
E-BOOK
Muhammad Azamuddin
Hafid Mukhlasin
MODERN REACT
Be Frontend Javascript Developer
KUNGFU KODING
2020

Modern React - Be Fullstack Javascript Developer

Sample Ebook https://bukureact.id

Daftar Isi

Javascript Fundamental Untuk ReactJS

Pendahuluan

Materi yang dibahas pada bab ini meliputi:

- Histori & Standardisasi Javascript
- Penulisan Sintaks & Coding Style
- Template Literal
- Variabel & Function
- Conditional Operator
- Fungsi Pada Array
- Spread Operator
- Destructuring Assignment
- Module: Export & Import
- Asynchronous Programming
- Functional Programming

Dengan mempelajari materi ini diharapkan kamu dapat lebih mudah memahami materi yang akan diajarkan pada bab-bab berikutnya.

Mengapa demikian?

Sebab penerapan pustaka ReactJS sendiri cukup banyak mengekplorasi fitur-fitur dari Javascript.

Namun, untuk dapat memahami bab ini dengan baik maka sebaiknya kamu telah mengenal dasar-dasar teknologi web seperti HTML, CSS dan Javascript.

Buku ini tidak akan membahas tentang materi dasar tersebut. Oleh karena itu jika kamu merasa lemah pada materi tersebut, maka sebaiknya kamu pelajari dahulu.

Berikut ini beberapa sumber belajar yang penulis rekomendasikan:

- HTML & CSS
- Javascript fundamental
- Javascript tutorial

Terdapat dua tools utama yang perlu kamu persiapkan untuk mengikuti tutorial pada bab ini yaitu web browser dan code editor.

Tools yang penulis rekomendasikan adalah:

- Google Chrome sebagai web browser;
- Visual Studio Code sebagai kode editor.

> Catatan: pastikan kamu sudah menginstalasi kedua tools tersebut.

Histori Javascript

Beberapa tahun belakangan ini adalah eranya Javascript, dimana perkembangannya cukup pesat dengan ditandai lahirnya banyak framework dan pustaka berbasis Javascript.

Javascript tidak hanya sebagai bahasa pemrograman di sisi client yang hanya berjalan di browser, namun kini Javascript dapat berjalan di luar browser.

Javascript berjalan pada sisi server melalui NodeJS, pada desktop melalui Electron, bahkan Jaascript juga digunakan pada pengembangan aplikasi mobile misalnya Ionic & React Native.

Kilas Balik

Awalnya, Javascript hanya berperan sebagai pelengkap untuk pengembangan aplikasi web.

Pada masa itu, penerapan Javascript, banyak kita jumpai hanya untuk tujuan validasi form input & animasi saja.

Dengan alasan kompatibilitas antar browser yang saat itu masih sangat buruk, bahkan seringkali penggunaan Javascript ini dihindari.

Diskusi tentang web lebih banyak membahas tentang kode-kode HTML (XHTML/HTML5) atau CSS yang digadang-gadang bakal mengubah web masa depan.

Setelah masa itu kemudian JQuery yang merupakan pustaka berbasis Javascript mengambil alih dalam waktu yang lama.

JQuery hadir dengan menawarkan API-API sederhana untuk memanipulasi document object model (DOM) HTML serta mengatasi kompatibilitas antar browser.

JQuery memang sangat memudahkan programmer dalam membuat website yang interaktif, sehingga menjadikannya sangat populer pada masa itu.

Banyak pustaka maupun framework web menggunakan JQuery sebagai mesin utamanya. Tidak lengkap rasannya jika membuat web tanpa JQuery, seolah JQuery telah menggantikan Javascript.

Kondisi ini kemudian berubah setelah keluarnya Javascript ES5 pada sekitar tahun 2011 dan lahirnya teknologi NodeJS. Hal itu kemudian memicu berkembangnya pustaka-pustaka Javascript modern berbasis ES5 dan NodeJS tersebut.

Pada masa ini, lahirlah pustaka Javascript modern seperti ReactJS, VueJS & Angular yang sedikit demi sedikit menggerus kejayaan JQuery dari muka bumi 😃.

Standardisasi Javascript

Javascript merupakan bahasa pemrograman yang awalnya dibuat oleh Brendan Eich dari Netscape pada tahun 1995. Saat itu, Javascript hanya didesain untuk berjalan pada platform web browser yaitu Netscape Navigator (sekarang Mozilla Firefox).

