Front End React Development
sesi 10

Aplikasi React Pertama

Sesi 10 | Aplikasi React pertama

Kali ini kita akan membuat aplikasi React pertama kita. Kita akan manfaatkan folder aplikasi yang telah dibentuk pada sesi sebelumnya, yang bernama **my-app**.

Sebelum kita memulai membuat aplikasi React pertama kita, kita akan terlebih dahulu merancang target hasil akhir tampilan dari aplikasi kita. Pada aplikasi pertama ini, kita belum akan bermain dengan styling (CSS) yang cukup menarik. Kita akan bermain dengan bentuk dasar HTML saja, dan nanti kita akan kembangkan seiring berjalannya materi

Target tampilan kita adalah seperti di samping ini

Sesi 10 | Aplikasi React pertama - Class Component

Dengan berbekal mindset **component**, maka kita akan membagi tampilan kita menjadi 3 buah bagian : Header, Content, Footer.

Langkah 1 :
Bukalah file src/App.js, lalu ubah isinya menjadi seperti di samping ini. Dan edit file src/App.css menjadi seperti di bawah ini

```css
.container {
  width: 500px;
  margin: 0 auto;
}

.header {
  text-align: center;
}

.footer {
  text-align: center;
}
```

```jsx
import React from 'react';
import './App.css';

class Header extends React.Component {
}

class Content extends React.Component {
}

class Footer extends React.Component {
}

class App extends React.Component {
  render() {
    return (
      <div className="container">
      </div>
    );
  }
}

export default App;
```

HACKTIV8

Sesi 10 | Aplikasi React pertama - Class Component

Langkah 2 :
Isikan functional component Header() dengan kode seperti di bawah ini

```
4   class Header extends React.Component {
5     render() {
6       return (
7         <header className="header">
8           <h1>My First React app</h1>
9         </header>
10      )
11    }
12  }
13
```

Langkah 3 :
Isikan functional component Content() dengan kode seperti di bawah ini

```
14  class Content extends React.Component {
15    render() {
16      return (
17        <div className="content">
18          <p>
19            Silakan isi dengan paragraf
20          </p>
21        </div>
22      )
23    }
24  }
25
```

HACKTIV8

Sesi 10 | Aplikasi React pertama - Class Component

Langkah 4 :
Isikan functional component Footer() dengan kode seperti di bawah ini

```
24  class Footer extends React.Component {
25    render() {
26      return (
27        <div className="footer">
28          <p>&copy; My self - 2021</p>
29        </div>
30      )
31    }
32  }
33
```

Langkah 5 :
Mari kita panggil semua functional component kita di App component kita, dengan kode seperti di bawah ini :

```
34  class App extends React.Component {
35    render() {
36      return (
37        <div className="container">
38          <Header />
39          <hr />
40          <Content />
41          <hr />
42          <Footer />
43        </div>
44      );
45    }
46  }
47
```

Sesi 10 | Aplikasi React pertama - Class Component

Langkah 6 :
Jalankan aplikasi dengan menjalankan perintah berikut pada folder root aplikasi kita :

```
> npm start
```

Dan silakan perhatikan hasilnya pada browser kita. Jika sudah muncul seperti di samping, maka kita telah berhasil membuat aplikasi React pertama kita. Tentu saja, bentuk ini masih sangat dasar, masih diperlukan banyak pengembangan setelah ini

