
# Full Stack Development
# sesi 17

# Callback & Promise

# Sesi 17 | Callback

JavaScript bersifat sekuensial dalam meng eksekusi baris-baris kode. Berikut salah satu contoh untuk memanggil 2 function yang sekuensial

```
1  function myDisplayer(some) {
2    document.getElementById("demo").innerHTML = some;
3  }
4
5  function myCalculator(num1, num2) {
6    let sum = num1 + num2;
7    return sum;
8  }
9
10 let result = myCalculator(5, 5);
11 myDisplayer(result);
12
```

Dengan teknik *callback* kita bisa membuat pemanggilan 2 function tersebut jadi lebih nice...

```
1  function myDisplayer(some) {
2    document.getElementById("demo").innerHTML = some;
3  }
4
5  function myCalculator(num1, num2, myCallback) {
6    let sum = num1 + num2;
7    myCallback(sum);
8  }
9
10 myCalculator(5, 5, myDisplayer);
11
```

**Dan dikutip dari w3schools :** *Callbacks are working good in asynchronous functions, where one function has to wait for another function (like waiting for a file to load)*.

# **Sesi 17** | Promises

## [ Darkside of Callback ]

Sebelumnya, kita menggunakan callback untuk menghandle proses asynchronous, namun lama-kelamaan kita akan merasa kesulitan ketika callback yang digunakan semakin banyak, bahkan, bakalan ada callback didalam callback dan seterusnya. Masalah ini sering disebut *callback hell*

```
// Callbak HELL
fs.readFile('content1.txt', 'utf-8', (err, content1) => {
  if (err) throw err
  fs.readFile('content2.txt', 'utf-8', (err, content2) => {
    if (err) throw err
    fs.readFile('content3.txt', 'utf-8', (err, content3) => {
      if (err) throw err
      fs.readFile('content4.txt', 'utf-8', (err, content4) => {
        if (err) throw err
        fs.writeFile('result.txt', content1 + content2 + content3 + content4, err => {
          if (err) throw err
          console.log('Writing done!')
        })
      })
    })
  })
})

```

# **Sesi 17** | Promises

## [ Promise ]

**Promise** adalah salah satu konsep yang hadir di ES6 (ES2015). Dikutip dari beberapa sumber, berikut adalah beberapa penjelasan mengain Promise :

- Sebuah object yang “mewakili” hasil akhir dari sebuah proses. Hasil ini bisa jadi sebuah keberhasilan atau kegagalan
- Sebuah object yang “dikembalikan”, dimana kita bisa menyambungkannya dengan 1 atau beberapa callbacks. Teknik menyambungkan callbacks nya berbeda dengan teknik yang sebelumnya, dimana kita melakukan pemanggilan callback sebagai parameter dari sebuah function

Promise memiliki 3 buah state :

- Pending ( sedang dalam proses )
- Fulfilled ( berhasil )
- Rejected ( gagal )

Promise memiliki fasilitas *chaining*, yaitu fasilitas dimana kita bisa melakukan pemanggilan beberapa callbacks secara sekuensial, dengan memanfaatkan return value dari callbacks sebelumnya sebagai parameter input dari callbacks selanjutnya

*Chaining* dapat kita lakukan dengan syntax *then()* yang disediakan oleh object Promise

# **Sesi 17** | Promises

## [ Callback vs Promise ]

```
// Callback style
setTimeout(function() {
  myFunction("I love You !!!");
}, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}


// Promise style
let myPromise = new Promise(function(myResolve, myReject) {
  setTimeout(function() {
    myResolve("I love You !!");
  }, 3000);
});

myPromise
  .then(function(value) {
    document.getElementById("demo").innerHTML = value;
  });

```

```
// Callback style
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);

// Promise style + arrow function
// Note : every function built returning Promise object
doSomething()
  .then(result => doSomethingElse(result))
  .then(newResult => doThirdThing(newResult))
  .then(finalResult => {
    console.log(`Got the final result: ${finalResult}`);
  })
  .catch(failureCallback);

```

# Document Object Model
# - DOM -

# Document Object Model - DOM

Dalam mengembangkan web, kita harus menyadari bahwa kita sekaligus membuat Document Object Model (DOM) yang tersusun dalam dokumen HTML. Dengan pengetahuan DOM, kita bisa secara lebih lengkap mengetahui dan mampu untuk membuat interaksi pada halaman web menggunakan JavaScript.