Nama **Javascript** sendiri sengaja digunakan untuk mendongkrak popularitasnya karena bahasa pemrograman yang saat itu cukup populer adalah Java. Hal ini tentu atas izin dari perusahaan Sun

Microsystem sebagai pemilik bahasa pemrograman Java. Nama sebelumnya adalah Mocha kemudian berganti nama menjadi Livescript.

Pada saat yang sama, Microsoft yang merupakan rival utama dari Netscape terutama dalam persaingan web browser, mengeluarkan browser terbarunya yaitu Internet Explorer versi 3.1 yang mengusung JScript yaitu bahasa pemrograman yang mirip dengan Javascript, namun dengan tambahan fitur lain serta berjalan hanya di browser IE tersebut.

Dua bahasa ini sedikit berbeda, sehingga developer web pada masa itu harus membuat web mereka dalam dua versi agar dapat mendukung keduanya.

Pada tahun 1996-1997, polemik ini kemudian mulai mereda semenjak para developer sepakat untuk mengajukan standardisasi bahasa scripting untuk browser (mencakup Javascript dan JScript) pada lembaga standardisasi komputer Eropa yang bernama European Computer Manufacturers Association (ECMA).

ECMA kemudian membentuk komite dengan nama TC-39 (Technical Committee 39) untuk membahas standardisasi ini dengan anggota yang berasal dari perusahaan TI besar saat itu seperti Microsoft, Netscape, Sun Microsystem, dll. Komite itu akhirnya menghasilkan standard baru yang diberi nama ECMAScript dengan label ECMA-262 atau ES1.

Pada tahun 1998, keluarlah ES2 yang mendukung *regular expression*, *statemen control* yang beragam dan penanganan string yang lebih baik.

Pada tahun 1999, keluarlah ES3 yang mendukung *regular expression* yang lebih *powerful*, penangan string, statemen control baru, penanganan eksepsi try/catch, definisi dari kesalahan program yang lebih ketat, pemformatan untuk keluaran numerik dan perubahan minor diantisipasi dari fasilitas internasionalisasi yang akan datang dan pertumbuhan bahasa masa depan.

ES4 berusaha mengadopsi secara besar-besaran banyak teknologi web namun sayang standardisasi itu tidak sampai selesai sehingga tidak dipublikasikan. Hal itu menyebabkan, antara tahun 1999 sampai dengan 2009 terjadi kevakuman dalam pengembangan standardisasi Javascript.

Sebaliknya, CSS justru semakin berkembang dengan munculnya banyak framework CSS seperti: Blueprint, 960, YUI Grids dsb.

Di sisi lain, pada masa itu Microsoft melalui Internet Explorer menguasai *market share* browser hingga mencapai 90%.

Microsoft memang ikut bergabung dalam TC-39 dan ikut berkontribusi pada ECMAScript, namun disisi lain mereka lebih banyak membuat aturan sendiri dengan menambahkan fitur-fitur baru untuk Javascript di browser mereka.

Pada tahun 2006, lahirlah JQuery untuk mengatasi kompatibilitas kode antar browser, dengan API yang universal dan sederhana membuatnya menjadi pustaka Javascript populer hingga bertahun tahun.

JQuery hanya fokus pada manipulasi tampilan melalui DOM dan tidak memiliki fitur spesifik untuk menangani komunikasi data antar tampilan. Hal ini kemudian memicu lahirnya beberapa pustaka Javascript yang menangani komunikasi data seperti Backbone, Knockout, dan Ember.

Pada tahun 2009 sampai dengan 2011, keluarlah ES5 yang fokus pada kodifikasi spesifikasi yang telah secara umum diterapkan pada browser. Fitur baru yang ditambahkan seperti: fungsi tambahan untuk manipulasi array, dukungan untuk format JSON serta peningkatan pemeriksaaan kesalahan dan keamanan program.

Pada masa ini lahirlah NodeJS yaitu tools untuk menjalankan kode Javascript di luar browser menggunakan Google V8 JavaScript engine. Lahir pula AngularJS versi satu yang merupakan framework Javascript MVC lahir. Angular mengusung beberapa fitur seperti two way data binding, routing serta dependecy injection.

