

Andre Pratama

React Uncover

Panduan Belajar Library React JS 18

DuniaIlkom

React Uncover

Panduan Belajar Library React JS 18

Andre Pratama

Buku ini ditulis dan diterbitkan secara mandiri oleh DuniaIlkom (www.duniailkom.com)

06 April 2022

~ Update Log ~

- ✓ First Release - Desember 2021
- ✓ Update ke React 18 - April 2022

Cover Photo by [Maximalfocus](#) on [Unsplash](#)

© 2022 DuniaIlkom

Daftar Isi

Ucapan Terima kasih.....	10
Tentang Penulis.....	11
Lisensi.....	12
Kata Pengantar.....	14
Asumsi / Pengetahuan Dasar.....	15
Contoh Kode Program.....	16
1. Berkenalan Dengan React.....	17
1.1. Pengertian React.....	17
1.2. Sejarah React.....	18
1.3. Kenapa Harus Menggunakan React?.....	19
SPA (Single Page Application) vs MPA (Multi Page Application).....	21
1.4. React vs Vue vs Angular.....	21
Belajar React untuk ke React Native.....	24
Facebook Yang Ada di Belakang React.....	25
2. JavaScript (ES6) untuk React.....	27
2.1. Membuat Variabel Dengan let.....	27
2.2. Membuat Konstanta Dengan const.....	30
2.3. Template String.....	30
2.4. Conditional Operator.....	32
2.5. Short-Circuit Conditionals.....	34
2.6. Arrow Notation.....	35
2.7. Function Sebagai First-Class Citizens.....	38
2.8. Class dan Object.....	39
2.9. Memproses Object.....	42
2.10. Array dan Object Destructuring.....	44
2.11. Spread Operator.....	45
2.12. Rest Parameter.....	48
2.13. Array Method: forEach() dan map().....	49
2.14. Asynchronous JavaScript (Promise / Async - Await).....	54

JavaScript Promise.....	56
Async - Await.....	59
2.15. JavaScript Module.....	60
Multiple Import / Export.....	63
Export Default.....	64
Import Alias.....	65
Import All.....	65
3. React dan React DOM.....	67
3.1. Mengenal File react.js dan react-dom.js.....	67
3.2. Membuat Template HTML.....	68
File React di CDN.....	68
File React di Local.....	69
3.3. Method React.createElement().....	71
3.4. Method ReactDOM.createRoot().render().....	72
Pesan Warning di Tab Console.....	75
3.5. Method ReactDOM.render().....	76
3.6. Multiple ReactDOM.createRoot().render().....	78
3.7. Nested React Element.....	79
4. JSX (JavaScript XML).....	85
4.1. Pengertian JSX.....	85
4.2. Mengakses File Babel.....	86
4.3. Aturan Dasar Penulisan JSX.....	89
4.4. Statement di Dalam JSX.....	93
4.5. Penulisan Atribut style.....	94
4.6. Percabangan Kondisi.....	95
4.7. Array Processing.....	97
4.8. JavaScript Event.....	103
5. React Component.....	105
5.1. Pengertian React Component.....	105
5.2. Class Component vs Functional Components.....	106
5.3. Membuat Class Component.....	107
Multiple Component.....	109
Nested Component.....	110
5.4. Membuat Functional Component.....	111

5.5. Gabungan Class dan Functional Component.....	112
5.6. Menambah Property dan Method di Class Component.....	113
5.7. Menampilkan Data dengan Class Component.....	116
5.8. Memecah Class Component.....	120
5.9. Menampilkan Data dengan Functional Component.....	123
6. React Props.....	125
6.1. Pengertian Props.....	125
6.2. Props di Class Component.....	125
Children Props.....	133
Mengakses Props di Constructor.....	133
6.3. Props di Functional Component.....	135
7. React Event.....	138
7.1. Pengertian Event dan Event Handler.....	138
7.2. Event Handler di Class Component.....	138
7.3. Event Binding.....	141
7.4. Event Object.....	145
7.5. Menurunkan (Passing Down) Event Handler.....	148
7.6. Event Handler di Functional Component.....	150
8. React State.....	154
8.1. Pengertian State.....	154
8.2. Prinsip Kerja State.....	160
8.3. Virtual DOM React.....	162
8.4. State di Class Component.....	165
Mengakses State Setelah di Update.....	167
Membuat Beberapa State.....	171
Mengisi Object Sebagai Nilai State.....	173
8.5. Mini Project: Slide Data Mahasiswa.....	176
8.6. Mini Project: Tukar Warna Background.....	183
8.7. Mini Project: Counter Click.....	186
8.8. Function Sebagai Nilai State.....	188
8.9. Mengirim Nilai State ke Child Component.....	191
8.10. Mengubah Nilai State dari dalam Child Component.....	194
8.11. Alur Pengiriman Data di React.....	200
9. React useState Hook.....	202

9.1. Pengertian React Hook.....	202
9.2. Cara Penggunaan useState Hook.....	202
Mengisi Object Sebagai Nilai State (Nested Object).....	206
9.3. Function Sebagai Nilai State.....	213
9.4. Mengirim / Mengupdate State dari Child Component.....	215
9.5. Mencegah State Dibuat Ulang.....	222
10. React ref dan useRef Hook.....	225
10.1. Pengertian ref dan useRef Hook.....	225
10.2. Ref di Class Component.....	230
11. React Form Processing.....	232
11.1. Mengakses Nilai Form (Value Property).....	232
11.2. Menggunakan onChange Event.....	234
11.3. Menampilkan Nilai Form Secara Realtime.....	236
11.4. Mengubah Nilai Form dari State.....	237
11.5. Mengakses Nilai Textarea, Select, Checkbox dan Radio.....	238
Mengakses Nilai Textarea.....	239
Mengakses Nilai Select.....	240
Mengakses Nilai Checkbox.....	243
Mengakses Nilai Radio Button.....	244
11.6. Mengakses Nilai Form Utuh.....	246
11.7. Validasi Form.....	251
11.8. Form Processing di Functional Component.....	258
11.9. Validasi Secara Realtime.....	262
12. Ref DOM Manipulation.....	266
12.1. Mengakses DOM dengan Ref.....	266
12.2. Mengakses Nilai Form dengan ref.....	268
13. React Component Lifecycle.....	274
13.1. Memahami React Component Lifecycle.....	274
13.2. Praktek Mounting, Updating dan Unmounting.....	276
13.3. Mini Project: Countdown.....	283
Mengatasi Warning "memory leak".....	285
Menambah Tombol "Stop".....	286
Auto start Countdown.....	289
Mengatasi Masalah Countdown Negatif.....	290

Multiple Countdown.....	291
13.4. useEffect Hook.....	295
useEffect Setelah Proses Render (Updating).....	296
useEffect untuk Tahap Mounting.....	298
useEffect untuk Tahap Unmounting.....	298
useEffect untuk Tahap Mouting dan Unmouting.....	299
Menjalankan Beberapa useEffect.....	299
Menjalankan useEffect untuk State Tertentu.....	300
13.5. Mini Project: Countdown (Functional Component).....	302
14. Mini Project: Todo.....	307
14.1. Todo App: Tampilan dan State Awal.....	307
14.2. Todo App: Validasi Inputan Todo.....	311
14.3. Todo App: List Awal Todo.....	313
14.4. Todo App: Menambah Todo.....	314
14.5. Todo App: Menghapus Todo.....	316
14.6. Todo App: Membuat Todo Component.....	319
14.7. Todo App: Membuat TodoForm Component.....	321
14.8. Todo App: Versi Functional Component.....	325
15. Create React App.....	328
15.1. Menginstall Create React App.....	328
15.2. File Bawaan Create React App.....	332
15.3. Menjalankan Live Server.....	334
15.4. Modifikasi File Create React App.....	336
15.5. Mengganti Perintah Render ke React 18.....	340
15.6. Membuat React Component.....	342
15.7. Membuat State.....	346
15.8. Pemisahan Komponen.....	348
15.9. Import File CSS.....	350
15.10. CSS Modules.....	352
15.11. useEffect hook dan Fragment.....	357
15.12. Class Component.....	359
15.13. Todo Project.....	360
15.14. Publish File React.....	364
Mengatur Path Hasil Build.....	367
Menjalankan Hasil Build Dari Static Server.....	370

