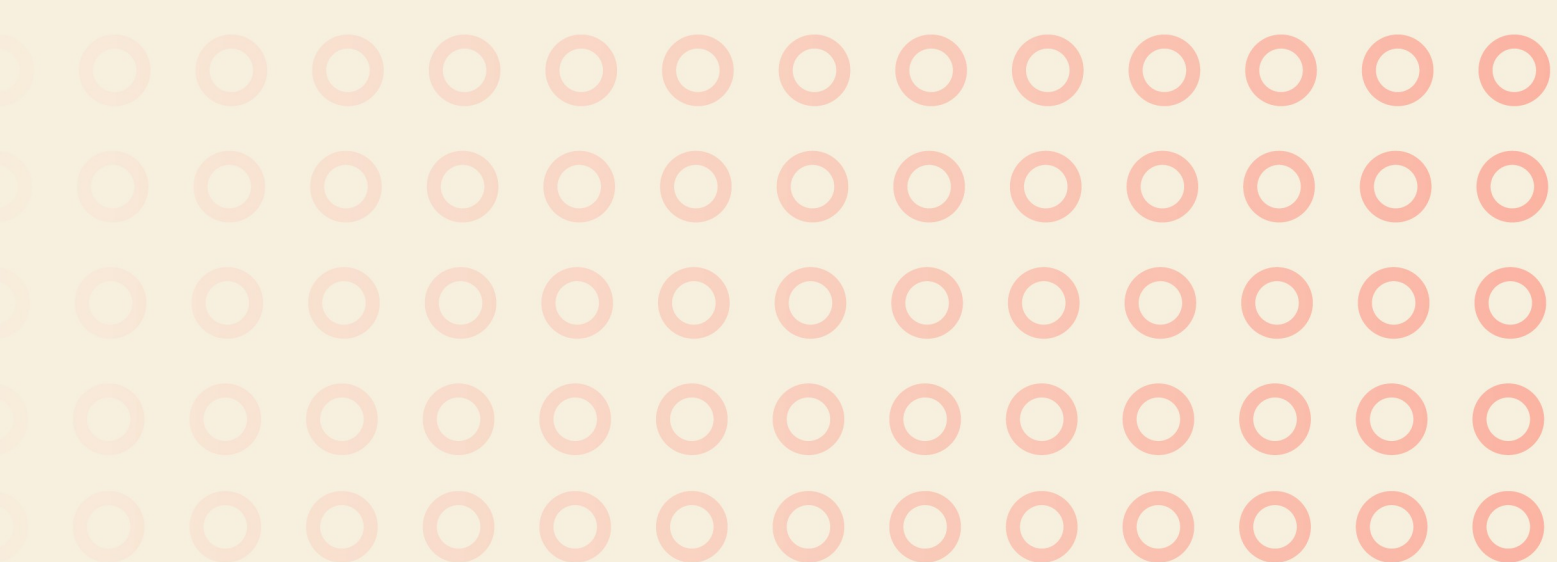




COURSE 4

React Fundamental



Modul Sekolah Fullstack Cilsy

Hak Cipta © 2020 PT. Cilsy Fiolution Indonesia

Hak Cipta dilindungi Undang-Undang. Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronis maupun mekanis, termasuk mecropy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis dan Penerbit.

Penulis : Michael Fernando Padang, Saefulloh Maslul
Editor : Muhammad Fakhri Abdillah, Iqbal Ilman Firdaus
Revisi Batch 4

Penerbit : **PT. Cilsy Fiolution Indonesia**
Web Site : <https://cilsyfiolution.com> , <https://sekolahFullstack.com>

Sanksi Pelanggaran Pasal 113 Undang-undang Nomor 28 Tahun 2014 tentang Hak Cipta

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam pasal 9 ayat (1) huruf i untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan atau pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).
2. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak cipta melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf c, huruf d, huruf f, dan atau huruf h, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah)
3. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf a, huruf b, huruf e, dan atau huruf g, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah)



Daftar Isi

Daftar Isi.....	3
6. Fundamental React.....	5
Learning Outcomes.....	5
Outline Materi.....	5
6.1. The Build Workflow.....	7
6.2. Using Create React App.....	8
6.2.1. Project initiation.....	8
6.2.2. Configuration.....	8
6.3. Understanding the Folder Structure.....	9
6.3.1. Node_modules.....	9
6.3.2. Public.....	10
6.3.3. Src.....	10
6.4. Understanding JSX.....	10
6.5. Components.....	11
6.6. Creating a Functional Component.....	11
6.7. Working with Reusable Components.....	12
6.8. Outputting Dynamic Content.....	12
6.9. Rendering Content Conditionally.....	13
6.10. Understanding Props.....	14
6.11. Understanding Children Props.....	15
6.12. Understanding State.....	16
6.13. Outputting Lists.....	17
6.14. Handling Events with Methods.....	18
6.15. More About Event.....	19
6.15.1. Composition Events.....	19
6.15.2. Keyboard Events.....	20
6.15.3. Focus Events.....	20
6.15.4. Mouse Events.....	21
6.15.5. Selection Events :.....	22



6.15.6. Touch Events:.....	22
6.15.7. UI Events.....	22
6.15.8. Wheel Events.....	22
6.15.9. Media Events.....	23
6.15.10. Image Events.....	23
6.15.11. Animation Events.....	23
6.15.12. Transition Events.....	23
6.15.13. Other Events.....	24
6.16. Manipulating the State.....	24
6.17. Passing Method Between Components.....	25
6.18. Adding Two Way Binding.....	26
6.19. Exercise.....	30

6.