Pada tahun 2013 hingga 2014, lahirlah pustaka Javascript modern yaitu ReactJS & VueJS yang mengusung konsep virtual DOM.

**Apa itu virtual DOM? **

DOM merepresentasikan struktur dokumen HTML yang berbentuk hirarki pohon, yang masing-masing node-nya merupakan elemen HTML.

Pada sebuah web yang besar dan kompleks tentu struktur DOMnya juga akan besar dan kompleks, sehingga manipulasi terhadap DOM membutuhkan sumber daya yang relatif besar sehingga dirasa kurang efisien.

Alih-alih melakukan manipulasi DOM secara langsung sebagaimana yang dilakukan oleh pustaka JQuery maupun AngularJS, virtual DOM mencoba membuat abstraksi struktur DOM dalam bentuk virtual serta menyimpannya pada memori.

Proses manipulasi terhadap struktur DOM tidak dilakukan langsung pada real DOM atau browser, akan tetapi hanya dilakukan pada objek virtualnya, sehingga prosesnya menjadi lebih cepat.

Setelah manipulasi terhadap objek virtual DOM selesai, maka objek virtual tersebut dibandingkan dengan real DOM sehingga didapatkan bagian yang berbeda. Hanya pada bagian yang berbeda itu saja yang kemudian akan diaplikasikan ke real DOM atau direnderer ke browser.

Metode ini tentu jauh lebih efektif dan cepat dibandingkan memanipulasi DOM secara langsung. Metode ini kemudian diikuti oleh Angular versi dua.

Pada masa ini lahirlah NPM (Node Package Manager) yaitu pusat repositori pustaka berbasis Javascript sekaligus tools untuk memudahkan instalasi pustaka Javascript secara online.

Sampai dengan tahun 2015, semua browser modern telah mendukung penuh ECMAScript 5.1, adapun browser lama mendukung setidaknya ECMAScript 3.

Pada tahun 2015, ES6 (ES2015) diumumkan dengan mengusung banyak fitur modern seperti arrow function, module, deklarasi class, lexical block scoping, iterator dan generator, promises untuk asynchronous programming, destructuring patterns, dsb.

Setelah itu setiap tahunnya diumumkan standard baru ES.

Semua browser modern pada versi terbarunya memang telah mendukung ES versi baru, namun belum semua pengguna mengupdate browsernya ke versi terbaru. Hal ini menyebabkan banyak kode Javascript yang menggunakan standard ES baru tidak dapat berjalan dengan baik.

Untuk mengatasi hal ini lahirlah pustaka BabelJS yaitu pustaka yang mengkonversi (transpile) kode Javascript versi ES6 ke atas menjadi kode Javascript versi ES5, sehingga lebih bisa diterima oleh banyak browser pengguna.

Solusi dari BabelJS ini tentu menjadi win-win solution baik bagi pustaka Javascript maupun pengguna web. Di mana pustaka Javascript tetap bisa mengikuti standard baru dari ES sedangkan pengguna web tidak harus selalu mengupdate browsernya ke versi terbaru. Hal inilah yang membuat BabelJS menjadi tools yang sangat populer untuk digunakan dalam pengembangan aplikasi berbasis Javascript.

Terkait dengan ReactJS, BabelJS memiliki fitur untuk mengkonversi kode JSX atau template element pada ReactJS menjadi bentuk HTML.

Pengenalan ReactJS

Pendahuluan

Materi yang akan dibahas pada bab ini meliputi:

- Mengenal ReactJS?
- Memulai menggunakan ReactJS
- Mengenal template JSX
- Menggunakan style CSS
- Memahami arsitekur ReactJS

Dengan mempelajari materi ini maka diharapkan kita dapat lebih mengenal, dapat menggunakannya serta memahami bagaimana arsitektur dari ReactJS ini.

Materi pada bab sebelumnya merupakan pondasi dasar untuk dapat mempelajari dengan lebih mudah pustaka ReactJS. Jadi sebelum melanjutkan untuk mempelajari bab ini maka pastikan kamu telah bener-benar memahami materi sebelumnya.