Menjalankan Static Server dari Terminal VS Code.....	372
16. Mini Project: CRUD Mahasiswa.....	376
16.1. Persiapan Awal.....	377
16.2. Menampilkan Data Mahasiswa.....	381
Membuat Komponen RowMahasiswa.....	384
Membuat Komponen RowTambahMahasiswa.....	386
16.3. Menambah Data Mahasiswa.....	389
16.4. Edit Data Mahasiswa.....	401
Membuat "Edit Mode".....	402
Membuat Validasi Form Edit.....	406
Simpan Data Edit.....	409
Membuat Fitur Reset Data.....	412
16.5. Hapus Data Mahasiswa.....	414
17. Mini Project: Ilkoom Expense Tracker.....	423
17.1. Persiapan Awal.....	424
17.2. Komponen Header dan Footer.....	430
17.3. Komponen Transaction dan ListTransaction.....	432
17.4. Komponen SaldoBox.....	441
17.5. Komponen AddTransaction.....	444
17.6. Menambah Transaction.....	453
17.7. Menghapus Transaction.....	456
18. JavaScript Fetch API.....	461
18.1. Pengertian API.....	461
18.2. Pengertian REST API.....	461
18.3. Pengertian JSON.....	462
18.4. Pengertian Fetch API.....	464
18.5. Cara Penggunaan Fetch API.....	465
Mengakses API reqres.in.....	465
Fungsi fetch() JavaScript.....	468
Error Handling Fetch API.....	472
Mengakses Fetch API dengan Async Await.....	474
Error Handling Fetch API dengan Async Await.....	475
18.6. Memproses Hasil API di React.....	476
Error Handling Fetch API di React.....	481

Membuat Pesan Loading.....	482
18.7. Mini Project: Fetch User.....	485
19. Mini Project: Ilkoom Movie Database.....	491
19.1. Persiapan Awal.....	494
19.2. Komponen Header dan Footer.....	498
19.3. Mendapatkan Hak Akses API.....	500
19.4. Mengakses Data API.....	506
19.5. Mengakses Data API dari React.....	517
19.6. Komponen Movie.....	523
19.7. Membuat Pilihan Tahun dan Genre.....	534
19.8. Membuat Load More.....	543
20. Mengonlinekan React App (Firebase Hosting).....	549
20.1. Membuat Akun Firebase.....	549
20.2. Membuat Firebase Project.....	551
20.3. Instalasi Firebase CLI.....	553
20.4. Upload File Ilkoom Movie Database.....	558
20.5. Upload File Ilkoom Expense Tracker.....	567
20.6. Mencoba Firebase Realtime Database.....	570
REST API Realtime Database.....	574
20.7. REST API Ilkoom Expense Tracker.....	587
20.8. Upload File CRUD Mahasiswa.....	598
Penutup React Uncover.....	607
Daftar Pustaka.....	610

Ucapan Terima kasih

Dalam kesempatan ini saya ingin mengucapkan terima kasih kepada Allah SWT karena dengan karuniaNya saya masih diberi kesempatan dan kesehatan untuk bisa menulis buku kesebelas DuniaIlkom: **React Uncover**.

Selanjutnya kepada keluarga yang terus memberi motivasi dan dukungan tiada henti untuk terus mengembangkan DuniaIlkom.

Terakhir kepada rekan-rekan pembaca dan pengunjung setia DuniaIlkom. Terutama bagi yang telah memberikan donasi untuk membeli buku saya sebelumnya. Karena *feedback* dan dukungan rekan-rekan lah saya bisa lanjut menulis buku ini. Terima kasih :)

Padang Panjang, 2022

Penulis

Andre Pratama

www.duniaIlkom.com

Tentang Penulis



Andre Pratama

Andre memiliki background S1 Ilmu Komputer dari Universitas Sumatera Utara. Karena kecintaan akan dunia programming, mulai merintis web DuniaIlkom sejak tahun 2012.

Harapannya, tutorial di web serta buku terbitan DuniaIlkom bisa menjadi salah satu media belajar programming terbaik di Indonesia.

Andre berdomisili di kota Padang Panjang, Sumatera Barat. Jika ada pertanyaan, saran, kritik yang membangun bisa menghubungi duniailkom@gmail.com atau WA ke 083180285808.

Lisensi

Terima kasih untuk tidak memperbanyak / mengedarkan / mencopy eBook ini

Menulis sebuah buku hingga ratusan halaman butuh waktu yang tidak sebentar. Belum lagi saya harus berjuang mempelajari referensi yang kebanyakan dalam bahasa inggris. Ini saya lakukan agar pembaca bisa mendapatkan materi yang detail, update, dan berkualitas.

Saya menyadari kekurangan sebuah ebook adalah mudah dicopy-paste dan disebarluaskan. Tapi dengan eBook, harga buku bisa ditekan. Selain tidak perlu mencetak, eBook DuniaIlkom ini bisa di dapat dengan mudah dan murah, termasuk bagi teman-teman di daerah yang ongkos kirimnya lumayan mahal (jika berbentuk buku fisik).

Atas dasar itulah saya mohon kerjasamanya dari rekan-rekan semua untuk **tidak memperbanyak, menggandakan, atau mengupload ulang buku ini di forum, situs maupun media lain dalam bentuk apapun (termasuk tidak membuat video youtube dari materi buku).**

Saya juga berharap rekan-rekan tidak memposting materi apapun yang ada di dalam buku ini. Jika ingin sebagai bahan artikel untuk postingan blog/situs, silahkan ambil materi yang ada di website duniaIlkom (jangan yang dari buku).

Apabila rekan-rekan memperoleh buku ini **bukan** dari DuniaIlkom, saya mohon bantuan donasinya untuk membeli versi asli. Donasi pembelian buku ini adalah sumber mata pencarian saya untuk menafkahi keluarga. Lisensi atau hak guna buku ini hanya untuk 1 orang, yakni yang telah membeli langsung ke duniaIlkom@gmail.com.

Dengan kualitas yang ditawarkan, harga buku ini cukup terjangkau. Buku ini saya buat dengan waktu yang tidak sebentar, hingga berbulan-bulan, kadang sampai tengah malam. Bantuan donasi dari rekan-rekan yang membeli buku secara resmi sangat saya hargai, selain mendapat ilmu yang berkah, ini juga bisa menjadi penyemangat saya untuk terus berkarya dan menghadirkan ebook-ebook programming berkualitas lainnya.

Untuk yang membeli dari DuniaIlkom, saya ucapan banyak terimakasih :)

Anda diperbolehkan untuk:

- ✓ Mencetak eBook ini untuk keperluan pribadi dan dibaca sendiri.
- ✓ Mencopy eBook ini ke laptop/smartphone/tablet milik sendiri.
- ✓ Membuat ringkasan buku untuk digunakan sebagai bahan ajar (bukan keseluruhan isi buku).

Anda tidak dibolehkan untuk:

- ✗ Mencetak eBook ini untuk dibaca oleh orang lain, walaupun gratis.
- ✗ Mencopy eBook ini untuk dijual ulang, maupun dibagikan kepada orang lain dengan gratis.
- ✗ Membeli buku ini untuk dibaca bersama-sama (lisensi buku ini hanya untuk 1 orang).
- ✗ Mengambil sebagian atau seluruh isi buku untuk di publish ke blog, situs, artikel, dan media lain dalam bentuk apapun.
- ✗ Menjadikan materi buku sebagai bahan video YouTube / media public lain.
- ✗ Membagikan eBook ini kepada murid/siswa/mahasiswa (jika digunakan untuk bahan pengajaran).

Setiap pelanggaran dari lisensi ini akan dituntut sesuai undang-undang yang berlaku di Republik Indonesia, terutama **Pasal 12 UU No. 19 Tahun 2002** tentang **Hak Cipta**.

Penjelasan lebih lanjut bisa ke: [Apakah Mengunduh E-book Termasuk Perbuatan Illegal?](#)

Khusus untuk pembaca muslim bisa ke: [Hukum Memakai Barang Bajakan](#). Mari kita jaga agar ilmu yang di dapat berkah dan bermanfaat, bukan dari sumber yang haram.

Kata Pengantar

Buku React Uncover saya tujuhan bagi rekan-rekan yang ingin mempelajari library React mulai dari awal.

React sendiri merupakan salah satu library/framework JavaScript front-end paling populer saat ini. Dengan React, kita bisa membuat halaman web yang dinamis, dan *react-ive*.

Web yang dibuat dengan React umumnya berbentuk **SPA** (Single Page Application) yang terdiri dari 1 halaman saja. Sepanjang penggunaan web, yang berubah hanya bagian tertentu, tidak perlu me-load ulang semua halaman. Ini membuat web SPA terasa lebih cepat dan responsive.

Karena React adalah sebuah library JavaScript, maka dasar JavaScript wajib dipahami. Tidak hanya itu, React juga banyak menerapkan konsep JavaScript modern yang sering dikenal sebagai **EcmaScript 6** (ES6). Agar lebih mudah, saya juga menyertakan 1 bab khusus di awal buku tentang "JavaScript (ES6) untuk React". Bab ini berisi materi kilat untuk memahami fitur dan cara penulisan terbaru dari ES6.

Setelah itu, kita akan masuk ke "dunia React". Mulai dari cara instalasi, struktur penulisan JSX, *props*, *event*, *state*, *ref*, *hook*, *react lifecycle*, hingga *form processing*.

Di akhir buku juga terdapat berbagai mini project sebagai implementasi dari materi yang sudah dipelajari, diantaranya membuat aplikasi CRUD Mahasiswa, Ilkoom Expense Tracker, dan Ilkoom Movie Database. Tidak ketinggalan, semua mini project akan kita upload ke **firebase hosting** supaya langsung online dan bisa diakses dari internet.