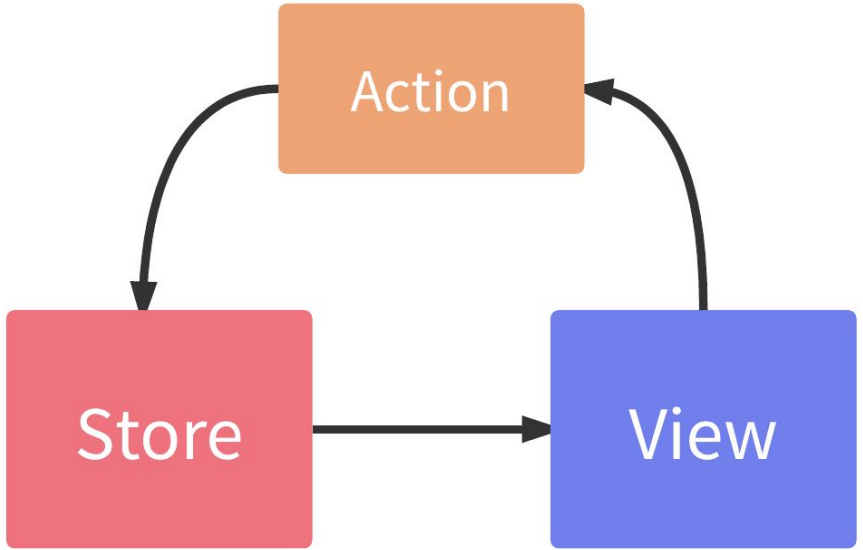
Data Flow di React

Sesi 10 | Data Flow di React

[Aliran data satu arah - 1/2]

Sedikit berbeda dengan beberapa framework seperti Angular yang menerapkan aliran data dua arah, di React data hanya mengalir satu arah. Hal ini menjadi signifikan karena dengan menerapkan hal ini, perubahan data menjadi lebih mudah dimengerti dan lebih mudah diprediksi.

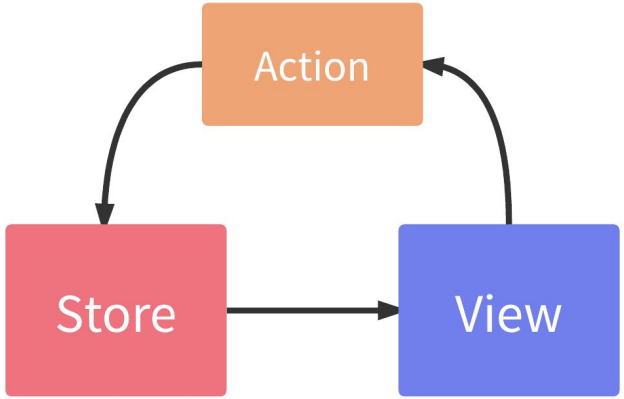
Aliran data di React dapat diilustrasikan seperti berikut.



Sesi 10 | Data Flow di React

[Aliran data satu arah - 2/2]

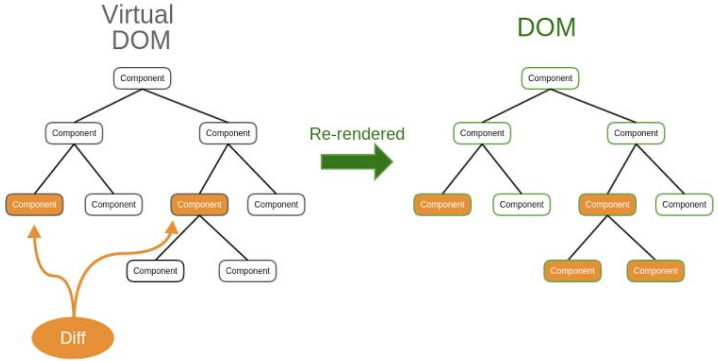
View adalah kumpolan komponen yagn sudah kita bahas di bagian sebelumnya. Di view juga ada berbagai action seperti onClick ketika button di klik, onChange ketika pengguna memasukkan data di form dan banyak lagi. Dan di React, semua data disimpan kedalam sebuah wadah yang disebut dengan store atau state. Uniknya, ketika terjadi perubahan data, katakanlah seorang pengguna meng-klik button atau action yang lainnya, view tidak bisa mengubah data atau apapun juga. View harus menghubungi action memberitahu bahwa ada sebuah action yang di trigger oleh pengguna. Dan tugas action-lah yang kemudian menghubungi store untuk kemudian mengubah data sesuai actionnya. Dan setelah itu baru kemudian store mengubah view hingga sesuai dengan data yang baru saja berubah. Jadi satu jalur, dari view ke action ke store dan kembali ke view. Dan tidak bisa sebaliknya. Untuk mengubah data, view harus selalu melalui action dan seterusnya.