DOM API di HTML umumnya adalah untuk node ataupun objek element, document, dan window. Setiap hal tersebut memiliki berbagai property (nilai) dan method (aksi), bahkan bisa juga dipasang sebuah penangan kejadian (event handler) sehingga jika ada kejadian tertentu dilakukan suatu statement akan dijalankan.

- window: frame, parent, self, top
- history
- location
- document
- element: body, h1, p, button, dll

# Document Object Model - DOM

Mulai dari bagian ini, snippet atau potongan kode ini akan berlanjut/bersambung. Sebelum mencoba melakukan seleksi dan manipulasi, kita coba asumsikan penggunaan Kita bisa melakukan seleksi terhadap DOM dengan mengunakan beberapa sintaks berikut:

```html
<html>
  <head>
    <title>Contoh Webpage Standard</title>
  </head>
  <body>
    <div id="page-title">Sample Page Title</div>
    <h1>Test Sample Heading</h1>
    <div class="page-box">Page Box 1</div>
    <div class="page-box">Page Box 2</div>
    <div class="page-box">Page Box 3</div>
    <script src="js-simple-dom-script.js"></script>
  </body>
</html>
```

# Document Object Model - DOM

```
1 var pageTitleElement =
document.getElementById("page-title");

2 // Menyeleksi DOM berdasarkan Id element dan
menampungnya ke dalam variabel. Isinya merupakan object
HTML element

3

4 var pageBoxElements =
document.getElementsByClassName("page-box");

5 // Menyeleksi DOM berdasarkan nama class element dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element, walau <h1> hanya ada 1.

6

7 var pageHeadings =
document.getElementsByTagName("h1");

8 // Menyeleksi DOM berdasarkan tag <h1> dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element
```

# Mengakses isi HTML dari DOM

```
1 var pageTitleElement =
document.getElementById("page-title");

2 // Menyeleksi DOM berdasarkan Id element dan
menampungnya ke dalam variabel. Isinya merupakan object
HTML element

3

4 var pageBoxElements =
document.getElementsByClassName("page-box");

5 // Menyeleksi DOM berdasarkan nama class element dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element, walau <h1> hanya ada 1.

6

7 var pageHeadings =
document.getElementsByTagName("h1");

8 // Menyeleksi DOM berdasarkan tag <h1> dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element
```

```
9

10 var pageTitleElementsContent =
pageTitleElement.innerHTML;

11 console.log('isi <div id="page-title"> :' +
pageTitleElementsContent);

12 // isi <div id="page-title"> adalah Sample Page Title

13

14 var pageBoxElementsContent =
pageBoxElements.innerHTML;

15 console.log('isi <div class="page-box"> :' +
pageBoxElementsContent);

16 // isi <div class="page-box"> adalah undefined!
```

# Mengakses isi HTML dari DOM

```
1 var pageTitleElement =
document.getElementById("page-title");

2 // Menyeleksi DOM berdasarkan Id element dan
menampungnya ke dalam variabel. Isinya merupakan object
HTML element

3

4 var pageBoxElements =
document.getElementsByClassName("page-box");

5 // Menyeleksi DOM berdasarkan nama class element dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element, walau <h1> hanya ada 1.

6

7 var pageHeadings =
document.getElementsByTagName("h1");

8 // Menyeleksi DOM berdasarkan tag <h1> dan
menampungnya ke dalam variabel. Isinya merupakan array
dari object HTML element
```

```
9

10 var pageTitleElementContent =
pageTitleElement.innerHTML;

11 console.log('isi <div id="page-title"> :' +
pageTitleElementContent);

12 // isi <div id="page-title"> adalah Sample Page Title

// Meloop array pageBoxElements

13

14 for(var i = 0; i < pageBoxElements.length; i++) {

15 var currentPageBoxElement         =
pageBoxElements[i];

16 var currentPageBoxElementContent  =
currentPageBoxElement.innerHTML;

17
```