Fundamental React

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Dapat Memahami Fundamental React & JSX
2. Dapat Memahami dan menggunakan Sintak-sintak dasar React & JSX

Outline Materi

1. The Build Workflow
2. Using Create React App
3. Understanding the Folder Structure
4. Understanding JSX
5. Components
6. Create a Function Component
7. Working with Reusable Component
8. Outputting Dynamic Content
9. Rendering Content Conditionally
10. Understanding Props
11. Understanding Children Props
12. Understanding State
13. Outputting Lists

14. Handling Events with Methods

15. More About Events

16. Manipulating the State

17. Passing Method Between Components

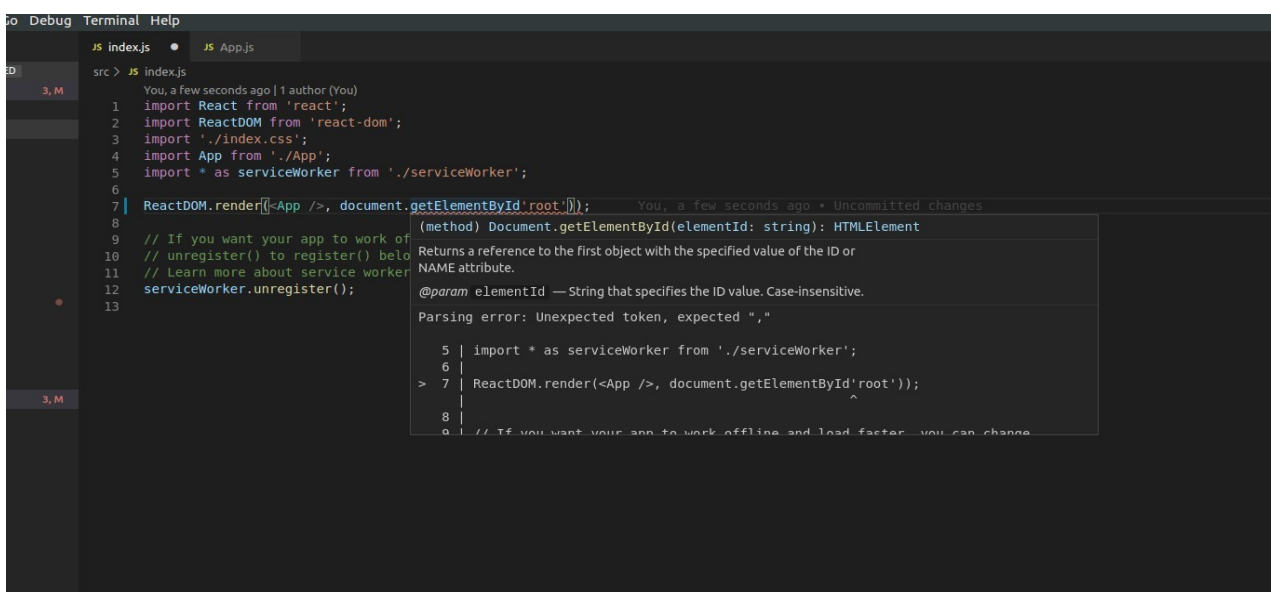
18. Add Two Way Binding

6.1. The Build Workflow

Build workflow digunakan untuk mengoptimasi kode anda. Untuk menjadikan lebih baik, aplikasi kita harus dapat menghasilkan kode kecil yang dapat dioptimalkan untuk meningkatkan kinerja aplikasi tersebut.

Alasan penting lainnya adalah untuk mendapatkan manfaat dengan menggunakan fitur javascript generasi selanjutnya. Javascript berkembang dari waktu ke waktu dengan penambahan fitur baru yang mungkin tidak didukung di semua browser. Memiliki sebuah tools seperti *babel* memungkinkan kita untuk menggunakan fitur-fitur tersebut dengan baik, karena dapat bertanggung jawab untuk menerjemahkan kode kita ke kode yang didukung oleh browser yang ditunjuk nantinya. Alasan lainnya menyangkut produktivitas. CSS *auto-prefixing* pada fitur javascript generasi berikutnya memungkinkan kita untuk mencapai dukungan browser maksimum untuk fitur CSS. Jika kita mencoba menambahkannya secara manual, itu akan merepotkan.

