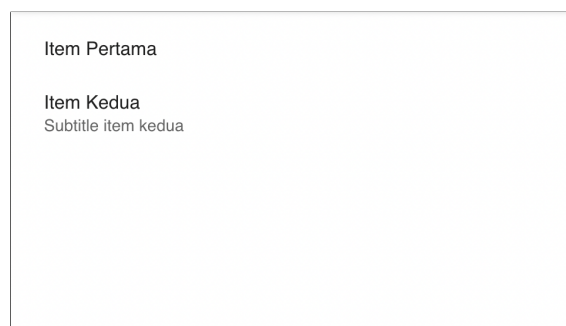
Gambar 15.5: Tampilan Card

## 15.6 List

List merupakan komponen yang digunakan untuk menampilkan listing data dalam jumlah banyak. List dapat diimplementasikan menggunakan element `<v-list-item>`. Sama halnya seperti Card, list juga memiliki sub-komponen tambahan yaitu content, title, dan subtitle:

```
1      <v-list-item>
2          <v-list-item-content>
3              <v-list-item-title>Item Pertama</v-list-item-title>
4          </v-list-item-content>
5      </v-list-item>
6
7      <v-list-item>
8          <v-list-item-content>
9              <v-list-item-title>Item Kedua</v-list-item-title>
10             <v-list-item-subtitle>Subtitle item kedua</v-list-item-subtitle>
11          </v-list-item-content>
12      </v-list-item>
```

Maka tampilan akan menjadi seperti berikut:



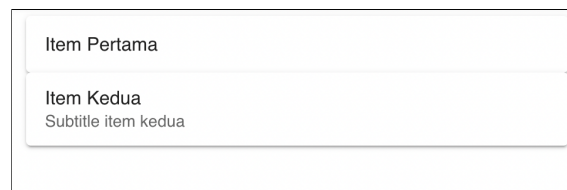
Gambar 15.6: Tampilan List

List juga dapat dikombinasikan dengan Card:

```
1      <v-card
2          elevation="2"
3      >
```

```
4      <v-list-item>
5        <v-list-item-content>
6          <v-list-item-title>Item Pertama</v-list-item-title>
7        </v-list-item-content>
8      </v-list-item>
9    </v-card>
10
11    <v-card
12      elevation="2"
13    >
14      <v-list-item>
15        <v-list-item-content>
16          <v-list-item-title>Item Kedua</v-list-item-title>
17          <v-list-item-subtitle>Subtitle item kedua</v-list-item-subtitle>
18        </v-list-item-content>
19      </v-list-item>
20    </v-card>
```

Maka tampilan akan menjadi seperti berikut:



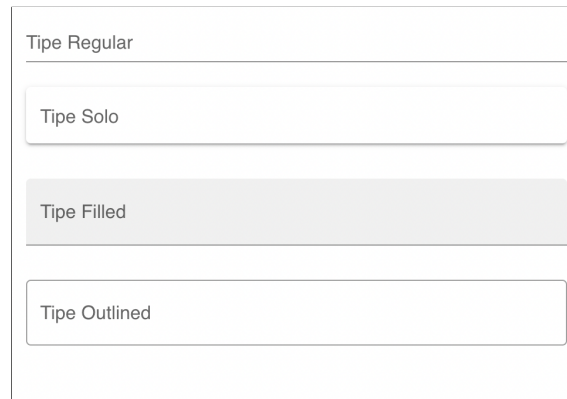
Gambar 15.7: Tampilan Kombinasi List dan Card

## 15.7 Form Input

Material Design juga memiliki beberapa panduan desain untuk Form dan Input. Sebuah Form harus diawali dengan komponen `<v-form>`, kemudian di dalamnya kita bisa menambahkan beberapa input. Untuk input berupa teks, kita dapat menggunakan element `<v-text-field>`. Textfiel sendiri memiliki beberapa jenis tampilan seperti pada kode berikut:

```
1 <v-text-field label="Tipe Regular"></v-text-field>
2 <v-text-field label="Tipe Solo" solo></v-text-field>
3 <v-text-field label="Tipe Filled" filled></v-text-field>
4 <v-text-field label="Tipe Outlined" outlined></v-text-field>
```

Maka tampilan akan menjadi seperti berikut:



Gambar 15.8: Tampilan Beberapa Jenis Text Field

Apabila ingin membuat text field untuk password, maka cukup tambahkan atribut `:type="password"` di dalam element tersebut.

Untuk membuat button, kita dapat menggunakan element `<v-btn>`. Sama seperti textfield, button juga memiliki beberapa varian tampilan:

```
1 <v-btn>Button Biasa</v-btn>
2 <v-btn rounded>Button Rounded</v-btn>
3 <v-btn outlined>Button Outlined</v-btn>
```

Maka tampilan akan menjadi seperti berikut:



Gambar 15.9: Tampilan Beberapa Jenis Button

## 15.8 Grid System

Material Design juga memiliki panduan untuk grid system, di mana komponen antarmuka akan disusun menggunakan aturan baris dan kolom. Terdapat dua element utama yang dapat digunakan yaitu `<v-row>` dan `<v-col>`, di mana `<v-col>` harus merupakan direct child dari `<v-row>`. Misalnya jika kita ingin membuat grid berukuran 2 baris x 3 kolom:

```
1 <v-row>
2   <v-col v-for="n in 3" :key="n">
3     <v-card outlined><v-card-text>{{n}}</v-card-text></v-card>
4   </v-col>
5 </v-row>
6 <v-row>
7   <v-col v-for="n in 3" :key="n">
8     <v-card outlined><v-card-text>{{n+3}}</v-card-text></v-card>
9   </v-col>
10 </v-row>
```

Maka tampilan akan menjadi seperti berikut:



Gambar 15.10: Tampilan Grid 2 x 3

Pada `<v-col>` terdapat beberapa pengaturan yang dapat dimodifikasi, yaitu untuk menentukan panjang dari kolom. Jumlah kolom keseluruhan pada `<v-col>` adalah 12, secara default panjang setiap item akan diatur secara merata. Apabila kita menginginkan panjang sebuah item maksimal, maka kita dapat menambahkan atribut `cols=12` pada `<v-col>`.

```

1      <v-row>
2          <v-col cols="12" v-for="n in 3" :key="n">
3              <v-card outlined><v-card-text>{{n}}</v-card-text></v-card>
4          </v-col>
5      </v-row>
6      <v-row>
7          <v-col v-for="n in 3" :key="n">
8              <v-card outlined><v-card-text>{{n+3}}</v-card-text></v-card>
9          </v-col>
10     </v-row>

```

Maka tampilan akan menjadi seperti berikut:

Gambar 15.11: Tampilan Untuk atribut `cols = 12`

Atribut `cols` berlaku untuk semua jenis breakpoint layar. Apabila kita ingin mengatur jumlah kolom pada breakpoint tertentu, maka pada bagian `cols` dapat kita ubah menjadi `sm` (small), `md` (medium), `lg` (large), dan `xl` (x-large).



## Bibliography

Articles

Books