Tools yang dibutuhkan untuk dapat mengikuti panduan pada bab ini adalah:

- Code editor (VS Code)
- Browser (Chrome)
- NodeJS
- Koneksi internet untuk instalasi

Eksplorasi Komponen

Pendahuluan

Materi yang akan dibahas pada bab ini meliputi:

- Membuat komponen
- Komponen list & tabel
- Komponen form

Dengan mempelajari materi ini maka diharapkan kita dapat memahami dan membuat komponen pada ReactJS secara lebih detail dan mendalam. Pemahaman yang baik terhadap materi pada bab sebelumnya sangat dibutuhkan agar dapat membantu kita lebih mudah dalam mempelajari materi ini.

Mengenal Komponen

Sebagaimana yang telah disinggungpada bagian sebelumnya bahwa komponen merupakan bagian terkecil dari aplikasi ReactJS yang dapat berdiri sendiri. Komponen disusun oleh elemen-elemen ReactJS. Secara konsep, komponen mirip seperti fungsi pada JS, yang dapat menerima input data dari luar komponen (berupa props) dan mengembalikan nilai balik berupa elemen ReactJS yang menggambarkan bagian-bagian dari antarmuka yang akan tampil dibrowser.

Membuat Komponen

Kita bisa membuat sebuah komponen pada ReactJS sebagaimana kita membuat fungsi Javascript pada umumnya yang mengembalikan suatu nilai. Nilai kembalian pada fungsi komponen ini dapat berupa: string atau teks, elemen HTML, atau komponen ReactJS lainnya.

Perhatikan contoh kode komponen berikut ini.

```
const Hello = () => 'Hello!'
```

Komponen Hello diatas mengembalikan nilai dalam bentuk teks `Hello!`. Fungsi komponen tersebut ditulis dengan menggunakan format penulisan ES6 short-hand arrow function.

Kita juga boleh saja menuliskan komponen `Hello` tersebut dalam format lain sebagai berikut:

```
// normal function
function Hello(){
    return 'Hello!'
}

// function as variable
const Hello = function (){
    return 'Hello!'
}
```

```
// arrow function
const Hello = () => {
    return 'Hello!'
}
```

Sesuai dengan konsensus, penulisan nama komponen pada ReactJS menggunakan format **StudlyCaps** atau huruf pertama setiap awal kata menggunakan huruf kapital, contoh: `Hello`, `HelloWorld`, dll.

Deklarasi dari komponen ini kemudian dapat kita simpan pada file tersendiri. Adapun nama filenya sama dengan nama dari komponennya, sebagai contoh, jika nama komponennya `Hello` maka disimpan pada file bernama `Hello.js`, jika `HelloWorld` maka tentu nama filenya `HelloWorld.js`.

Agar suatu komponen dapat digunakan atau diimpor ditempat lain maka kita bisa gunakan perintah untuk mengekspor komponen tersebut. Ada beberapa cara mengekspor sebuah komponen, salah satunya dengan menambahkan perintah `export` di depan konstanta dari fungsi komponen.

```
export const Hello = () => 'Hello!'
```

Varian lain untuk mengekspor komponen adalah dengan meletakkan perintah `export` setelah deklarasi komponen.

```
const Hello = () => 'Hello!'
export Hello
```

Kita bisa menambahkan parameter `default` setelah perintah `export` jika dalam satu file komponen tersebut hanya berisi satu komponen (*best practice*-nya memang satu file satu komponen), atau kita ingin ada komponen utama yang diajikan acuan ketika `import`.

Maka kodenya menjadi sebagai berikut:

```
export default const Hello = () => 'Hello!'
```

Atau tidak mengapa kita menuliskannya sebagai berikut:

```
export default () => 'Hello!'
```

Menampilkan Komponen

Sebelum kita menampilkan komponen, maka kita perlu mengimpor file berisi deklarasi komponen yang telah kita buat sebelumnya.

```
```

```
import Hello from './Hello';
```

Perintah diatas artinya mengimpor komponen `<Hello>` pada file `Hello.js` yang berada pada direktori yang sama dengan file yang mengimpornya. Kita tidak perlu menambahkan ektensi `.js` setelah nama file komponen, karena CRA atau dalam hal ini `webpack` akan otomatis melakukan hal tersebut untuk kita.