Aspek lainnya dari produktivitas sebuah *tools* yang disediakan adalah Linter. Memperoleh linter dalam kode kita IDE akan menghemat waktu kita untuk menemukan kesalahan sebelum kita menjalankan kode tersebut. Berikut contoh yang menggambarkan cara kerja linter.



```

src > JS index.js
1 You, a few seconds ago | 1 author (You)
2 import React from 'react';
3 import ReactDOM from 'react-dom';
4 import './index.css';
5 import App from './App';
6 import * as serviceWorker from './serviceWorker';
7 ReactDOM.render(<App />, document.getElementById('root'));
8 // If you want your app to work of
9 // unregister() to register() belo
10 // Learn more about service worker
11 serviceWorker.unregister();
12
13
You, a few seconds ago • Uncommitted changes
(method) Document.getElementById(elementId: string): HTMLElement
Returns a reference to the first object with the specified value of the ID or
NAME attribute.
@param elementId — String that specifies the ID value. Case-insensitive.
Parsing error: Unexpected token, expected "\",
5 | import * as serviceWorker from './serviceWorker';
6 |
> 7 | ReactDOM.render(<App />, document.getElementById('root'));
  |                                     ^
8 |
9 | // If you want your app to work offline and load faster... you can change

```

6.2. Using Create React App

Create React App adalah React *boilerplate generator* yang diinisiasi oleh Facebook. *create-react-app* ini menyediakan lingkup pengembangan yang dikonfigurasi untuk kemudahan penggunaan dengan pengaturan minimal, termasuk :

1. ES6 dan JSX transpilation
2. Dev server dengan hot module reloading
3. Code linting
4. CSS auto-prefixing
5. Build script with JS, CSS dan image bundling, dan sourcemaps
6. Jest testing framework

6.2.1. Project initiation

Pertama, jalankan *create-react-app* menggunakan perintah berikut.

```
npx create-react-app greetings
```

Selanjutnya pindah ke direktori yang telah dibuat dan jalankan start script project:

```
cd greetings  
npm start
```

Browser bawaan anda secara otomatis akan terbuka dengan aplikasi baru yang sudah di render pada halaman browser.

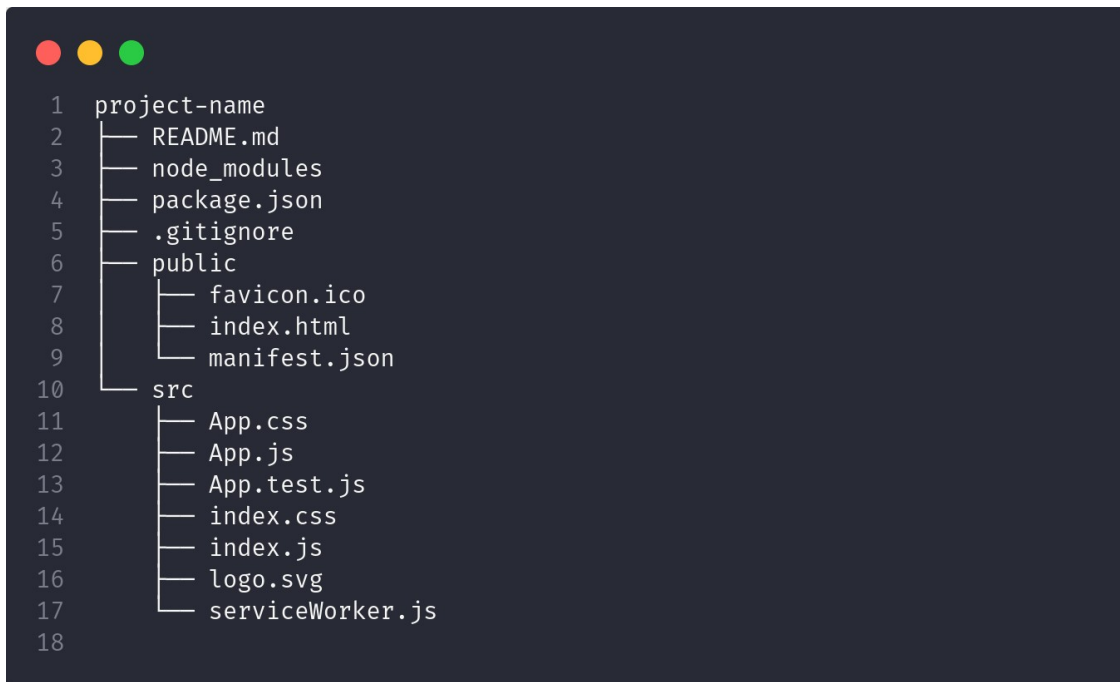
6.2.2. Configuration

create-react-app sengaja tidak dapat dikonfigurasi secara *default*. Jika dibutuhkan penggunaan yang tidak standar, contoh untuk menggunakan bahasa css yang dikompilasi seperti SASS, maka perintah *eject* dapat digunakan


```
npm run eject
```

6.3. Understanding the Folder Structure

Ketika kita membuat project baru dengan create-react-app, akan ada struktur folder seperti ini.



6.3.1. Node_modules

Folder ini berisi *library* javascript yang sudah disediakan secara otomatis. *Library* tersebut yang nantinya akan digunakan dalam membuat project react.

Selain itu kita juga dapat menginstal *library* lain seperti bootstrap dan react-router-dom dengan cara:

```
npm install --save nama_library
```

contoh:

```
npm install --save react-bootstrap bootstrap
```

6.3.2. Public

Folder public berisi konfigurasi file html dan favicon. Apabila kita membuka file `index.html` terdapat kode.

```
<div id="root"></div>
```

Pada kode tersebut terdapat id root yang nantinya digunakan untuk menempatkan hasil render dari React.