Itulah sekilas materi yang akan dibahas sepanjang buku ini. Mempelajari React memang tidak gampang, namun saya berusaha menyajikan materi yang mudah dipahami disertai banyak contoh kode program.

Besar harapan semoga buku **React Uncover** bisa memandu rekan-rekan untuk menguasai React dengan praktis. Sampai jumpa di bab terakhir :)

Asumsi / Pengetahuan Dasar

React adalah library JavaScript, sehingga saya berasumsi rekan-rekan sudah paham tentang **JavaScript** sebelum membaca buku ini. Selain itu, tentu saja juga harus punya dasar **HTML** dan **CSS**.

Untuk mempercantik tampilan design, saya memakai CSS framework **Bootstrap** di beberapa mini project. Ini tidak wajib, namun menjadi nilai plus jika sudah pernah belajar Bootstrap.

Materi-materi di atas tidak harus dipelajari dari buku DuniaIlkom, tapi bisa juga dari tutorial/buku lain. Jika diperlukan, duniaIlkom juga tersedia buku-buku yang dimaksud:

- ◆ HTML Uncover
- ◆ CSS Uncover
- ◆ JavaScript Uncover
- ◆ Bootstrap Uncover

Contoh Kode Program

Seluruh kode program yang ada dalam buku **React Uncover** ini bisa di download dari folder sharing Google Drive yang di kirim pada saat pembelian: **belajar_react.zip**. Isinya berupa file HTML, CSS dan JS yang disusun sesuai dengan bab tempat kode tersebut dibahas.

File ini bisa dipakai sebagai rujukan jika ada kode program yang tidak jalan, karena biasanya itu disebabkan salah ketik atau ada perintah yang lupa ditulis.

Sepanjang pembahasan nanti kadang saya juga tidak menampilkan kode program utuh (untuk menghemat tempat). Kode program lengkap tersedia di file **belajar_react.zip**.

Khusus untuk materi yang sudah menggunakan *create react app*, file yang tersedia bukan sebuah aplikasi react lengkap, tapi hanya beberapa file saja. Agar bisa berjalan, copy folder **src** dan **public** ke folder aplikasi yang sudah dibuat dengan *create react app*.

Penomoran baris (*line numbering*) pada contoh kode program berguna untuk memudahkan pembahasan. Jika ingin men-copy kode ini langsung dari eBook **pdf**, gunakan kombinasi tombol **ALT + tahan tombol mouse selama proses seleksi** agar *line numbering* tidak ikut di-copy. Namun ini hanya bisa dilakukan dari aplikasi pdf reader tertentu seperti **Adobe Acrobat Reader**.

Cara yang lebih disarankan adalah dengan mengetik ulang seluruh kode program yang ada supaya lebih cepat paham sekaligus bisa menghafal fungsi dari setiap kode. Jika setelah diketik ternyata tidak jalan, besar kemungkinan ada penulisan yang salah.

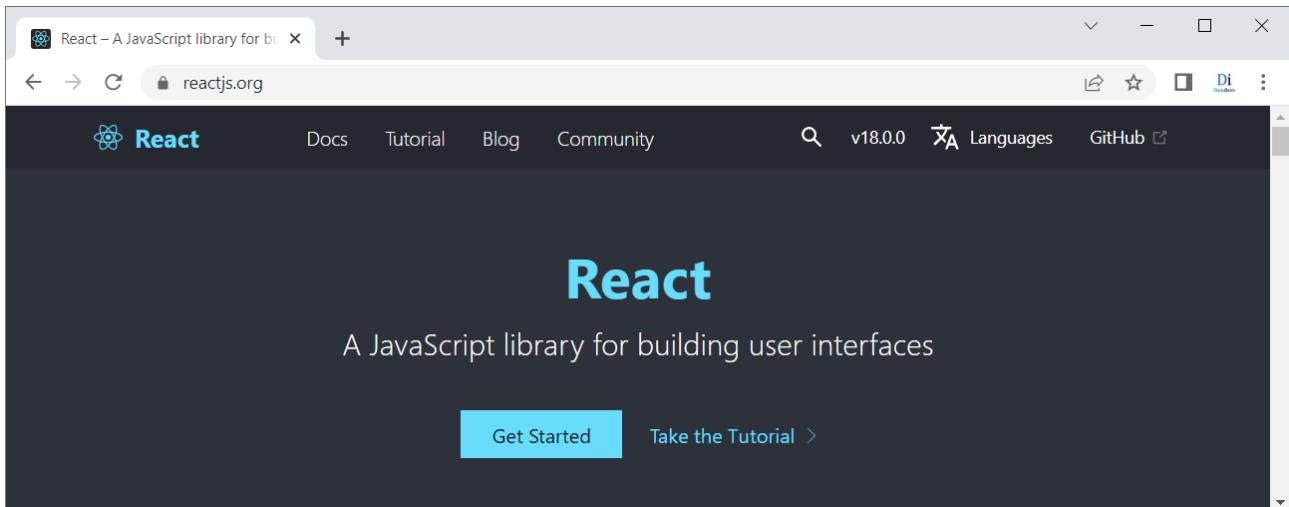
Di dalam programming, 1 saja karakter yang kurang apakah itu berupa titik, tanda koma, atau tanda " > ", kode program tidak akan jalan sempurna. Jika ini yang terjadi, coba samakan dengan file yang ada di dalam folder **belajar_react.zip**.

1. Berkenalan Dengan React

Dalam bab pertama buku React Uncover ini kita akan membahas pengertian React, sejarah singkat, membahas React sebagai library JavaScript, serta perbandingannya dengan library JavaScript lain seperti Vue dan Angular.

1.1. Pengertian React

Mengutip web resmi React di reactjs.org¹, dinyatakan bahwa *React is a JavaScript library for building user interfaces*. Yang artinya React adalah **sebuah library JavaScript untuk membuat antarmuka aplikasi**.



Gambar: Tampilan halaman home React di https://reactjs.org

Dalam programming, *library* merujuk ke kumpulan kode program untuk mempercepat penyelesaian suatu masalah. Sedangkan *user interface* adalah tampilan atau media yang menghubungkan antara user dengan aplikasi.

Maka React juga bisa disebut sebagai "kumpulan kode bantu yang ditulis dalam bahasa JavaScript untuk memudahkan pembuatan media interaksi antara user dengan aplikasi".

Selain library, terdapat juga istilah *framework* yang sama-sama berisi kumpulan kode program. Namun framework lebih kompleks dan punya aturan yang lebih baku. Sebuah framework biasanya terdiri dari kumpulan library saling bekerja sama satu sama lain.

Ketika pertama kali mengenal React, saya sempat bingung apakah React ini sebuah library atau

1. <https://reactjs.org/>

lebih pas disebut framework. Terlebih pesaing React seperti **Vue** dan **Angular** menyatakan diri sebagai framework.

Di internet cukup banyak artikel yang membahas masalah ini. Yang sepakat menyebut React sebagai library adalah karena React hanya fokus ke satu hal spesifik, yakni membuat user interface saja. File yang diperlukan juga cuma satu, yakni file `react.js`. Fitur lain diserahkan ke library terpisah seperti `react-dom`, `react-router`, `redux`, dll.

Ada pula yang menyatakan kalau React lebih pas disebut suatu framework karena punya alur proses tersendiri (*React lifecycle*). Kita harus menulis kode program berdasarkan alur ini, tidak bisa bebas seperti **jQuery** yang murni library. Selain itu mayoritas file library React hanya bisa dipakai dengan React saja, tidak bisa berdiri sendiri.

Terlepas dari perdebatan ini, file utama React memang sederhana (hanya perlu 1 file). Dalam prakteknya nanti, kita harus mengakses file lain untuk membuat berbagai fitur. Pemisahan ini terasa seperti kelemahan karena file asli React bukan "paket lengkap". Namun konsep modular membuat React sangat fleksibel dan bisa disesuaikan tergantung kompleksitas aplikasi.

1.2. Sejarah React

Cikal bakal React berasal dari sebuah prototype library bernama **FaxJS**². Library ini dikembangkan pada tahun 2011 oleh **Jordan Walke**, seorang programmer di Facebook. Saat itu FaxJS coba diterapkan Jordan ke fitur pencarian milik Facebook.

Satu tahun setelahnya, FaxJS disempurnakan menjadi library baru yang diberi nama **React**. Pemilihan nama react kemungkinan besar terinspirasi dari kata "*reactive*".

Saat itu React masih tertutup dan dipakai secara internal oleh tim Facebook saja. React di implementasikan pada fitur periklanan facebook (*facebook ads*) yang sedang ditangani Jordan serta perbaikan fitur Instagram yang baru saja di akuisisi facebook di tahun 2012.