Sesi 10 | Data Flow di React

[Virtual DOM]

Masih berhubungan dengan alur data satu arah sebelumnya setiap kali terjadi perubahan data yang terjadi di store, React atau dalam hal ini view akan selalu melakukan proses render ulang. Ya, betul setiap kali ada perubahan data, React akan melakukan proses render ulang seluruh komponen dan hal ini yang membuat React keren! Dan React memang didesain seperti itu untuk memudahkan workflow kita sebagai developer. Hal ini juga menyebabkan React tidak membutuhkan data binding yang aneh-aneh (baca: magical).

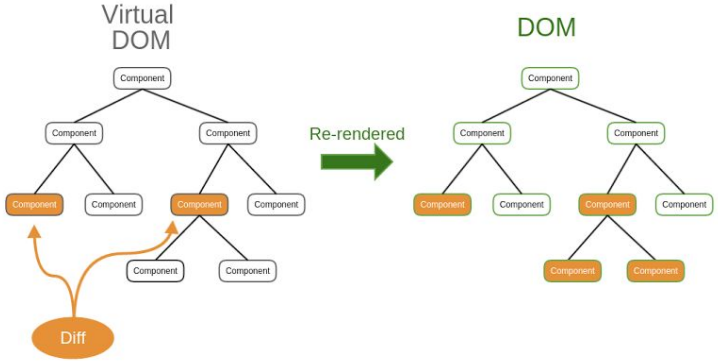


DOM ini strukturnya berbentuk *tree* atau pohon. Dan ketika sebuah elemen diakses, dimodifikasi ataupun membuat elemen baru, browser harus mencari ke seluruh cabang-cabang dari pohon DOM tersebut. Dan hal itu membutuhkan waktu yang lama dan komputasi yang cukup berat. Dan karena aplikasi web modern membutuhkan data yang terus menerus berubah (misalnya aplikasi social media seperti facebook atau twitter) dan setiap perubahan harus melakukan komputasi yang berat untuk mengubah DOM, makanya React memutuskan menggunakan Virtual DOM daripada harus mengubah DOM secara langsung.

Sesi 10 | Data Flow di React

[Virtual DOM]

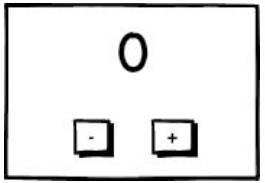
Virtual DOM ini sebenarnya hanyalah struktur data yang menyimpan informasi lengkap tentang DOM. Seperti blueprint kalau kita mau membangun rumah atau ruangan. Jadi setiap kali terjadi perubahan data dan harus mengubah DOM, React melakukan perubahan tersebut di VirtualDOM terlebih dahulu. Baru kemudian React membandingkan VirtualDOM yang sudah berubah dengan DOM yang sebenarnya. Jika ada yang berbeda, VirtualDOM akan mengubah DOM yang berubah saja, jadi tidak semuanya diganti sehingga tidak seberat jika harus mengubah seluruh pohon DOM.\ Mengubah VirtualDOM jauh, jauh lebih ringan karena VirtualDOM sederhananya hanyalah struktur data atau kerangkanya saja jadi perubahan akan lebih cepat dan ringan.



Sejak React mengadaptasi Virtual DOM dan berhasil mempermudah workflow developer sekaligus membuat aplikasi menjadi cepat secara performa, library/framework lain juga ikut mengadaptasi Virtual DOM. Sebut saja Vue, Angular hingga Ember saat ini juga menerapkan Virtual DOM.

Sesi 10 | Data Flow di React

[Aliran data satu arah]



Mari kita lihat contoh nyata. Misalkan kita punya sebuah aplikasi counter dengan button + untuk menambahkan angka dan - untuk mengurangi angka. Angka counter awal adalah 0.

Ketika pengguna memencet tombol +, view memberitahu action bahwa tombol + dipencet. View disini tidak memberitahu counter sekarang berapa. Dia hanya memberikan informasi bahwa tombol. + dipencet.

Action kemudian melanjutkan informasi tersebut ke store juga tidak membawa informasi tentang current counter number yang saat ini masih 0 yang tampil di view. Karena yang mengetahui informasi current counter number adalah store sendiri, bukan action dan juga bukan view.