6.3.3. Src

Pada folder src kita dapat menemukan banyak sekali konfigurasi file React. Kita pahami terlebih dahulu *flow* dari React ini. Pertama kita mempunyai `App.js`. File ini yang akan di-*render* oleh `index.js` untuk diletakan pada element dengan id root. Kemudian file `App.js` ini yang nantinya menjadi inti dari kode kita. Kita dapat mengubahnya menjadi project yang kita inginkan.

6.4. Understanding JSX

JSX merupakan kependekan dari *Javascript Syntax Extension*. JSX merupakan *syntax* yang digunakan oleh React sehingga kita dapat menuliskan HTML/XML (bukan murni html tetapi sesuatu yang mirip "html") pada React. JSX dan kode EcmaScript akan ditransformasi oleh `babel` untuk dikonversi menjadi kode javascript yang sebenarnya (standar kode javascript). Dengan menggunakan JSX penulisan elemen-elemen react akan menjadi sangat mudah. Pada JSX kita menuliskan layaknya kode html seperti biasanya. Hanya saja kita dapat menambahkan seperti *javascript expression*, mengakses fungsi, mengakses properti dan lain sebagainya melalui JSX. Nantinya kode JSX ini akan ditransformasikan oleh `babel` menjadi kode standar javascript. Contoh dari *syntax* JSX adalah,

```

1 function App() {
2   return (
3     <div>
4       <h1>Hello World</h1>
5     </div>
6   )
7 }

```

6.5. Components

Komponen adalah *the core building block* dari React. Pada dasarnya, React benar-benar hanya sebuah *library* untuk membuat komponen. Oleh karena itu, aplikasi React dapat digambarkan sebagai komponen pohon yang memiliki satu komponen root ("App") dan kemudian komponen-komponen tersebut memiliki anak komponen yang tidak terbatas.

Setiap komponen perlu mengembalikan atau merender beberapa kode JSX, itu mendefinisikan kode HTML React mana yang harus di-*render* oleh DOM sebenarnya. JSX bukan HTML tetapi memang sangat mirip. Perbedaan dapat dilihat ketika kita lebih detail lagi dalam memperhatikannya (misalnya `className` di JSX dengan `class` di "HTML biasa").

6.6. Creating a Functional Component

Functional Component adalah komponen yang didefinisikan menggunakan function yang sangat simpel, tanpa harus belajar `class`, `this` context, `state`, dan lain sebagainya. Syaratnya hanya cukup mengerti bagaimana function javascript bekerja.

```

1 const greeting = () => {
2   return "Hello World"
3 }
4
5 greeting()

```

Di atas adalah contoh simple dari definisi function pada javascript. Ketika kita menjalankan function `greeting()`, maka mendapat nilai balikan yaitu string "Hello world". Sedangkan saat kita konversikan function tersebut dalam bentuk *Functional Component* maka akan menjadi seperti ini:

```
1 import React from 'react'
2
3 const Greeting = () => {
4   return (
5     <h1>Hello World!</h1>
6   )
7 }
8
9 export default Greeting
10
```

Hampir sama dengan deklarasi dilakukan pada javascript hanya saja function `Greeting()` mempunyai nilai balikan berupa JSX Tag atau mudahnya syntax HTML/XML di javascript yang nantinya akan di-*render* menjadi *React element*.

6.7. Working with Reusable Components

Sebuah kumpulan komponen sangat baik karena kita dapat memfokuskan kode kita di setiap file dan dengan demikian membuatnya lebih mudah di-*maintenance*. Tidak memasukkan semuanya kedalam file `App.js`. Ini yang kemudian membuat komponen dapat digunakan kembali dan dapat dikonfigurasi.

6.8. Outputting Dynamic Content

```
1 const Greeting = () => {
2   return (
3     <p>Hello World, I'm {Math.floor(Math.random() * 1000)} years old.</p>
4   )
5 }
6
```

Dengan kita membungkus *dynamic content* apapun yang ingin kita tambahkan pada komponen dalam kurung kurawal. Ini juga menunjukkan bahwa React dapat menempatkan *dynamic content* didalam JSX.

6.9. Rendering Content Conditionally

Konsep *rendering content conditionally* yaitu kita dapat me-*render* konten di React sesuai kondisi yang kita tentukan sebelumnya.

```

1  import React from 'react'
2
3  const Greeting = () => {
4    const isRender = false
5
6    if (isRender === true) {
7      return (
8        <h1>Hello World</h1>
9      )
10   } else {
11     return (
12       ''
13     )
14   }
15 }
```

Pada contoh kode di atas, kita me-*render* konten sesuai nilai dari variabel `isRender`. Apabila `isRender` bernilai `true`, maka React akan merender konten:

```

1  <h1>Hello World</h1>
```

Sedangkan, apabila nilai `isRender` `false`, maka React akan me-*render* string kosong (''). Kode di atas dapat buat lebih simpel menjadi seperti ini:

```

1  import React from 'react'
2
3  const Greeting = () => {
4    const isRender = false
5
6    return isRender ? (
7      <h1>Hello World</h1>
8    ) : (
9      ''
10   )
11 }
12
13 export default Greeting
14
15