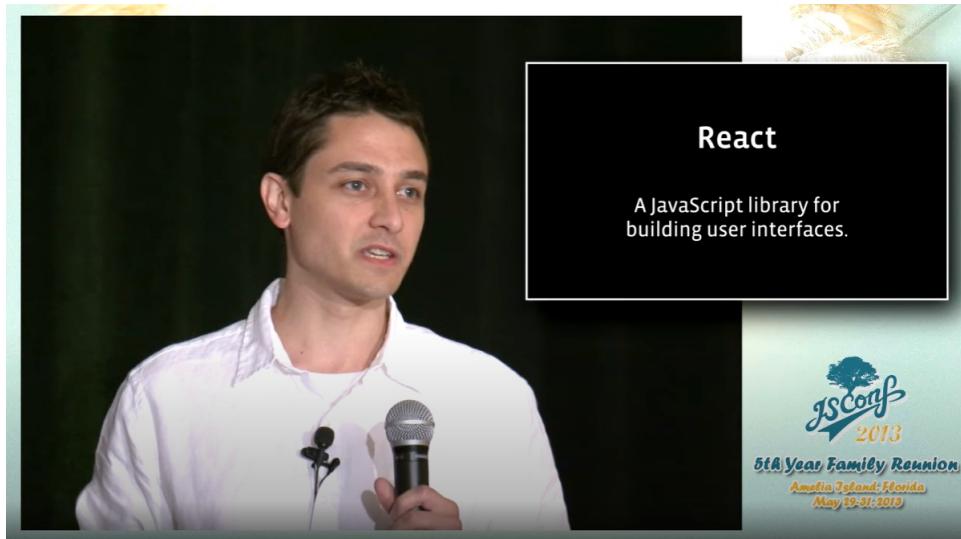
Barulah pada tahun 2013 Jordan Walke secara resmi memperkenalkan React³ saat konferensi JavaScript (**JSConf**) di Florida, Amerika Serikat dan menjadikannya sebagai proyek open source. Mulai dari saat itu React terus dikembangkan dan makin populer sebagai library/framework JavaScript front-end.

Di tahun 2015, tim React memperkenalkan **React Native**. Ini merupakan library tambahan yang memungkinkan React dipakai untuk membuat aplikasi mobile (Android dan iOS). Berbagai perusahaan besar juga mulai menggunakan React, terutama Netflix dan Airbnb.

Pada tahun yang sama, React mengubah penomoran versi. Sejak awal dirilis hingga tahun 2015, penomoran versi selalu diawali angka 0. Mulai dari 0.3 pada tahun 2013 (saat rilis pertama kali), hingga 0.14 di akhir 2015. Setelah itu di April 2016 penomoran React langsung lompat ke 15.0.0.

2. <https://github.com/jordwalke/FaxJs>

3. <https://www.youtube.com/watch?v=GW0rj4sNH2w>



Jordan Walke pada tahun 2013 saat memperkenalkan React (JSConf, Florida, US)

Di awal 2019, React merilis versi 16.8 yang salah satunya menghadirkan fitur *hook*. **Hook** ini cukup revolusioner karena membawa perubahan signifikan bagi mayoritas programmer React. Dengan adanya hook, kode React bisa sepenuhnya ditulis dalam bentuk *functional component*, dari sebelumnya *class component*. Penjelasan lebih lengkap topik ini akan kita bahas pada bab tersendiri.

Pada saat buku ini saya update, rilis terakhir React ada di versi 18. Melihat perkembangan-nya, React akan terus disempurnakan dan membawa fitur-fitur baru untuk menjawab kebutuhan programmer JavaScript.

1.3. Kenapa Harus Menggunakan React?

Berangkat dari pengertian, React berfungsi untuk merancang user interface. Namun kenapa harus menggunakan React? Karena toh kita tetap bisa membuat user interface web dengan HTML, CSS dan JavaScript saja. Jadi kenapa harus menggunakan React?

Untuk menjawab pertanyaan ini, mari bahas sekilas tentang evolusi user interface di web programming.

Melihat dari sejarah, JavaScript awalnya dikembangkan sebagai "bahasa pelengkap" untuk menambah efek interaktif ke HTML. Pemrosesan utama halaman web tetap dilakukan di sisi server menggunakan bahasa server seperti PHP, ASP, atau Java.

Alur yang terjadi adalah ketika user men-klik sebuah link, web browser akan meminta file HTML baru ke web server. Setiap kali proses ini terjadi, halaman web akan dimuat ulang dan berganti menjadi halaman baru. Mayoritas web yang ada di internet saat ini masih memakai prinsip yang sama. Ciri khasnya, halaman web akan reload begitu kita men-klik link atau mengisi form.

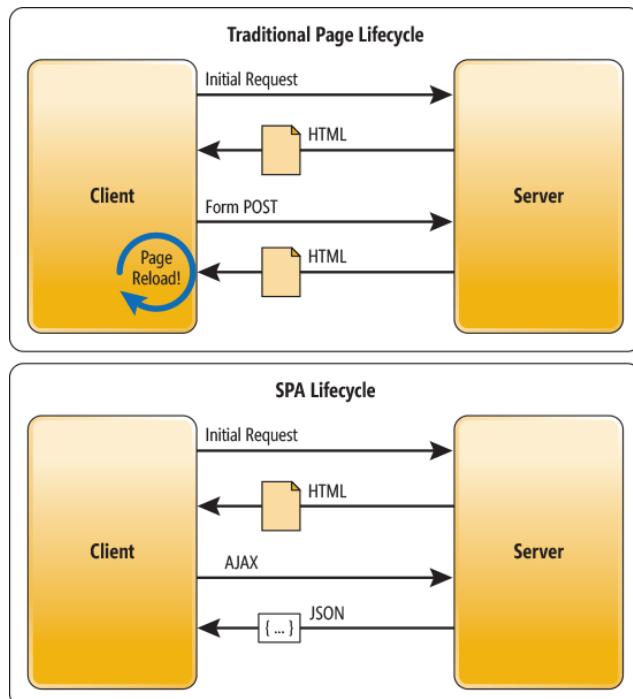
Di tahun 1999, Microsoft merilis web browser Internet Explorer 5 dengan fitur **XMLHttp Request**. Fitur ini memungkinkan JavaScript bisa berkomunikasi langsung ke server tanpa perlu men-load seluruh halaman. Ini cukup revolusioner dan membuka banyak cara baru dalam berinteraksi di suatu website.

Sesaat setelah itu muncul web yang memanfaatkan penuh XMLHttpRequest, di antara yang paling terkenal adalah Gmail. Pada saat kita mengakses gmail, rasanya lebih mirip aplikasi dekstop daripada website. Halaman web tidak butuh refresh ketika sebuah menu di klik, padahal itu sebuah link yang dibuat dari tag <a> HTML biasa.

Saat membuka web Gmail, pada dasarnya kita mengakses satu halaman saja. Yang berubah hanya bagian-bagian tertentu dari halaman tersebut. Karena itulah web seperti gmail disebut dengan istilah **SPA** (Single Page Application). Web SPA terasa responsive dan lebih cepat sebab web server tidak perlu mengirim halaman baru setiap kali link di klik, tapi cukup data yang diminta JavaScript saja.

Di sisi programming, membuat SPA dengan JavaScript murni terbilang cukup sulit. Terlebih saat itu masih banyak perbedaan implementasi dari satu web browser dengan web browser lain (efek browser war antara IE vs Netscape/Firefox). Inilah yang menjadi alasan munculnya library JavaScript seperti **jQuery** yang memudahkan penulisan kode antar web browser.

Sejak awal 2010 hingga beberapa tahun setelahnya, jQuery sangat populer dan memiliki fitur AJAX (Asynchronous Javascript and XML) yang memudahkan penulisan XMLHttpRequest.



Prinsip kerja "web tradisional" vs SPA (sumber gambar: [medium](#))

Akan tetapi seiring kompleksitas website modern, jQuery mengalami kendala saat memproses halaman SPA yang makin kompleks. Di sisi lain, JavaScript terus di update dan memiliki fitur-

fitur baru yang bisa menggantikan banyak fungsi jQuery.

Disinilah **React** hadir sebagai "versi modern" dari jQuery terutama dalam pembuatan SPA. React menerapkan konsep *virtual DOM* yang lebih efisien, serta memecah halaman web menjadi komponen-komponen kecil yang saling bekerja sama satu sama lain.

Jadi, fungsi utama React adalah mempermudah pembuatan SPA (Single Page Application). Jika web yang akan anda buat tidak butuh fitur ini, maka juga tidak perlu menggunakan React.

SPA (Single Page Application) vs MPA (Multi Page Application)

Jika web dengan fitur full AJAX disebut sebagai SPA (Single Page Application), maka "web tradisional" bisa disebut sebagai MPA (Multi Page Application).

Beberapa sumber ada yang menyebut kalau SPA adalah teknologi masa depan. Salah satu bukti kuat, mayoritas web perusahaan besar sudah beralih ke SPA seperti Facebook, Instagram, Twitter, serta semua produk Google (Youtube, Google Drive, Google Map, dll). Web besar dan startup Indonesia juga banyak yang menerapkannya.

Namun bukan berarti SPA tidak memiliki kelemahan. Salah satu yang sering dibahas adalah berkaitan dengan SEO (Search Engine Optimization). Search engine seperti Google masih kesulitan meng-index web yang sepenuhnya di generate menggunakan JavaScript. Di sini, MPA atau web tradisional masih lebih mudah di index Google.