Setelah informasi sampai ke store, tugas store adalah melakukan kalkulasi yang tadinya angkanya 0 ditambah 1 menjadi 1. Dan seteleh data diubah, store menginformasikan kepada view bahwa data berubah, silakan ubah view nya sehingga datanya dari 0 menjadi 1. Dan seterusnya.

Sesi 10 | Data Flow di React

[State - 1/5]

Aplikasi web modern biasanya dibangun berdasarkan data atau istilah kerennya data-driven. Bayangkan aplikasi social media seperti Facebook, Twitter atau Instagram. Sebagai social media, data adalah aliran darah. Bayangkan sebuah news feed yang secara dinamis bertambah sesuai dengan aliran data yang masuk. Disinilah salah satu keunggulan utama React dan memang React dibuat untuk dapat menangani hal-hal seperti ini.

Mari kita lihat contoh yang lebih sederhana agar kita dapat melihat dan mengetahui apakah state itu sebenarnya. Percaya atau tidak, hampir semua aplikasi itu memiliki state. Apakah teman-teman mendevelop penggunakan View, jQuery, Angular, Ember dan lain sebagainya pasti semua library/framework tersebut memiliki cara untuk menangani state. Secara definisi, state adalah:

Seluruh informasi dari sebuah aplikasi yang dibutuhkan pada satu waktu.

Definisi diatas memang sudah disederhanakan, tapi cukup menjelaskan apa itu state dan tujuan dari state. Dengan kata lain state adalah seluruh informasi yang kita butuhkan untuk menggambarkan user interface pada satu waktu.

Sesi 10 | Data Flow di React

[State - 2/5]

Mari kita, perhatikan potongan kode di samping ini

MENDEFINISIKAN / MEMBUAT STATE

Perhatikan baris ke 6 - 8 dari kode di samping ini. Inilah contoh mendefinisikan sebuah state di dalam sebuah class component, yaitu di dalam constructor nya si component

MEMBACA STATE DI DALAM RENDER

Perhatikan baris ke 13 dari kode di samping ini. Inilah contoh cara membaca state di dalam render nya component

```
1   import React from 'react';
2
3   export default class Users extends React.Component {
4     constructor() {
5       super()
6       this.state = {
7         username: 'user01'
8       }
9     }
10
11    render() {
12      return (
13        <h1>{this.state.username}</h1>
14      )
15    }
16  }
17
```

HACKTIV8

Sesi 10 | Data Flow di React

[State - 3/5]

MEMBACA STATE DI DALAM FUNCTION

Perhatikan baris ke 11 - 13 dari kode di samping ini. Inilah contoh cara membaca state di dalam function yang didefinisikan di dalam class component

Dan pada baris 17 adalah contoh kode yang memanggil function yang sudah kita buat di atas

```
1   import React from 'react';
2
3   export default class Users extends React.Component {
4     constructor() {
5       super()
6       this.state = {
7         username: 'user01'
8       }
9     }
10
11    getUsername() {
12      return this.state.username
13    }
14
15    render() {
16      return (
17        <h1>{this.getUsername()}</h1>
18      )
19    }
20  }
21
```

Sesi 10 | Data Flow di React

[State - 4/5]

MENULIS STATE DI DALAM SEBUAH FUNCTION

Perhatikan baris ke 12 - 14 dari kode di samping ini. Ini adalah contoh kode untuk melakukan penulisan terhadap sebuah state. SANGATLAH DILARANG untuk melakukan update langsung dengan syntax

✗ `this.state.username = 'user01'`

Function setUsername di trigger dengan klik button di baris 22

```
1  import React from 'react';
2
3  export default class Users extends React.Component {
4    constructor() {
5      super()
6      this.state = {
7        username: ''
8      }
9    }
10
11   setUsername = () => {
12     this.setState({
13       username: 'user01'
14     })
15   }
16
17   render() {
18     return (
19       <div>
20         <h1>{this.state.username}</h1>
21         <br />
22         <button onClick={this.setUsername}>Set Username</button>
23       </div>
24     )
25   }
26 }
27
```

Sesi 10 | Data Flow di React

[State - 5/5]

UPDATE STATE DI DALAM SEBUAH FUNCTION

Pada contoh ini, kita membuat 2 buah function : increment dan decrement untuk menambah dan mengurangi angka counter.