Keyword `Hello` setelah `import` akan merujuk kepada komponen `<Hello>` yang diekspor secara default. Namun jika misalnya komponen `<Hello>` tidak dieskpor secara default maka perintah impornya menjadi sebagai berikut:

```
import { Hello } from './Hello';
```

Yap, komponen yang diimpor akan dikonversi kedalam bentuk objek yang berisi variabel yang diekspor pada file komponen, sehingga kita bisa menggunakan teknik *object destructuring* untuk mengambil salah satu item variabel yang diekspor tersebut.

Penulisan komponen ReactJS ini mirip dengan elemen HTML biasa yaitu dengan menggunakan tag sesuai dengan nama komponennya.

```
<Hello></Hello>
```

Jika komponen tersebut tidak memiliki komponen anak maka kita juga bisa menuliskannya dalam bentuk tunggal.

```
<Hello />
```

Komponen tersebut kemudian bisa kita tampilkan secara langsung pada browser dengan menggunakan pustaka ReactDOM melalui method `ReactDOM.render()`.

```
import * as Hello from './Hello';
ReactDOM.render(
    <Hello />,
    document.getElementById('root')
)
```

Pada CRA, kita bisa letakkan kode `render` diatas pada file `src/index.js`, silakan disesuaikan. Maka jika dijalankan `npm start`, pada browser akan muncul teks `Hello!` yang merupakan nilai balik dari komponen `<Hello>` di atas.

```
Hello!
```

Mungkin dalam hati kamu bertanya, "kenapa tampilannya jadi flat tanpa CSS sebagaimana pada bagian sebelumnya?"

Yap karena tampilan yang sebelumnya tersebut didefiniskan pada file `App.js`. File `App.js` sebenarnya merupakan komponen biasa namun oleh CRA dijadikan sebagai container atau wrapper dari komponen lainnya. Pada file ini terdapat perintah untuk mengimpor file CSS `App.css` sehingga tampilan yang dihasilkan sudah ada stylingnya.

Oleh karena itu, kita bisa gunakan komponen `<Hello>` di sini, jadi tidak langsung pada `index.js` (kembalikan kode pada file `index.js` seperti semua).

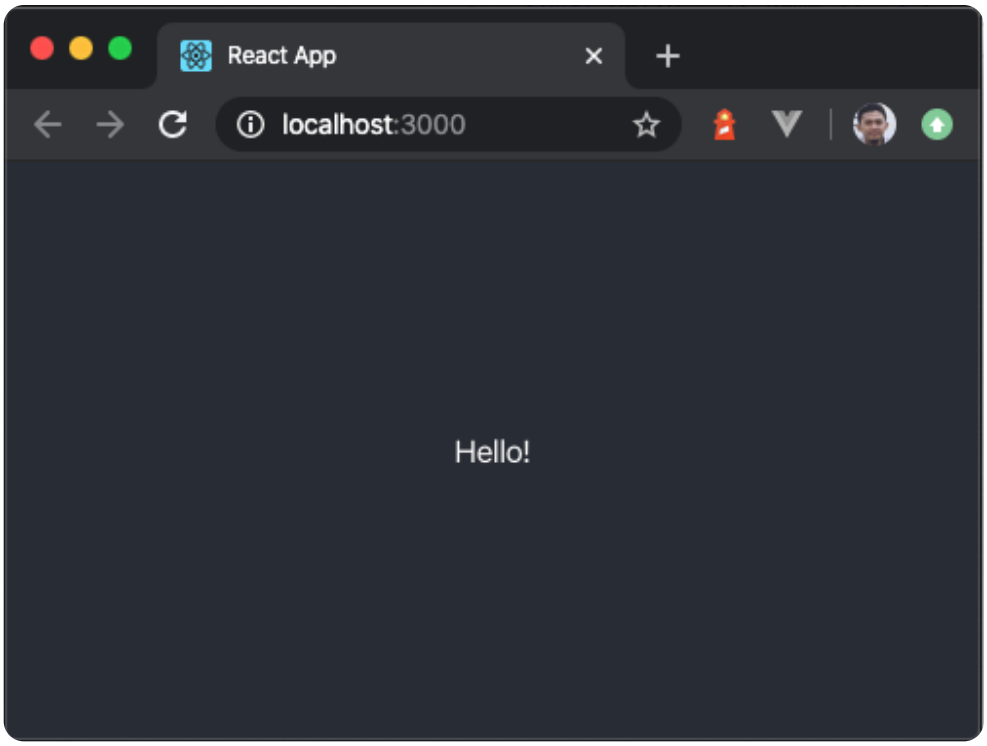
Berikut ini kode lengkap dari komponen `<App>` pada file `src/App.js`.

```
import React from 'react'
import './App.css'
import Hello from './Hello'

function App () {
  return (
    <div className='App'>
      <header className='App-header'>
        <Hello />
      </header>
    </div>
  )
}

export default App
```