# Mengakses isi HTML dari DOM

```
// Mengambil isi elemen pageBoxElements yang kedua, yaitu index ke 1

var secondPageBoxElement          = pageBoxElements[1];

var secondpageBoxElementContent   = secondPageBoxElement.innerHTML;


// Mengambil isi elemen pageBoxElements yang ketiga, yaitu index ke 2

var thirdPageBoxElement           = pageBoxElements[2];

var thirdpageBoxElementContent    = thirdPageBoxElement.innerHTML;
```

```

// Menampilkan isi elemen dengan console.log

console.log('isi <div class="page-box"> yang pertama:' + firstpageBoxElementContent);

console.log('isi <div class="page-box"> yang kedua:' + secondpageBoxElementContent);

console.log('isi <div class="page-box"> yang ketiga:' + thirdpageBoxElementContent);
```

# DOM Transversing

## Apa itu DOM Transversing?

Di layout HTML yang cukup kompleks, kita akan bertemu dengan banyak element HTML yang memiliki hubungan parent-child yang dalam, dan pada saat kita menggunakan JavaScript untuk menseleksi atau memanipulasinya, tidak mungkin kita harus memberikan id atau class ke semua element-nya. Kita bisa menseleksi element HTML dari parent atau dari childnya. Untuk lebih dalam memahami hal ini, kamu harus telah mengerti hierarki parent-child yang terjadi di susunan HTML. Tapi tenang saja, kita akan mengulas ulang hal tersebut.

# DOM Transversing

## Apa itu DOM Transversing?

Di layout HTML yang cukup kompleks, kita akan bertemu dengan banyak element HTML yang memiliki hubungan parent-child yang dalam, dan pada saat kita menggunakan JavaScript untuk menseleksi atau memanipulasinya, tidak mungkin kita harus memberikan id atau class ke semua element-nya. Kita bisa menseleksi element HTML dari parent atau dari childnya. Untuk lebih dalam memahami hal ini, kamu harus telah mengerti hierarki parent-child yang terjadi di susunan HTML. Tapi tenang saja, kita akan mengulas ulang hal tersebut.

```html
<html>
  <head></head>
  <body>
    <h1></h1>
    <div id="contoh-div-1">
      <p id="contoh-p-1">
        <strong></strong>
        <em></em>
      </p>
    </div>
    <div id="contoh-div-2">
      <h2></h2>
      <p id="contoh-p-2"></p>
      <ul>
```

```
      <li></li>
      <li></li>
    </ul>
  </div>
  <script src="dom-transverse-1-intro.js"></script>
  <script
src="dom-transverse-2-siblings.js"></script>
  <script
src="dom-transverse-3-chaining-selectors.js"></script>
</body>
</html>
```

# DOM Transversing

html : merupakan parent paling atas

head : merupakan child dari html body : merupakan child dari html, sibling dari head h1 : merupakan child dari body

div id="contoh-div-1" : merupakan child dari body, sibling dari <h1>

p id="contoh-p-1" : merupakan child dari div id="contoh-div-1"

strong : merupakan child dari p id="contoh-p-1"

em : merupakan child dari p id="contoh-p-1", sibling dari strong

div id="contoh-div-2" : merupakan child dari body, sibling dari h1 dan div id="contoh-div-1"

h2 : merupakan child dari div id="contoh-div-2" p id="contoh-p-2" : merupakan child dari div id="contoh-div-2", sibling dari h2

ul : merupakan child dari div id="contoh-div-2", sibling dari h2 dan p id="contoh-p-2"

li : merupakan child dari ul

# Parent - Child

Untuk mulai mengenai transerving atau penjelajahan di dalam DOM, kita coba mulai dengan menjelajahi hubungan Parent - Child. Contoh pertama kita mulai dengan menseleksi <body> dan mendapatkan element HTML apa saja yan menjadi children dari <body>.

```
/*
==================================================
Menseleksi DOM berdasarkan hubungan Parent - Child
==================================================
*/