Kedua function tersebut di trigger ketika tombol pada baris 28 dan 29 di click

Perhatikan bahwa kita melakukan update state dengan setState, dan kita tetap memanggil state awalnya (this.state.counter) untuk di update dengan ditambah atau dikurangi 1

```
1  import React from 'react';
2
3  export default class Counter extends React.Component {
4    constructor() {
5      super()
6      this.state = {
7        counter: 0
8      }
9    }
10
11   increment = () => {
12     this.setState({
13       counter: this.state.counter + 1
14     })
15   }
16
17   decrement = () => {
18     this.setState({
19       counter: this.state.counter - 1
20     })
21   }
22
23   render() {
24     return (
25       <div>
26         <h3>{this.state.counter}</h3>
27         <br />
28         <button onClick={this.increment}>+</button>
29         <button onClick={this.decrement}>-</button>
30       </div>
31     )
32   }
33 }
34
```

Sesi 10 | Data Flow di React

[Props - 1/2]

Props adalah salah satu fasilitas dari React untuk bisa menerima data yang dikirimkan dari component lain. Data ini sangatlah luas, bisa jadi berupa informasi statis, atau state dari component pengirim.

PROPS DARI INFORMASI STATIS

Pada contoh disamping, didemonstrasikan pengiriman props dari component Hero ke component HeroName dengan menggunakan nama props “name”, seperti terlihat pada baris ke 11

Component HeroName menerima props dengan cara yang ditunjukan pada baris ke 5

```
1   import React from 'react';
2
3   class HeroName extends React.Component {
4     render() {
5       return <h3>{this.props.name}</h3>
6     }
7   }
8
9   export default class Hero extends React.Component {
10    render() {
11      return <HeroName name="Hercules" />
12    }
13  }
14
```

Sesi 10 | Data Flow di React

[Props - 2/2]

Props adalah salah satu fasilitas dari React untuk bisa menerima data yang dikirimkan dari component lain. Data ini sangatlah luas, bisa jadi berupa informasi statis, atau state dari component pengirim.

PROPS DARI STATE YANG DIKIRIM

Pada contoh disamping, didemonstrasikan pengiriman props dari component Hero ke component HeroName dengan menggunakan nama props “name”, seperti terlihat pada baris ke 18. Dan value yang dikirim adalah value dari state yang bernama “name”

Component HeroName menerima props dengan cara yang ditunjukan pada baris ke 5

```
1  import React from 'react';
2
3  class HeroName extends React.Component {
4    render() {
5      return <h3>{this.props.name}</h3>
6    }
7  }
8
9  export default class Hero extends React.Component {
10   constructor() {
11     super()
12     this.state = {
13       name: "Hercules"
14     }
15   }
16
17   render() {
18     return <HeroName name={this.state.name} />
19   }
20 }
21
```

Proptypes +

Sesi 10 | PropTypes

PropTypes memungkinkan kita melakukan validasi terhadap props yang kita harapkan.

Pertama-tama kita harus melakukan instalasi library proptypes terlebih dahulu melalui terminal, pada root folder aplikasi kita, dengan perintah :

```
> npm install prop-types --save
```

Misal pada contoh kali ini, kita akan melakukan pemeriksaan apakah **name** adalah sebuah string.

PropTypes ini bisa melakukan validasi untuk berbagai tipe data. Mulai dari tipe data scalar hingga ke struktur data seperti array, object dan kita juga bisa melakukan validasi sesuai dari bentuk dari sebuah object (shapeOf).

```
1  import React from 'react';
2  import PropTypes from 'prop-types';
3
4  class HeroName extends React.Component {
5    render() {
6      return <h3>{this.props.name}</h3>
7    }
8  }
9
10 HeroName.propTypes = {
11   name: PropTypes.string
12 }
13
14 export default class Hero extends React.Component {
15   constructor() {
16     super()
17     this.state = {
18       name: true
19     }
20   }
21
22   render() {
23     return <HeroName name={this.state.name} />
24   }
25 }
26
```