Jika kita jalankan pada browser melalui perintah `npm start` maka hasilnya:



Properti Pada Komponen

Properti komponen atau biasa disebut dengan **props** merupakan atribut yang dapat dikirimkan atau diinputkan pada suatu komponen. Tujuannya agar komponen dapat menampilkan sesuatu secara dinamis sesuai dengan data props yang diinputkan.

Aliran data pada props bersifat satu arah yaitu dari komponen induk ke komponen anaknya. Props pada komponen ReactJS bersifat *read only*, sehingga kita hanya boleh menggunakannya tanpa boleh memodifikasinya.

Deklarasi Props

Props dapat kita deklarasikan melalui argumen pada fungsi komponen. Perhatikan kode berikut:

```
const Hello = (props) => `Hello ${props.name}!`
// const Hello = (props) => 'Hello '+props.name+'!'
```

Props bertipe objek, adapun `name` nantinya merupakan variabel atau item pada objek props yang kemudian disertakan pada nilai balik dari komponen tersebut.

Selanjutnya, kita bisa mendefinisikan nilai props tersebut pada saat menggunakan komponen. Cara mendefinisikan nilai props dengan menambahkan atribut sesuai dengan nama variabel yang kita inginkan dari props. Contoh:

```
<Hello name='Mastah'>
```

Kode di atas, jika direnderpada browser akan menampilkan teks:

```
Hello Mastah!
```

Bagaimana jika props tidak kita definisikan pada saat memanggil komponen.

```
<Hello />
```

Maka pada browser hanya akan muncul teks `Hello undefined!`.

Boleh saja kemudian kita destrukturisasi props menjadi sebagai berikut:

```
const Hello = (props) => {
    const { name } = props
    return `Hello ${name}!`
}
```

Atau bisa juga didestrukturisasi langsung pada argumen fungsinya.

```
const Hello = ({ name }) => {
    return `Hello ${name}!`
}
```

Cara terakhir ini tentu lebih pendek dibanding cara lainnya.

Bagaimana jika kita ingin menambahkan variabel props baru? misalnya kita ingin menambahkan variabel props `gretting`. Ya cukup dengan menambahkan item baru pada objek props yang didestrukturisasi.

```
const Hello = ({ name, gretting }) => {
    return `Hello ${name}!, ${gretting}`
}
```

Props Bersifat ReadOnly

Props pada komponen ReactJS bersifat read only, sehingga kita hanya boleh menggunakannya tanpa boleh memodifikasinya.

Perhatikan kode berikut:

```
const Hello = (props) => {
    props.name = 'Transformer'
    return `Hello ${props.name}!`
}
```

Item name pada objek props diubah nilainya menjadi `Transformer`, maka ketika dijalankan akan muncul error `Uncaught TypeError: Cannot assign to read only property 'name' of object`. Hal ini disebabkan, objek props telah diset sebagai objek readonly atau immutable.

Yap, tentu saja kalo kita destrukturisasi maka kita bisa saja mengubah nilai item `name` dengan aman:

```
const Hello = ({ name }) => {
    name = 'Transformer'
    return `Hello ${name}!`
}
```

Namun secara konsep, props ini sebaiknya diperlakukan secara read-only. Lalu bagaimana jika ada kasus di mana kita perlu memodifikasi nilai dari props? Jawabannya adalah tidak dengan menggunakan props, namun untuk kasus tersebut ReactJS menggunakan cara lain yaitu state.

Nilai Default Pada Props

Kita bisa juga memberikan initia value (nilai default) pada props, sehingga jika props tidak didefinisikan nilainya maka komponen akan merender nilai defaut dari propsnya.