Untuk masalah *loading time*, web SPA memang lebih cepat karena tidak perlu memuat ulang seluruh halaman. Akan tetapi proses loading awal (saat halaman pertama dibuka) butuh waktu yang lebih lama karena harus mendownload banyak kode JavaScript.

Serta bagaimana untuk web browser yang tidak mendukung JavaScript (atau JavaScriptnya di matikan)? Well... web SPA tidak akan bisa berjalan sama sekali.

Atas beberapa alasan itulah sampai sekarang kita masih banyak menemukan web MPA. Teknologi untuk membuat web SPA juga masih relatif baru dan mulai booming sejak kemunculan library/framework JS seperti React atau Vue dalam beberapa tahun ini.

Bagi rekan-rekan yang ingin mendalami web programming atau ingin berkarier sebagai web programmer profesional, menurut saya teknologi SPA tetap menjadi salah satu skill yang harus dimiliki. Memang tidak semua web cocok dibuat menjadi SPA, akan tetapi kita harus punya skill jika client meminta web seperti itu.

1.4. React vs Vue vs Angular

Sebagaimana layaknya library dan framework, tersedia banyak pilihan yang bisa kita pakai. React termasuk kelompok library/framework JavaScript *front-end* dan bukan satu-satunya. Saya yakin pembaca buku ini sudah pernah mendengar framework **Vue** yang menjadi pesaing terdekat **React**, atau alternatif lain seperti **Angular**.

Setiap framework/library selalu punya kelebihan dan kelemahan masing-masing. Mengukur mana yang lebih baik akan cukup sulit karena punya sisi pro dan kontra tersendiri.

Misalnya React memilih jalur "minimalis" dan menyebut diri sebagai library. Ini membuatnya lebih sederhana namun tidak lengkap (butuh library tambahan). Di sisi lain framework Angular hadir dengan paket lengkap, namun terasa lebih kompleks.

Karena keterbatasan waktu, umumnya kita tidak bisa mempelajari semua framework ini dan terpaksa memilih satu atau dua. Jadi yang mana sebaiknya dipelajari? Apakah React, Vue, Angular, atau yang lain?

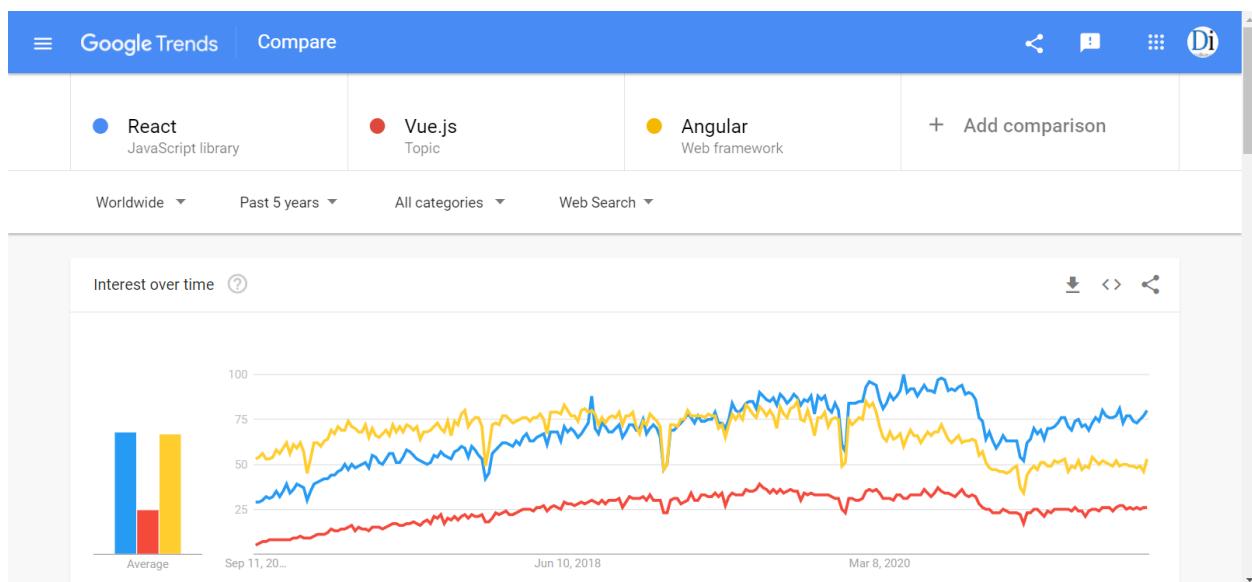
Faktor utama yang menjadi pertimbangan saya pribadi adalah **tingkat popularitas**. Semakin populer framework tersebut, (seharusnya) memiliki masa depan yang lebih terjamin dibandingkan framework yang kurang dikenal.

Meskipun agak jarang, bisa saja framework yang punya fitur bagus ternyata berumur pendek karena komunitasnya tidak berkembang (hanya hadir 1 atau 2 tahun, kemudian hilang).

Sehingga akan lebih aman jika kita mempelajari framework yang banyak dipakai saja.

Terdapat berbagai cara untuk mencari tingkat popularitas ini. Paling mudah, bisa menggunakan [Google Trends](#). Hasil yang didapat bisa memperlihatkan berapa banyak framework tersebut di ketik pada kotak pencarian Google. Semakin banyak yang mencari, berarti semakin populer.

Berikut hasil yang saya dapat ketika membandingkan kata kunci "[React](#)", "[Vue.js](#)" dan "[Angular](#)".

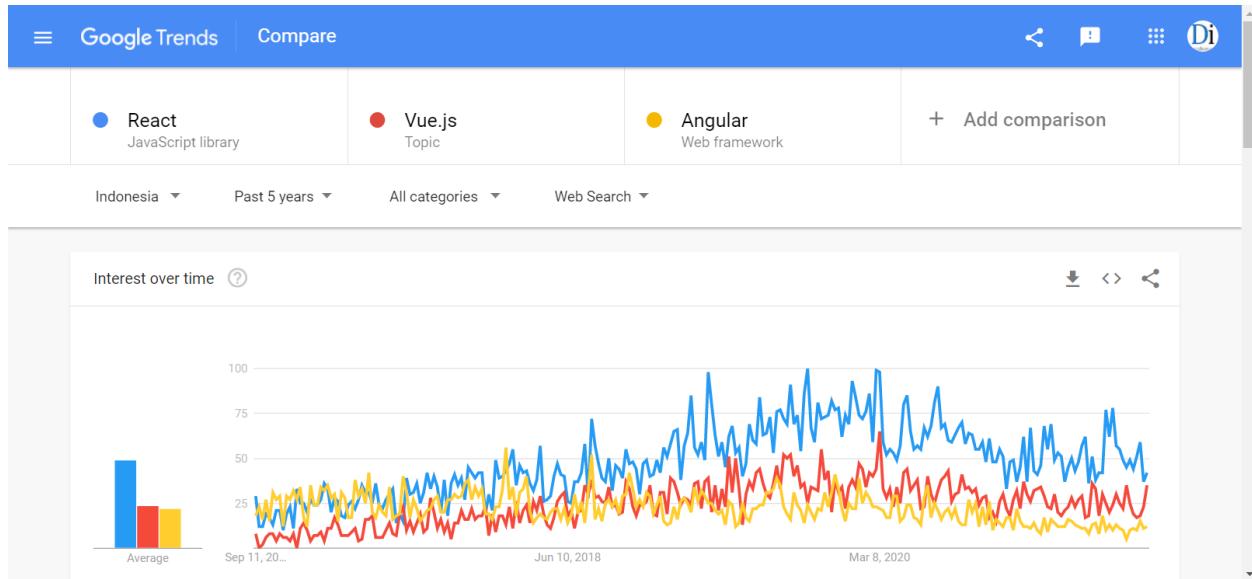


Gambar: Hasil Google Trend worldwide untuk kata kunci "React", "Vue.js" dan "Angular" dalam 5 tahun terakhir

Hasilnya, React menjadi framework JavaScript front-end nomor 1, yang diikuti Angular di tempat kedua dan baru Vue di nomor 3. Ini sedikit mengejutkan karena selama ini saya merasa Vue lebih populer dibandingkan Angular, terutama saat berinteraksi di group-group

programming Indonesia.

Dan itu sebenarnya tidak salah. Ketika cakupan pencarian di tukar dari worldwide menjadi Indonesia, hasilnya juga berubah. Sekarang Vue berada di belakang React:



Gambar: Hasil Google Trend Indonesia untuk kata kunci "React", "Vue.js" dan "Angular" dalam 5 tahun terakhir

Terlepas dari posisi 2 dan 3, React tetap menjadi framework/library front-end paling populer berdasarkan Google Trends.

Cara kedua yang juga bisa jadi pertimbangan (terutama bagi mahasiswa) adalah mencari jumlah lowongan kerja dari setiap framework. Bagi programmer freelance ini mungkin tidak terlalu penting karena bisa memilih alur teknologi sendiri, akan tetapi bagi yang ingin bekerja di perusahaan, tentu berharap kalau skill yang kita miliki sesuai dengan kebutuhan dunia kerja.