```

6.10. Understanding Props

Prop singkatan dari *Property*. Ini mirip seperti atribut pada tag HTML. Dalam pembuatannya, jika dalam *functional component* maka prop ini adalah parameternya. Contohnya kita melakukan penambahan informasi nama dan umur pada komponen `Greeting`, dengan begitu React mengambil seluruh atribut yang sudah dikirim dan memberi kita akses di dalam komponen penerima pada objek yang bernama `props` (dalam kasus ini pada komponen `Greeting`). Kita sebenarnya dapat memberi nama apapun pada objek tersebut nantinya. Contoh seperti berikut:

```

1  import React from 'react';
2  import Greeting from './Greeting';
3
4  const App = () => {
5    return (
6      <div>
7        <h1>Greeting from React App</h1>
8        <Greeting name='Michael' age='19' />
9        <Greeting name='Budi' age='21' />
10       <Greeting name='Anne' age='22' />
11      </div>
12    );
13  }
14
15  export default App;
16

```

Selanjutnya kita perlu membuat file bernama Greeting.js dan menambahkan parameter props pada komponen Greeting untuk menampilkan properti nama dan umur tersebut.

```

1  import React from 'react';
2
3  const Greeting = (props) => {
4    return (
5      <p>Hello World! My name is {props.name} and I am {props.age} years old.</p>
6    );
7  }
8
9  export default Greeting;
10

```

6.11. Understanding Children Props

Setiap komponen mempunyai props bernama children. props.children akan menghasilkan return semua yang ada di antara opening komponen dan closing JSX tags. Sejauh ini, semua komponen yang kita lihat mempunyai tag *self-closing*, seperti <Greeting />. Mereka yang tidak mempunyai tag penutup, kita bisa menulis <Greeting></Greeting>, dan itu akan tetap berjalan. props.children akan mereturn apapun antara <Greeting> dan </Greeting>. Contoh:

```

1 <div>
2   <Heading>Greeting from React App</Heading>
3   <Greeting name='Michael' age='19' />
4   <Greeting name='Budi' age='21' />
5   <Greeting name='Anne' age='22' />
6 </div>

```

Pada file App.js, kita mengubah element pada h1 menjadi komponen Heading. Kemudian pada file Heading.js kita merender props.children.

```

1 import React from 'react'
2
3 const Heading = (props) => {
4   return (
5     <h1>{props.children}</h1>
6   )
7 }
8
9 export default Heading

```

code: [Heading.js](#)

6.12. Understanding State

State adalah data *private* sebuah komponen. Data ini hanya di inialisasi dan digunakan untuk komponen tersebut dan tidak bisa diakses dari komponen lain. Komponen dapat merubah statenya itu sendiri. Untuk menggunakan state di *functional component*, kita perlu menggunakan *function hooks* dari React yang bernama useState. Contohnya seperti dibawah ini.


```

1  import React, { useState } from 'react';
2  import Greeting from './Greeting';
3  import Heading from './Heading';
4
5  const App = () => {
6    const initialState = [
7      {name: 'Michael', age: 19},
8      {name: 'Budi', age: 21},
9      {name: 'Anne', age: 22}
10   ]
11
12   const [persons, setPersons] = useState(initialState)
13
14   return (
15     <div>
16       <Heading>Greeting from React App</Heading>
17       <Greeting name={persons[0].name} age={persons[0].age} />
18       <Greeting name={persons[1].name} age={persons[1].age} />
19       <Greeting name={persons[2].name} age={persons[2].age} />
20     </div>
21   );
22 }
23
24 export default App;
25

```

code: [App.js](#)

Pada line 12, kita menggunakan *function hooks* `useState` sebagai inialisasi state. Variabel `initialState` adalah nilai awal dari state `persons`. Sedangkan `setPersons` merupakan function untuk mengubah state `persons` itu sendiri.

6.13. Outputting Lists

Kita telah berhasil membuat list `Greeting` sebanyak tiga buah dengan cara mengulang kode pada komponen `Greeting` sebanyak tiga kali. Hal ini berfungsi sebagaimana mestinya. Namun cara seperti ini adalah cara yang salah. Karena ketika ada penambahan atau pengurangan item di array `persons`, kita tidak bisa mengubah tampilannya menjadi dinamis sesuai item dari `persons`.

Untuk membuatnya menjadi dinamis sesuai nilai array `persons`, kita dapat memanfaatkan *array iteration* pada javascript. Sehingga kode di atas menjadi:



```

1 <div>
2   <Heading>Greeting from React App</Heading>
3   {
4     persons.map(person => (
5       <Greeting key={person.name} name={person.name} age={person.age} />
6     ))
7   }
8 </div>

```

code: [App.js](#)

Jika kita perhatikan, kita hanya menggunakan fungsi map yang ada di javascript untuk membuat perulangan sesuai item dari array persons. Kemudian kita membuat return-nya berupa elemen Greeting. Namun terdapat satu perbedaan antara elemen Greeting sebelumnya dengan kode di atas, kita menambahkan satu props baru bernama key.