```

```
// Menseleksi element <body>
 var body = document.body;

// Mendapatkan element children dari <body>
var bodyChilds = body.children;

// Menampilkan DOM yang menjadi child dari <body>
dalam bentuk array
console.log(bodyChilds); // h1, div
id="contoh-div-1", div id="contoh-div-2", scripts js
```

```
/*
==================================================
Menseleksi DOM berdasarkan hubungan Parent - Child
==================================================
*/

// Menseleksi element <body>
var body = document.body;

// Mendapatkan element children dari <body>
var bodyChilds = body.children;

// Menampilkan DOM yang menjadi child dari <body>
```

```
console.log(bodyChilds);


// Menseleksi element <div id="contoh-div-1">
var contohDiv1 =
document.getElementById('contoh-div-1');


// Mendapatkan element children dari <div
id="contoh-div-1"> dalam bentuk array
var contohDiv1Childs = contohDiv1.children;
```

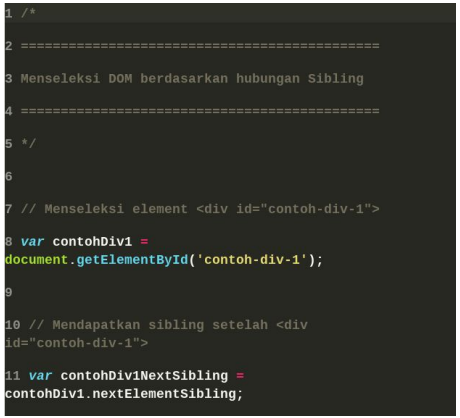
```

// Mendapatkan element children dari <div
id="contoh-div-1"> dalam bentuk array
var contohDiv1Childs = contohDiv1.children;
console.log(contohDiv1FirstChild); // <p
id="contoh-p-1">...</p>

// Note: Walaupun children mungkin hanya 1 element,
tetap tertampung dalam array!
```

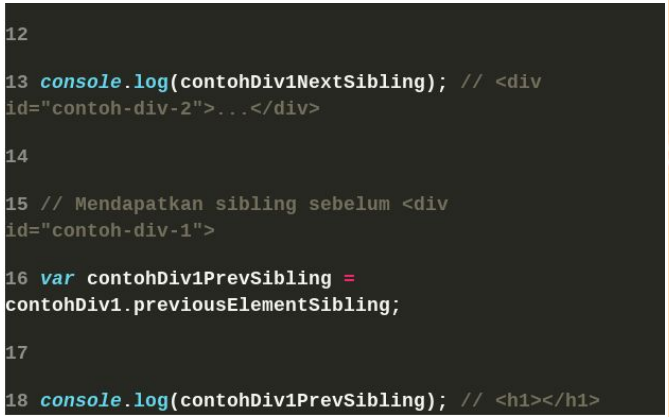
# Siblings

Apabila sebelumnya kita mempelajari hubungan DOM sebagai parent dan child, sekarang kita akan membahas tentang hubungan antar sibling. Sibling, layaknya saudara kandung dalam analogi keluarga, merupakan DOM yang merupakan child dari parent yang sama.

```
/*
==========================================
Menseleksi DOM berdasarkan hubungan Sibling
==========================================
*/


// Menseleksi element <div id="contoh-div-1">
var contohDiv1 =
document.getElementById('contoh-div-1');


// Mendapatkan sibling setelah <div
id="contoh-div-1">
```

```
var contohDiv1NextSibling =
contohDiv1.nextElementSibling;


console.log(contohDiv1NextSibling); // <div
id="contoh-div-2">...</div>
```