Untuk itu saya akan memakai fitur pencarian lowongan kerja di platform LinkedIn. Setelah login, silahkan klik menu "Jobs" di bagian atas, lalu ketik keyword yang ingin dicari. Berikut hasil yang saya temukan pada tanggal 11 September 2021:

This screenshot shows the LinkedIn Jobs search interface. The search bar has three terms: 'react', 'vue', and 'angular', with 'Indonesia' selected as the location. Below the search bar are several dropdown filters: 'Jobs', 'Date Pos', 'Jobs', 'Date F', 'Jobs', 'Date Posted', 'Experience Level', and 'Com'. The results are displayed in three columns: 'React in Indonesia' (877 results), 'Vue in Indonesia' (236 results), and 'Angular in Indonesia' (308 results). Each column shows a summary card with the company logo, name, location, and a 'Job Alert Off' toggle. Underneath each summary are specific job listings. For example, the first job in the React column is for 'PT PERSIS Solo' in Surakarta, Central Java, posted 3 days ago with 4 applicants.

Gambar: Jumlah lowongan kerja untuk keyword "React", "Vue" dan "Angular" di LinkedIn

- ✓ **React:** 877 lowongan kerja
- ✓ **Vue:** 237 lowongan kerja
- ✓ **Angular:** 309 lowongan kerja

Sebagai tambahan, ketika mengetik kata "React" di lowongan kerja LinkedIn, sebenarnya hasil yang didapat tidak murni tentang React di web programming saja, tetapi sudah bercampur dengan React Native, yakni library react untuk membuat aplikasi android.

Agar lebih adil, saya lanjut mencari jumlah lowongan kerja untuk "React"Native", dan hasilnya ada 314. Maka lowongan kerja untuk React sebagai framework web programming adalah 877 - 314 = 563. Ini tetap memperlihatkan kalau React menjadi framework yang lowongan kerjanya paling banyak.

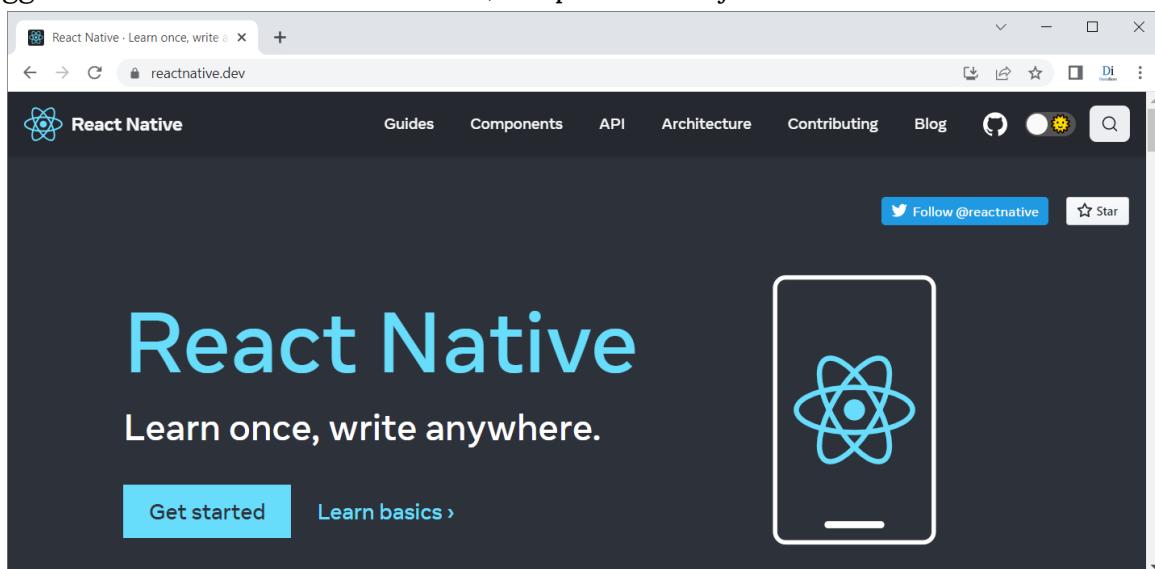
Dua hasil ini (Google Trends dan LinkedIn) rasanya cukup sebagai bukti kalau React layak dipelajari dibandingkan framework JavaScript front-end lain.

Akan tetapi bukan berarti kalau Vue dan Angular tidak layak dipelajari. Tingkat popularitas sebuah teknologi akan terus berubah. Jika ternyata setelah mempelajari React anda merasa ini kurang cocok, tidak masalah coba belajar framework JavaScript lain.

Diluar itu juga ada beberapa framework JS yang patut diperhatikan, antara lain **Svelte**, **Alpine.js**, **Preact**, dan **LitElement**. Namun framework ini masih relatif baru dan lebih cocok dipelajari sebagai tambahan, bukan sebagai framework utama.

Belajar React untuk ke React Native

Selain alasan di atas, rasanya kurang pas jika tidak menyebut tentang [React Native](#)⁴. **React Native** adalah library tambahan yang memungkinkan kita membuat aplikasi Android dan iOS menggunakan React. Untuk bisa ke sana, tetap harus belajar React dasar terlebih dahulu.



Gambar: Tampilan halaman home React Native di <https://reactnative.dev>

4. <https://reactnative.dev>

Daripada repot-repot belajar bahasa Java, Kotlin atau Swift, kita sebagai web programmer tinggal lanjut belajar React saja. Ini menjadi "shortcut" bagi yang ingin mencoba mobile programming menggunakan JavaScript.

Vue pun sebenarnya punya [Vue Native](#)⁵, tapi belum sepopuler React Native.

Dalam buku React Uncover ini kita belum membahas tentang React Native. Rencana saya itu akan menjadi buku terpisah yang mudah-mudahan bisa menyusul dalam waktu dekat.

Facebook Yang Ada di Belakang React

Fakta bahwa React dibuat pertama kali oleh programmer Facebook dan didukung penuh Facebook, bisa mendatangkan manfaat dan kerugian tersendiri.

Dengan perusahaan sebesar Facebook ada di balik React, masa depannya menjadi lebih terjamin. Banyak proyek open source terpaksa berhenti karena tidak ada dukungan finansial yang kuat (hanya mengandalkan donatur kecil saja).

Akan tetapi ini juga bisa menjadi masalah karena Facebook dapat mengontrol dan membatasi penggunaan React di masa depan. Ini pernah terjadi terkait masalah lisensi.

Pada saat rilis awal di 2013, React menggunakan Apache License 2.0, yang kemudian diganti menjadi BSD License 2.0 pada tahun 2014.

Kedua lisensi ini cukup sering dipakai dalam project open source, akan tetapi Facebook menambah aturan bahwa "Jika seseorang / sebuah perusahaan menuntut Facebook di pengadilan (terkait sengketa paten), perusahaan tersebut tidak boleh lagi menggunakan React" (sumber: [Facebook's open source React library is increasingly worrying devs](#)⁶).

Ini menimbulkan perdebatan di kalangan developer. Ada yang pro karena toh React memang dibuat oleh Facebook, jadi beberapa batasan tidak masalah. Namun ada juga yang kontra dengan pendapat bahwa yang mengembangkan React sekarang tidak hanya programmer Facebook saja, tapi juga programmer di luar Facebook secara suka rela.

Akhirnya beberapa developer memutuskan tidak mau menggunakan React sama sekali. Salah satunya **Automattic**, perusahaan dibalik CMS WordPress yang tidak jadi menggunakan React untuk Gutenberg editor, padahal sudah jauh-jauh hari dikembangkan (sumber: [On React and WordPress](#)⁷).

Setelah beberapa waktu kemudian, Facebook mengubah lisensi React menjadi MIT license di tahun 2017 (sumber: [Facebook re-licenses React under MIT license after developer backlash](#)⁸). Dengan demikian masalah tuntutan tadi sudah tidak ada lagi.

5. <https://vue-native.io/>

6. <https://thenextweb.com/news/should-developers-be-afraid-of-zuckerbergs-bearing-gifts>

7. <https://ma.tt/2017/09/on-react-and-wordpress/>

8. <https://thenextweb.com/news/facebook-re-licenses-react-mit-license-developer-backlash>

Adanya perusahaan komersial besar di balik sebuah project open source memang menjadi "warna" tersendiri di dunia programming.

Framework Angular di backing oleh Google, dan Vue sendiri ditulis oleh mantan programmer Google (namun bukan produk Google). Setahu saya Vue belum memiliki dukungan perusahaan besar, lebih ke komunitas saja.

Bagi developer open source yang "idealis", faktor dukungan perusahaan sebisa mungkin di hindari. Namun bagi kita sebagai developer kecil-kecilan, rasanya itu tidak masalah. Jika teknologi tersebut menawarkan peluang besar dan tidak menyalahi aturan, silahkan diikuti. Toh sehari-hari kita juga menggunakan produk dari perusahaan tersebut (Google, Facebook, Microsoft, dll).