Untuk melakukannya, kita bisa menggunakan properti `defaultProps`. Perhatikan contoh berikut:

```
const Hello = ({ name }) => {
    return `Hello ${name}!`
}

Hello.defaultProps = {
    name: 'anonim'
}

export default Hello
```

Jika props name tidak didefinisikan saat memanggil komponen Hello: `<Hello />`. Maka akan menghasilkan tampilan `Hello anonim!`.

Menggabungkan Komponen

Tipe Data Pada Props

Tipe data dari item pada props tidak hanya string dan angka saja, namun juga bisa menggunakan tipe data yang lebih komplek seperti array dan objek. Terkait tipe data pada props, akan dibahas lebih detail pada bagian selanjutnya.

Hook

Perkenalan Hook

React *Hook* atau *Hook* merupakan fitur yang tergolong belum lama muncul di ReactJS. Pertama kali dirilis pada ReactJS versi 16.8.0.

Sehingga tidak heran banyak *developer* yang belum terlalu familiar dengan *Hook* meskipun mereka sudah lama menggunakan ReactJS.

Hook merupakan cara agar kita bisa menggunakan atau mengakses `state` dan ReactJS `lifecycle` dengan *functional component*.

Sebelum ada *Hook* kita tidak bisa menggunakan `state` maupun mengakses `lifecycle` ReactJS dari *functional component*. Ketika itu hanya `class component` yang bisa mengakses `state` dan `lifecycle`, sedangkan *functional component* hanya dipakai sebagai `stateless component` yaitu *component* yang tidak memiliki *State*.

Komponen *stateless*, semacam itu hanya digunakan untuk menerima *props* dan menampilkan UI sesuai dengan *props* tersebut.

Agar memudahkan kamu memahaminya mari kita lihat contoh *stateless component* berikut ini:

```
import React from 'react';

const Button = ({text}) => {
    return <button> {text} </button>
}
```

Komponen di atas menerima *props* `text` dan menampilkan text tersebut sebagai `<button/>`.

Bagaimana caranya jika kita ingin menampilkan `text` dan sebuah `counter` yang bernilai 0 di awal `render` lalu setiap `<button/>` tersebut diklik `counter` tersebut akan bertambah satu.

Untuk membuat `counter` semacam itu, kita perlu menggunakan *state* untuk menyimpan nilai `counter` dan mengupdatenya setiap kali `button` diklik.

Sayangnya, sebelum adanya *Hook* hal tersebut tidak memungkinkan untuk dilakukan menggunakan *functional component* yang notabene adalah *stateless component* .

Maka, kita harus mengubah terlebih dahulu *stateles component* di atas menjadi *class component* seperti ini:

```
import React, { Component } from 'react';

class Button extends Component{
  state = {
     counter: 0
```

```
  }

  handleClick = () => {
    this.setState({
      counter: this.state.counter + 1
    })
  }

  render(){
     return <button onClick={handleClick}>
      {this.props.text} {this.state.counter}
     </button>
  }
}
```

Kode di atas adalah cara untuk membuat komponen menggunakan ES6 `class` karena kita ingin menggunakan *state*.

useState

Semenjak ReactJS versi 16.8.0 kita bisa menggunakan *functional component* dan tetap bisa mengakses *state*. Sekarang kita akan mengubah *class component* di atas menjadi *stateful functional component* menggunakan *Hook* seperti ini:

```
import React, {useState} from 'react';

const Button = ({text}) => {

   const [counter, setCounter] = useState(0);

   const handleClick = () => setCouter(counter + 1);

   return <button onClick={handleClick}>
     {text} {counter}
   </button>
}
```

Kode di atas akan menghasilkan komponen yang sama seperti pada contoh sebelumnya yang menggunakan *class component*.

Untuk mengakses fitur *state* kita menggunakan *Hook* yang akan paling sering kita pakai kedepannya yaitu `useState`.

`useState` sudah kamu pelajari pada bab Eksplorasi Komponen sehingga kami tidak perlu mengulangi lagi pembahasan detail mengenai `useState`.