# DOM Creation
## Apa itu DOM Creation?

Dalam aplikasi web, kita tidak terbatas memanfaatkan DOM untuk dimanipulasi atau ditambahkan event-event tertentu. Kita juga dapat membuat DOM baru menggunakan JavaScript. Bahkan, kita dapat membuat sebuah halaman web yang full dengan element HTML namun script HTML kita sangat sedikit, dan hampir seluruhnya di render menggunakan JavaScript. Prinsip inilah yang dilakukan oleh Angular, React, dan teknologi front-end lainnya untuk merender DOM.

```html
<html>
  <head>
    <title>Super Simple Web Page</title>
  </head>
  <body>
    <div id="main">
      <div id="inside-main">
        <h1>Heading Sample 1</h1>
        <button>Click Me!</button>
      </div>
    </div>
  </body>
</html>
```

```html
<html>
<head>
  <title>Super Simple Web Page - Fresh From the JS</title>
</head>
<body>
  <!-- it's empty, just contain a js file -->
  <script src="js-dom-creation-script.js"></script>
</body>
</html>
```

```js
// Pertama, kita seleksi terlebih dahulu <body>
var body = document.body;

// Kemudian, kita buat sebuah element HTML <div> menggunakan createElement
var mainDiv = document.createElement('div');

// Untuk membuat <div id="main">, maka kita harus membuat HTML attribute id
var mainDivAttrId = document.createAttribute('id');

// Untuk memberikan nilai kepada id, maka kita gunakan .value
mainDivAttrId.value = "main";

// id="main" kita sudah siap. Sekarang kita harus menambahkan attribute tersebut ke mainDiv
mainDiv.setAttributeNode(mainDivAttrId);

// mainDiv kita sudah menjadi <div id="main">. Saatnya kita tambahkan ke dalam <body>
// Karena Kita akan meletakkan <div id="main"> di dalam <body>, maka kita gunakan appendChi
body.appendChild(mainDiv);
```

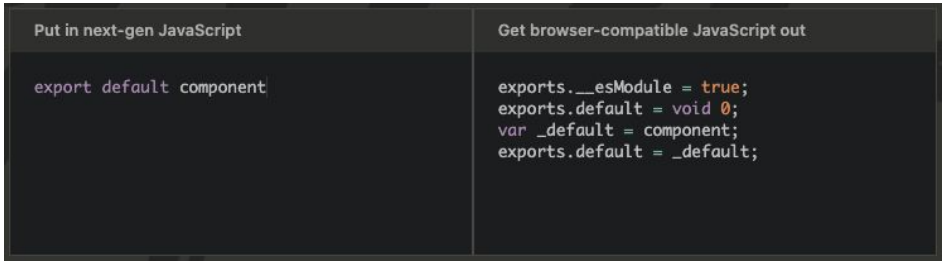
# Pengenalan Babel dan Webpack

# **Sesi 17** | Pengenalan Babel dan Webpack


Javascript adalah salah satu bahasa pemrograman dengan perkembangan yang paling pesat. Berbagai macam fitur dan sintaks diperkenalkan di ES6. Namun perkembangan tadi ternyata membutuhkan waktu yang cukup lama untuk dapat diadaptasi baik oleh browser maupun NodeJS

Nah, Babel adalah penyelamat kita gaes. Hal ini disebabkan karena babel itu sendiri adalah sebuah transpiler, yang tugasnya menterjemahkan sintaks-sintaks yang unsupported ke sintaks yang di-support oleh browser atau NodeJS