Fungsi key membantu React mengidentifikasi item mana yang berubah, ditambahkan, atau dihapus. Key juga harus diberikan ke elemen di dalam array untuk memberikan elemen identitas yang stabil. Pada contoh di atas kita membuat key berdasarkan nama dari person, untuk kedepannya akan lebih baik jika menggunakan id ataupun unique key lainnya.

6.14. Handling Events with Methods

React handle *event* menggunakan atribut sama halnya seperti html pada umumnya. Hanya saja terdapat aturan yang khusus pada penggunaan atribut tersebut yakni :

1. Jika pada html biasa *event* ditulis dengan huruf kecil maka pada react penulisannya menggunakan *camelCase*.
2. Jika pada html kita memberikan *expression* berupa string yang mengarah pada suatu fungsi tertentu, maka pada react memberikan fungsi *event handler*.

Kita akan membuat suatu *event handler* sederhana yakni dari sebuah button yang akan menampilkan alert berupa "Hello world". Pertama kita siapkan terlebih dahulu elemen button disertai dengan *event handler*-nya. Berikut adalah contoh penggunaan metode pada *event handler*.



```

1  const showMessage = () => {
2    alert('Hello World')
3  }
4
5  return (
6    <div>
7      <Heading>Greeting from React App</Heading>
8      {
9        persons.map(person => (
10         <Greeting key={person.name} name={person.name} age={person.age} />
11       ))
12      }
13      <button onClick={showMessage}>Show Message</button>
14    </div>
15  );

```

code: [App.js](#)

Pada kode di atas, kita membuat function bernama showMessage yang berfungsi untuk menampilkan alert "Hello World". Function tersebut kemudian dipanggil pada elemen button pada event onClick. Ketika kita tekan tombol Show Message maka alert tersebut akan tampil.

6.15. More About Event

Untuk event lainnya kita dapat langsung menemukan dari referensi <https://reactjs.org/docs/events.html#supported-events>.

Clipboard Events

Event names:

onCopy onCut onPaste

Properties:

DOMDataTransfer clipboardData

6.15.1. Composition Events

- Event names:

```
onCompositionEnd onCompositionStart onCompositionUpdate
```

- Properties:

```
string data
```

6.15.2. Keyboard Events

- Event names:

```
onKeyDown onKeyPress onKeyUp
```

- Properties:

```
boolean altKey  
number charCode  
boolean ctrlKey  
boolean getModifierState(key)  
string key  
number keyCode  
string locale  
number location  
boolean metaKey  
boolean repeat  
boolean shiftKey  
number which
```

6.15.3. Focus Events

- Event names:

```
onFocus onBlur
```

- Properties:

```
DOMEventTarget relatedTarget
```

```
Form Events
```



- Event names:

```
onChange onInput onInvalid onSubmit
```

6.15.4. Mouse Events

- Event names:

```
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter  
onDragExit
```

```
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter  
onMouseLeave
```

```
onMouseMove onMouseOut onMouseOver onMouseUp
```

The `onMouseEnter` and `onMouseLeave` events propagate from the element being left to the one being entered instead of ordinary bubbling and do not have a capture phase.

- Properties:

```
boolean altKey  
number button  
number buttons  
number clientX  
number clientY  
boolean ctrlKey  
boolean getModifierState(key)  
boolean metaKey  
number pageX  
number pageY  
DOMEventTarget relatedTarget  
number screenX  
number screenY  
boolean shiftKey
```

6.15.5. Selection Events :

- Event names:

```
onSelect
```

6.15.6. Touch Events:

- Event names:

```
onTouchCancel onTouchEnd onTouchMove onTouchStart
```

- Properties:

```
boolean altKey  
DOMTouchList changedTouches  
boolean ctrlKey  
boolean getModifierState(key)  
boolean metaKey  
boolean shiftKey  
DOMTouchList targetTouches  
DOMTouchList touches
```

6.15.7. UI Events

- Event names:

```
onScroll
```

- Properties:

```
number detail  
DOMAbstractView view
```

6.15.8. Wheel Events

- Event names:

```
onWheel
```

- Properties:

```
number deltaMode  
number deltaX  
number deltaY  
number deltaZ
```

6.15.9. Media Events

- Event names:

```
onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied  
onEncrypted  
onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause  
onPlay  
onPlaying onProgress onRateChange onSeeked onSeeking onStalled  
onSuspend  
onTimeUpdate onVolumeChange onWaiting
```

6.15.10. Image Events

- Event names:

```
onLoad onError
```

6.15.11. Animation Events

- Event names:

```
onAnimationStart onAnimationEnd onAnimationIteration
```

- Properties:

```
string animationName  
string pseudoElement  
float elapsedTime
```

6.15.12. Transition Events

- Event names:

```
onTransitionEnd
```

- Properties:

```
string propertyName  
string pseudoElement  
float elapsedTime
```

6.15.13. Other Events

- Event names:

```
onToggle
```

6.16. Manipulating the State

React tidak memperbolehkan untuk mengupdate state secara langsung. Untuk memperbarui state, dapat menggunakan method yang telah disediakan pada useState. Ingat ketika membuat useState, kita menerima dua data, yaitu data state dan method untuk memperbarui state itu sendiri.

```
1 const [persons, setPersons] = useState(initialState)
```