Fungsi *Updater*

Satu hal yang belum dibahas adalah cara lain menggunakan fungsi `updater`, sebelumnya kita tahu bahwa kita bisa mengupdate nilai *State* `counter` dengan cara seperti ini:

```
setCounter(counter + 1);
```

Yaitu kita memberikan nilai baru untuk *State* melalui fungsi `updater` --`setCounter`--. Selain memberikan nilai baru, kita juga bisa memberikan fungsi sebagai argumen pada `setCounter`, seperti ini:

```
const stateUpdater = function(currentCounter){
  return currentCounter + 1
}

setCounter(stateUpdater);
```

Atau langsung dimasukan ke `setCounter` seperti ini:

```
setCounter(function(currentCounter){ return currentCounter + 1 });
```

Fungsi yang diberikan sebagai argumen pada `setCounter` harus mengembalikan sebuah nilai, nilai itulah yang akan menjadi nilai terbaru untuk *State* `counter`.

Selain itu fungsi tersebut juga menerima nilai dari *State* `counter` saat ini, sehingga kita bisa menggunakannya untuk melakukan `increment` misalnya, dengan mengembalikan nilai saat ini ditambah 1, seperti pada contoh di atas.

Fungsi di atas terlihat agak susah dibaca, umumnya _developer menggunakan sintak ringkas ES6 seperti dibawah ini untuk hasil yang sama seperti contoh sebelumnya:

```
setCounter( counter => counter + 1 );
```

Tentu kita bisa meringkasnya lagi menjadi seperti ini:

```
setCounter(c => c + 1);
```

Tapi harus diperhatikan jika dibaca oleh *developer* lain, kira-kira lebih enak di baca yang mana? Lebih pendek belum tentu lebih enak dibaca.

Saya ingin memberi tahu bahwa kita bebas memberikan nama untuk nilai saat ini *State* `counter` jika menggunakan fungsi *updater*, meskipun nilainya sama dengan *State* `counter` tapi variabel tersebut berada di lingkup berbeda dan tidak merefensikan variabel *State* `counter` secara langsung, sehingga kita bebas memberikan nama lain, seperti contoh tadi `currentCounter`, `counter` atau bahkan `c`.

Struktur Projek & Routing Aplikasi

Pendahuluan

Materi yang akan dibahas pada bab ini meliputi:

- Struktur Projek Aplikasi
- Routing Aplikasi

Dengan mempelajari materi ini maka diharapkan kita dapat memahami bagaimana struktur projek aplikasi yang cocok untuk projek kita sehingga memudahkan kita dalam mengembangkan aplikasi.

Disamping itu diharapkan kita juga dapat menyusun routing aplikasi dengan baik sehingga pengguna aplikasi dapat mengakses halaman atau fitur dari aplikasi yang kita kembangkan dengan mudah.

Beberapa materi yang akan dibahas pada bab ini membutuhkan dasar pengetahuan tentang komponen dan hooks yang telah dibahas pada bab sebelumnya.

Oleh karena itu pemahaman yang baik tentang topik tersebut akan sangat membantu memudahkan kita menguasai materi pada bab ini.

Tutorial pada bab ini akan menggunakan ReactJS dan tentu berbasis CRA (Create React App). Oleh karena itu silakan diinstalasi terlebih dahulu.

State Management

Pendahuluan

Materi yang dibahas pada bab ini adalah management state pada aplikasi berbasis ReactJS dengan menggunakan pustaka Redux.

Publikasi Aplikasi

Pendahuluan

Publikasi aplikasi React yang dibangun menggunakan `create-react-app` sangatlah mudah.

Secara umum, kita hanya perlu melakukan *build* terhadap *source code* aplikasi. Kemudian `create-react-app` akan memproses *source code* kita menjadi *file-file* pada folder `build`.

File-file tersebut tinggal kita *upload* saja ke penyedia hosting yang kita pakai.

Sementara itu *file-file source code* kita tidak perlu diupload, *file source code* tersebut hanya perlu kita simpan di *repository*.

Dan sebaliknya folder `build` tidak perlu diikutkan di *repository* projek kita. Kita bisa mengabaikan folder `build` dari *git* dengan menambahkannya di *file* `.gitignore`, seharusnya sudah secara default masuk di `.gitignore` saat kita pertama kali membuat projek dengan `create-react-app`.

Di bab ini kita akan berlatih mempublikasikan aplikasi frontend React yang dibuat menggunakan `create-react-app` menggunakan platform Vercel, Netlify, Firebase Hosting & Github Page.

Kamu bisa menggunakan projek baru atau menggunakan projek aplikasi Grid yang sudah kita buat.