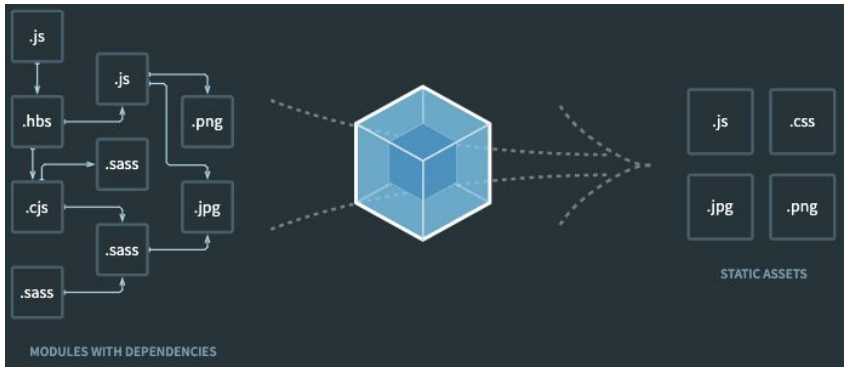
Silakan temen-temen pelajari dan pahami Babel langsung dari [https://babeljs.io/](https://babeljs.io/)



# **Sesi 17** | Pengenalan Babel dan Webpack

Semua berawal dari *module*. Sederhananya, *module* adalah sebuah berkas yang berisi script kode. Module memiliki sifat khusus, yakni dapat memuat atau dimuat oleh *module* lainnya. Berkat sifat inilah antar *module* dapat saling ekspor dan impor untuk bertukar fungsi.

Nah, *module* ini jumlahnya bisa banyak, kompleks, dan bahkan bisa jadi ada beberapa *module* dengan nama yang sama. Untuk itu, kita memerlukan sebuah *tool* yang dapat bertindak sebagai *module management*. Dalam hal ini, istilah yang dipakai oleh *module management* ini adalah *bundler*



# **Sesi 17** | Pengenalan Babel dan Webpack


*Core concept* dari **Webpack** adalah :

- Entry
- Output
- Loaders
- Plugins
- Mode
- Browser Compatibility

Untuk memahami lebih dalam soal **Webpack** langsung dari websitenya, silakan kunjungi : https://webpack.js.org/

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 1 ]

Nah terus, apa sih gunanya kita mengenal Babel dan Webpack ini ? Jadi, di materi selanjutnya, kita akan berkenalan dengan React JS. Namun, sebelum kita masuk ke materi tersebut, kita akan melakukan instalasi React JS secara manual. Nah, langkah-langkah manual ini, mengikutsertakan Babel dan Webpack dalam proses instalasinya. Langkah-langkah ini dirangkum dari https://dev.to/deadwing7x/setup-a-react-app-with-webpack-and-babel-4o3k

**CATATAN :** Semua perintah-perintah yang dicontohkan adalah untuk dilakukan pada OS Linux. Untuk WINDOWS, beberapa perintah harus dicari ekivalensinya, ATAU dilakukan dengan UI pada Windows Explorer

Berikut adalah langkah-langkahnya :

**1. Buat folder aplikasi dan masuk ke folder tersebut**
> mkdir reactApp
> cd reactApp

**2. Generate file “package.json”**
> npm init -y

**3. Install React dan React Dom**
> npm install react --save
> npm install react-dom --save

... lanjut ke slide selanjutnya ->

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 2 ]

... lanjutan :

**4. Install webpack dan semua pendukungnya**

```
> npm install webpack --save-dev
> npm install webpack-dev-server --save-dev
> npm install webpack-cli --save-dev
```

**5. Install babel dan semua pendukungnya**

```
> npm install @babel/core --save-dev
> npm install @babel/node --save-dev
> npm install @babel/preset-env --save-dev
> npm install @babel/preset-react --save-dev
> npm install babel-loader --save-dev
```

**6. Install path dan webpack plugin**

```
> npm install path --save-dev
> npm install html-webpack-plugin --save-dev
```

**7. Buat folder src dan isi dengan file-file berikut**

```
> mkdir src
> touch src/index.html
> touch src/index.js
```

Untuk WINDOWS, touch bisa digantikan dengan :

```
> type nul > index.html
```

... lanjut lagi ya gaes ->

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 3 ]