Ketika state diperbarui, props juga ikut diperbarui karena mereka terikat pada properti di dalam state dan properti tersebut diteruskan ke komponen lain.


```

1  const onChangePersons = () => {
2    setPersons([
3      {name: 'Kesya', age: 21},
4      {name: 'Adi', age: 19},
5      {name: 'Silvy', age: 22}
6    ])
7  }
8
9  return (
10   <div>
11     <Heading>Greeting from React App</Heading>
12     {
13       persons.map(person => (
14         <Greeting key={person.name} name={person.name} age={person.age} />
15       ))
16     }
17     <button onClick={showMessage}>Show Message</button>
18     <button onClick={onChangePersons}>Change Persons</button>
19   </div>
20 );

```

Pada kode di atas. kita membuat sebuah function `onChangePersons`, yang akan mengeksekusi method `setPersons` dan mengganti *value* dari `persons`. Function `onChangePersons` kemudian dipanggil pada event `onClick` pada button `Change Persons`.

6.17. Passing Method Between Components

Sebelumnya kita menggunakan *event handler* untuk mengubah state dan mengubah informasi yang ditampilkan pada *user interface*. Saat ini kita ingin melakukan perubahan itu tidak hanya pada komponen asli. Tetapi dengan mempassingnya pada komponen lain, sehingga komponen lain dapat memanipulasi state *parent*.

Pada contoh kali ini, kita akan membuat komponen button secara terpisah, kemudian mengeksekusi function `onChangePersons` pada komponen terpisah tadi. Pertama buatlah file bernama `Button.js`.



```

1  const Button = (props) => {
2    return (
3      <button onClick={props.onChangePersons}>Change Persons</button>
4    )
5  }
6
7  export default Button

```

code: [Button.js](#)

Pada komponen Button, kita membuat sebuah button yang memiliki event onClick yaitu mengeksekusi props bernama onChangePersons. Kemudian pada file App.js, kita gunakan komponen button tersebut, serta mem-*passing* sebuah props bernama onChangePersons.

```

1  <div>
2    <Heading>Greeting from React App</Heading>
3    {
4      persons.map(person => (
5        <Greeting key={person.name} name={person.name} age={person.age} />
6      ))
7    }
8    <button onClick={showMessage}>Show Message</button>
9    <Button onChangePersons={onChangePersons} />
10 </div>

```

code: [App.js](#)

Hasil kode kita akan sama dengan sebelumnya, namun sekarang kita sudah bisa memanipulasi state dari komponen lain.

6.18. Adding Two Way Binding

Konsep *Two-Ways data binding* adalah setiap perubahan yang kita lakukan di Javascript maka akan berpengaruh terhadap tampilan (view layer) yang terdapat di dalam kode HTML kita, dan sebaliknya perubahan yang terjadi pada tampilan (view layer) akan berpengaruh juga terhadap value yang terdapat di dalam Javascript nya.



Pada contoh kali ini, kita akan membuat state bernama person yang berisi name dan age. State tersebut akan dimanipulasi oleh tampilan (elemen input). Kemudian state person akan kita tambahkan pada persons sehingga *list persons* akan berubah sesuai dengan inputan yang kita masukan.

Pertama buat terlebih dahulu state person.

```
1  const initialPerson = {
2    name: '',
3    age: ''
4  }
5
6  const [persons, setPersons] = useState(initialState)
7  const [person, setPerson] = useState(initialPerson)
```

code: [App.js](#)

Kemudian buat function untuk mengubah person,

```
1  const onChangeInput = (e) => {
2    setPerson({
3      ...person,
4      [e.target.name]: e.target.value
5    })
6  }
```

code: [App.js](#)

Function `onChangeInput` berfungsi untuk mengubah person menggunakan method `setPerson`. Jika kita perhatikan line 3 dan 4, kita memperbarui person menggunakan *value* person sebelumnya dan data yang kita kirimkan melalui *event handler* `onChange`.

Kita menamakan properti state yang akan kita ubah sebagai `e.target.name` (menggunakan properti `name` pada elemen `input`) dan nilainya sebagai `e.target.value` (menggunakan properti `value` pada elemen `input`).

Selanjutnya kita buat dua elemen input untuk memperbarui state person tersebut.

```

1  <div>
2    <div>
3      <label htmlFor="name">Name</label>
4      <input
5        type="text"
6        name="name"
7        id="name"
8        onChange={onChangeInput}
9        value={person.name}
10     />
11  </div>
12  <div>
13    <label htmlFor="age">Age</label>
14    <input
15      type="number"
16      name="age"
17      id="age"
18      onChange={onChangeInput}
19      value={person.age}
20    />
21  </div>
22 </div>

```

Kemudian buat sebuah button yang berfungsi untuk menambahkan state person ke dalam persons.

```

1  <button onClick={addPerson}>Add Person</button>