Itulah sekelumit perkenalan dengan React. Semoga bisa memberi gambaran tentang apa itu React, sejarah singkatnya serta perbandingan popularitas React dengan library / framework JavaScript lain.

Untuk bab selanjutnya kita akan bahas sekilas tentang konsep JavaScript modern (ES6) yang nantinya diperlukan saat mulai menulis kode program React.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

2. JavaScript (ES6) untuk React

React adalah library JavaScript, sehingga bisa menulis kode JavaScript menjadi skill dasar yang harus dimiliki. Tidak hanya "JavaScript tradisional", React juga banyak menerapkan fitur terbaru dari ECMAScript 6 (ES6).

Karena alasan itulah saya ingin menyajikan panduan singkat dasar-dasar ES6 yang akan banyak kita pakai nantinya. Materi ini juga bisa jadi *refresher* atau penyegar bagi yang sudah lama tidak menulis perintah JavaScript.

Diantaranya kita akan bahas tentang *let*, *const*, *template string*, *arrow notation*, *spread operator*, *JavaScript module* hingga *promise*.

Bagi teman-teman yang sudah tidak sabar ingin langsung belajar React, boleh saja melompati bab ini dan kembali lagi saat butuh penjelasan tentang perintah ES6.

Dan agar tidak terlalu panjang, materi ES6 saya bahas dengan cukup ringkas. Jika butuh panduan yang lebih lengkap, tersedia di buku [JavaScript Uncover](#)⁹.

2.1. Membuat Variabel Dengan *let*

Sejak awal kemunculan JavaScript, keyword *var* biasa dipakai membuat variabel. Akan tetapi *var* memiliki beberapa kelemahan, diantaranya tidak mendukung konsep *scope* (pembatasan hak akses). Karena masalah inilah ES6 menghadirkan keyword baru untuk membuat variabel, yakni "*let*".

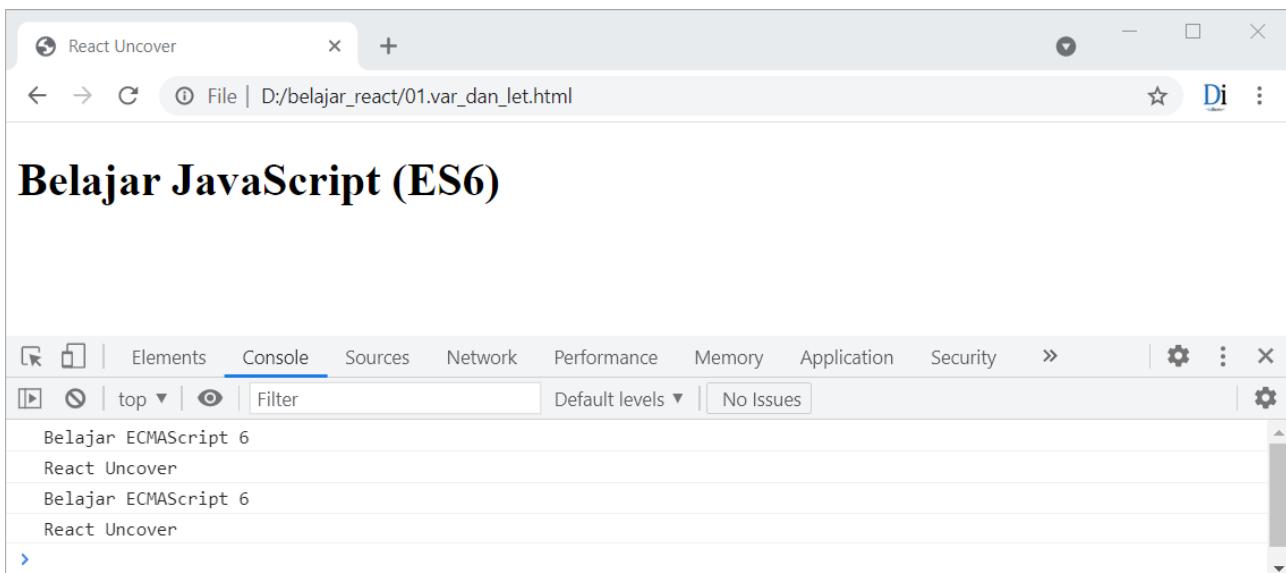
Untuk penggunaan dasar, hampir tidak ada perbedaan antara *let* dan *var* ketika membuat variabel:

```
01.var_dan_let.html
1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>React Uncover</title>
7  </head>
8  <body>
9    <h1>Belajar JavaScript (ES6)</h1>
```

9. <https://www.duniaIlkom.com/javascript-uncover-panduan-belajar-javascript-untuk-pemula/>

JavaScript (ES6) untuk React

```
10 <script>
11   var foo = "Belajar ECMAScript 6";
12   console.log(foo); // Belajar ECMAScript 6
13
14   foo = "React Uncover";
15   console.log(foo); // React Uncover
16
17   let bar = "Belajar ECMAScript 6";
18   console.log(bar); // Belajar ECMAScript 6
19
20   bar = "React Uncover";
21   console.log(bar); // React Uncover
22 </script>
23 </body>
24 </html>
```



Gambar: Membuat variabel dengan var dan let

Di awal kode program, variabel `foo` saya buat dengan perintah `var`, lalu memodifikasi nilainya pada baris 14.

Hal yang sama juga dilakukan untuk variabel `bar` yang dibuat dengan perintah `let` di baris 17, kemudian diubah nilainya pada baris 20. Terlihat nilai kedua variabel sukses disimpan dan juga bisa ditimpas.

Perbedaan utama antara `var` dan `let` ada di konsep scope. Dalam programming, **scope** merujuk ke ruang lingkup dimana sebuah variabel masih bisa diakses. Sebuah scope umumnya dibatasi dengan tanda kurung kurawal "{" dan "}". Tanda ini juga dipakai ketika membuat perulangan `for`, `while`, kondisi `if`, maupun `function`.

Pada mayoritas bahasa pemrograman, variabel yang ada di dalam scope tidak bisa diakses dari luar scope. Akan tetapi ini tidak berlaku untuk `var` di JavaScript:

02.var_vs_let_scope_1.html

```
1  {
2    var foo = "Belajar JavaScript";
3    let bar = "Belajar React";
4  }
5
6 console.log(foo); // Belajar JavaScript
7 console.log(bar); // Uncaught ReferenceError: bar is not defined
```

Untuk menghemat tempat, saya tidak lagi menampilkan kode HTML lengkap, tapi hanya potongan perintah JavaScript seperti contoh di atas.

Jika anda ingin menjalankannya, silahkan copy kode tersebut ke dalam tag <script>, lalu buka tab Console pada Developer Tools di web browser. File kode program ini juga tersedia pada file [belajar_react.zip](#) di Google Drive.

Dalam contoh di atas saya membuat sebuah scope antara baris 1 – 4, kemudian mendeklarasikan variabel `foo` dan `bar` di dalamnya. Ternyata `foo` tetap bisa diakses dari luar scope (baris 6), namun akan error ketika mengakses `bar` yang dideklarasikan dengan `let` (baris 7).

Sekilas error ini tampak menjadi masalah, namun itulah yang kita inginkan. Konsep scope akan memproteksi suatu variabel agar tidak bisa diakses dari luar. Contoh berikut memperlihatkan masalah lain yang bisa terjadi:

03.var_vs_let_scope_2.html

```
1 var i = "Belajar JavaScript";
2
3 for (var i = 1; i <= 5; i++) {
4   console.log(i); // 1, 2, 3, 4, 5
5 }
6
7 console.log(i); // 6
```

Di baris 1, variabel `i` saya isi dengan string "Belajar JavaScript". Dalam konsep programming secara umum, variabel `i` ini berada di *global scope* karena tidak menjadi bagian dari suatu scope.

Setelah itu terdapat perulangan `for` yang juga memakai variabel `i` sebagai variabel counter. Perhatikan bahwa meskipun pada baris 3 variabel `i` di deklarasikan di luar kurung kurawal, tapi itu ada di dalam scope perulangan `for`.

Setelah perulangan, saya memeriksa isi variabel `i` di baris 7. Hasilnya, nilai `i` sudah bertukar menjadi 6 karena efek perintah `i++` dari perulangan `for`, perilaku inilah yang kadang tidak terduga di JavaScript. Di kebanyakan bahasa pemrograman lain, isi *global scope* seharusnya tidak bisa diubah oleh scope yang lebih kecil.

Mari kita ganti deklarasi variabel `i` menggunakan `let`:

04.var_vs_let_scope_3.html

```
1 let i = "Belajar JavaScript";
2
3 for (let i = 1; i <= 5; i++) {
4   console.log(i); // 1, 2, 3, 4, 5
5 }
6
7 console.log(i); // Belajar JavaScript
```

Sekarang variabel `i` di baris 7 tetap berisi string "Belajar JavaScript". Di sini perintah `i++` dalam perulangan `for` tidak mempengaruhi isi variabel `i` yang ada di *global scope*.