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  entry: path.join(__dirname, "src", "index.js"),
  output: { path: path.join(__dirname, "build"), filename: "index.bundle.js" },
  mode: process.env.NODE_ENV || "development",
  resolve: { modules: [path.resolve(__dirname, "src"), "node_modules"] },
  devServer: { static: path.join(__dirname, "src") },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ["babel-loader"]
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: path.join(__dirname, "src", "index.html"),
    }),
  ],
};
```

... lanjutan :

**8. Buat file webpack.config.js dan isikan dengan kode di samping**

```
> touch webpack.config.js
```

... ayo guys, lanjut lagi ->

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 4 ]

... lanjutan :

**9. Buka file package.json dan ubah bagian “scripts” menjadi seperti di samping**

```
  "scripts": {
    "webpack": "webpack",
    "start": "webpack serve",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

**10. Buka file index.html dan isikan dengan kode seperti di samping**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React with Webpack and Babel</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root">
      <!-- This div is where our app will run -->
    </div>
  </body>
</html>
```

... ayo guys, lanjut lagi ->

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 5 ]

... lanjutan :

**11. Buka file src/index.js dan isikan dengan kode seperti di bawah**

```
import React from 'react';
import ReactDOM from 'react-dom';

const HelloWorld = () => {
  return (
    <h1>Hello World</h1>
  );
}

ReactDOM.render(<HelloWorld />,
  document.getElementById("root"));
```

**12. Buat file .babelrc dan isikan dengan kode seperti di bawah**

```
> touch .babelrc
```

... ayo guys, lanjut lagi ->

```
{
  "presets": [
    "@babel/env",
    "@babel/react"
  ]
}
```

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 6 ]

... lanjutan :

**13. Lakukan kompilasi dengan webpack**

```
> npm run webpack
```

```
└─▪ npm run webpack

> react-app@1.0.0 webpack
> webpack

asset index.bundle.js 1010 KiB [emitted] (name: main)
asset index.html 496 bytes [compared for emit]
runtime modules 274 bytes 1 module
modules by path ./node_modules/ 974 KiB
  modules by path ./node_modules/scheduler/ 26.3 KiB
    modules by path ./node_modules/scheduler/*.js 412 bytes 2 modules
    modules by path ./node_modules/scheduler/cjs/*.js 25.9 KiB
      ./node_modules/scheduler/cjs/scheduler.development.js 17.2 KiB [built] [code generated]
      ./node_modules/scheduler/cjs/scheduler-tracing.development.js 8.79 KiB [built] [code generated]
  modules by path ./node_modules/react/ 70.6 KiB
    ./node_modules/react/index.js 190 bytes [built] [code generated]
    ./node_modules/react/cjs/react.development.js 70.5 KiB [built] [code generated]
  modules by path ./node_modules/react-dom/ 875 KiB
    ./node_modules/react-dom/index.js 1.33 KiB [built] [code generated]
    ./node_modules/react-dom/cjs/react-dom.development.js 874 KiB [built] [code generated]
  ./node_modules/object-assign/index.js 2.06 KiB [built] [code generated]
./src/index.js 518 bytes [built] [code generated]
webpack 5.51.1 compiled successfully in 2188 ms
```

**14. Jalankan server dan buka aplikasi di browser**

```
> npm start
```

```
└─▪ npm start

> react-app@1.0.0 start
> webpack serve

<i> [webpack-dev-server] Project is running at:
<i> [webpack-dev-server] Loopback: http://localhost:8080/
<i> [webpack-dev-server] On Your Network (IPv4): http://192.168.1.107:8080/
<i> [webpack-dev-server] On Your Network (IPv6): http://[fe80::1]:8080/
```

# **Sesi 17** | Pengenalan Babel dan Webpack

## [ Instalasi - part 7 ]

**SELAMAT !!! HELLO WORLD VERSI REACT JS DENGAN INSTALASI MANUAL TELAH BERHASIL KAMU BUAT**