```

Selanjutnya buat function addPerson,

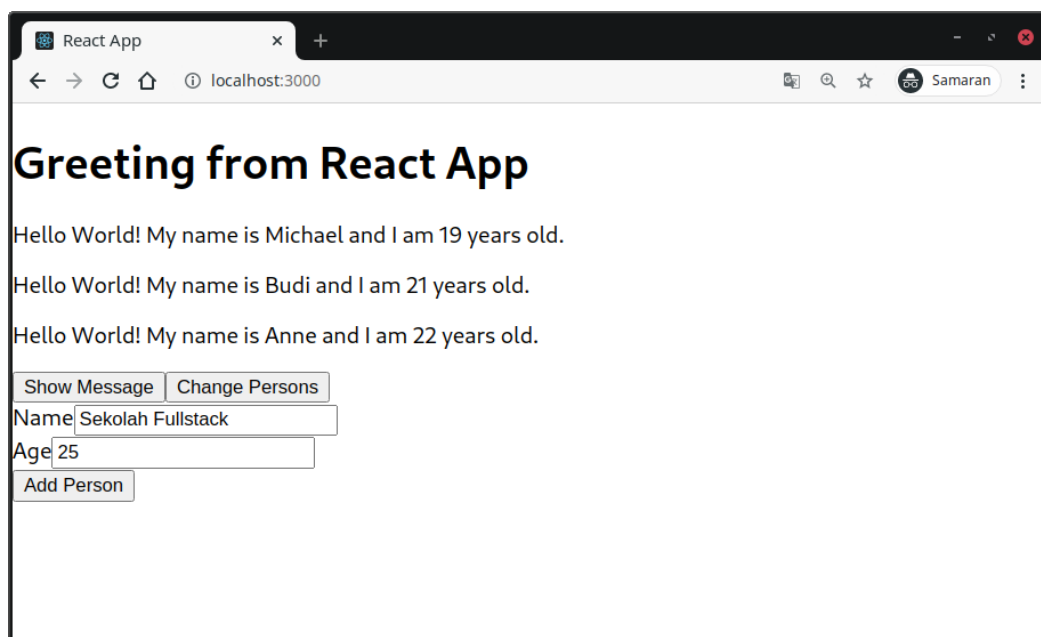
```

1  const addPerson = () => {
2    setPersons([
3      ...persons,
4      person
5    ])
6    setPerson(initialPerson)
7  }

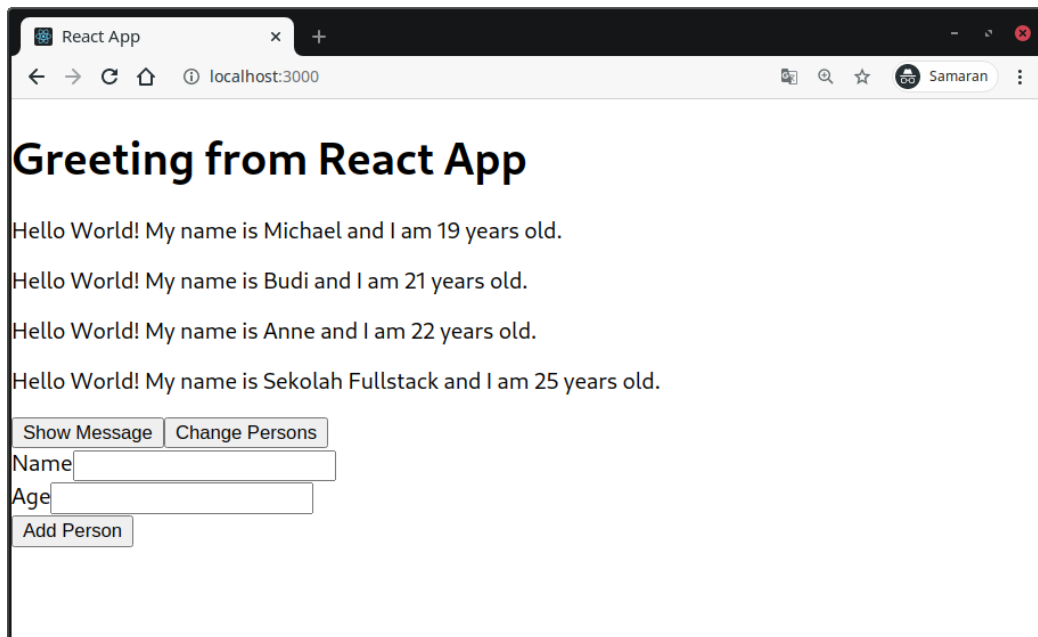
```

Function `addPerson` berfungsi untuk mengubah state `persons` berdasarkan nilai `persons` sebelumnya dan menambahkan state `person` kedalamnya. Setelah itu, mengembalikan state `person` seperti semula.

Sekarang, coba ubah nilai dari `name` dan `age`. Kemudian tekan tombol `Add Person`. Maka *list persons* akan bertambah sesuai data yang kita masukan.



Setelah menekan tombol `Add Person` menjadi:



6.19. Exercise

Buat 2 komponen *User input* (dengan tag `input`), *User output* (dengan tag `span`), dan *User submit* (dengan tag `button`). Jadikan komponen *User input* dan *User output* saling berhubungan secara *real time* dengan menggunakan props (pertukaran data dari komponen *User input* ke *User output*). Sedangkan *User submit* akan menampilkan alert untuk memvalidasi input telah diisi atau belum.

Note: Pelajari kembali state, props, dan passing props ke komponen lain.