Perbedaan konsep scope antara `var` dan `let` memang tidak selalu menjadi masalah, namun perilaku `var` yang di luar kebiasaan membuat mayoritas programmer JavaScript lebih memilih `let` daripada `var`. Namun baik `var` dan `let` tetap bisa dipakai untuk membuat variabel di dalam JavaScript, selama paham letak perbedaannya.

2.2. Membuat Konstanta Dengan `const`

Selain `let`, ES6 menghadirkan perintah `const` untuk membuat **konstanta**. Konsep konstanta sering dijumpai dalam berbagai bahasa pemrograman lain. Fungsi `const` adalah membuat "variabel" yang nilainya tidak bisa diubah setelah di deklarasikan.

Berikut contoh penggunaan perintah `const` dalam JavaScript:

05.const.html

```
1 let foo = "Belajar JavaScript";
2 const bar = "Belajar JavaScript";
3
4 foo = "Belajar React"; // Uncaught TypeError: Assignment to constant variable.
5 bar = "Belajar React";
```

Hasilnya terjadi error di baris 5 ketika saya coba menukar nilai konstanta `bar`. Karena seperti pengertiannya, nilai yang tersimpan dalam konstanta tidak bisa diubah setelah di deklarasikan.

Saat ini mayoritas programmer JavaScript lebih condong ke konsep **immutable**, dimana sebuah variabel sebaiknya tidak diubah. Jika kita butuh nilai baru, simpan nilai tersebut ke variabel lain agar nilai lama tidak rusak dan masih bisa diakses. Karena alasan inilah perintah `const` akan lebih banyak kita temui pada kode JavaScript modern, termasuk di dalam React.

2.3. Template String

Template string (dikenal juga dengan istilah *template literals* atau *string interpolation*) adalah alternatif penulisan string yang memudahkan kita mengakses variabel serta menjalankan suatu *expression* saat berada di dalam string.

Contoh berikut akan memperjelas pengertian *template string* di JavaScript:

06.template_strings.html

```
1 const foo = "JavaScript";
2 const bar = "React";
3
4 console.log("Belajar " + foo + " dan " + bar); // Belajar JavaScript dan React
5
6 console.log(`Belajar ${foo} dan ${bar}`); // Belajar JavaScript dan React
```

Di baris 1 dan 2 saya mendeklarasikan konstanta `foo` dan `bar` serta mengisinya dengan string `"JavaScript"` dan `"React"`.

Jika menggunakan cara biasa, kedua konstanta bisa diakses dari dalam string dengan pemisah operator `" + "` seperti di baris 4. Dengan penulisan ini, ketika kita ingin mengakses variabel atau konstanta, maka harus keluar dari tanda kutip dua, diikuti operator `" + "` dan baru tulis variabel yang akan diakses.

Versi *template string* ada di baris 6. Caranya, awali penulisan string dengan tanda *backtick* ```. Karakter ini ada di sebelah kiri angka 1 pada keyboard (di atas tombol tab). Setelah itu jika ingin menampilkan isi variabel, tulis karakter dollar `$` diikuti nama variabel dalam tanda kurung kurawal `{ }`. Jika anda sudah pernah mempelajari bahasa PHP, cara penulisan seperti ini tentunya sangat familiar.

Selain menampilkan variabel atau konstanta, nilai dalam tanda `${ }` kepuanyaan *template string* juga bisa diisi dengan *expression*. Dalam programming, **expression** adalah suatu perintah yang bisa menghasilkan sebuah nilai. Ini membuat *template string* menjadi sangat praktis karena kita bisa menjalankan kode JavaScript langsung di dalam string:

07.template_strings_expression.html

```
1 console.log(`10 tambah 7 adalah ${10 + 7}`); // 10 tambah 7 adalah 17
```

Di sini, perintah `${10 + 7}` akan di proses dan menghasilkan angka 17. JavaScript *expression* juga mencakup pemanggilan function:

08.template_strings_function.html

```
1 function kuadrat(a) {
2   return a*a;
3 }
4
5 console.log(`5 kuadrat adalah ${kuadrat(5)}`); // 5 kuadrat adalah 25
6 console.log(`11 kuadrat adalah ${kuadrat(11)}`); // 5 kuadrat adalah 121
```

Di baris 1 – 3 saya membuat function `kuadrat()`. Function ini butuh satu argument `a` dan mengembalikan nilai berupa hasil perkalian argument tersebut. Di baris 5 dan 6, function `kuadrat()` diakses secara langsung dari *template string*. Penulisan seperti ini akan banyak kita

pakai di React nantinya.

2.4. Conditional Operator

Conditional operator (sering juga disebut sebagai *ternary operator*) adalah operator khusus untuk mempersingkat penulisan kondisi if else sederhana. Operator ini ditulis menggunakan tanda " ? " dan " : ".

Sebagai contoh, silahkan pelajari sejenak perintah if else berikut:

09.if_else.html

```
1 const user = "Admin";
2 let result = "";
3
4 if (user === "Admin") {
5     result = "Welcome, admin";
6 }
7 else {
8     result = "User not found";
9 }
10 console.log(result); // Welcome, admin
```

Dalam kode ini, kondisi if else dipakai untuk memeriksa nilai konstanta `user`. Jika `user` berisi string "Admin", maka variabel `result` akan diisi string "Welcome, admin". Jika tidak, variabel `result` akan diisi dengan string "User not found".

Jika memakai conditional operator, perintahnya menjadi lebih singkat:

10.conditional_operator_1.html

```
1 const user = "Admin";
2 const result = (user === "Admin") ? "Welcome, admin" : "User not found";
3
4 console.log(result); // Welcome, admin
```

Penulisan conditional operator mengikuti format berikut:

<variabel penampung> = <kondisi yang diperiksa> : <nilai kembalian jika kondisi terpenuhi (true)> ? <nilai kembalian jika kondisi tidak terpenuhi (false)>

Dalam contoh di atas, konstanta `result` akan berisi string "Welcome, admin" karena kondisi `user === "Admin"` menghasilkan nilai `true`.

Namun jika kondisi yang diperiksa menghasilkan `false` (tidak terpenuhi), konstanta `result` akan diisi string "User not found":

10.conditional_operator_1.html

```
1 const user = "Rudi";
2 const result = (user === "Admin") ? "Welcome, admin" : "User not found";
3
4 console.log(result); // User not found
```

Kondisi yang bisa diperiksa tidak hanya operasi perbandingan saja, tapi juga bisa hasil pemanggilan function, selama function itu mengembalikan boolean true atau false:

11.conditional_operator_2.html

```
1 const input = "100";
2
3 const result1 = Number.isInteger(input) ? "valid" : "invalid";
4 console.log(result1); // invalid
5
6 const result2 = (typeof input === "string") ? "valid" : "invalid";
7 console.log(result2); // valid
```

Pada awal kode program saya mengisi konstanta `input` dengan string "100". Lalu di baris 3 method `Number.isInteger(input)` akan memeriksa apakah nilai yang tersimpan dalam `input` berupa angka integer atau tidak.

`Number.isInteger()` adalah method bawaan JavaScript yang mengembalikan nilai `true` jika argument di isi angka integer, atau `false` jika argument bukan integer.

String "100" yang tersimpan dalam konstanta `input` bukanlah angka, tapi sebuah string. Maka method `Number.isInteger(input)` akan menghasilkan nilai `false` dan konstanta `result1` berisi string "invalid".

Lanjut ke contoh kedua, di baris 6 kondisi yang diperiksa adalah `typeof input === "string"`. Dalam JavaScript, `typeof` adalah operator khusus yang mengembalikan string dari jenis tipe data suatu variabel / konstanta. Karena konstanta `input` berisi string, maka operasi `typeof input === "string"` akan menghasilkan nilai `true` dan konstanta `result2` berisi string "valid".

Selain penulisan yang lebih singkat, `conditional operator` juga sebuah `expression`. Sehingga, bisa kita tulis ke dalam template string:

12.conditional_operator_3.html

```
1 const input = "100";
2 const result = `<div class="${Number.isInteger(input) ? 'valid' : 'invalid'}">`;
3
4 console.log(result); // <div class="invalid">
```

Di baris 2 saya menulis `template string` yang di dalamnya terdapat `conditional operator`. Pemeriksaan kondisi seperti ini tidak bisa ditulis dengan `if else` karena perintah `if else` bukanlah suatu `expression`.

Di react nanti, conditional operator sangat efektif untuk mempersingkat penulisan kode program.

2.5. Short-Circuit Conditionals

Short-circuit conditionals sebenarnya lebih ke "trik" daripada perintah dasar JavaScript. Di sini kita memanfaatkan cara JavaScript memproses operator logika untuk menjalankan sebuah perintah.

Agar bisa memahami konsep ini, silahkan pelajari sejenak kode berikut:

13.logical_operator.html

```
1 let user = "Rudi";
2 let password = "qwerty";
3
4 if ((user === "Admin") && (password === "qwerty")) {
5   console.log("Welcome, admin")
6 }
```

Perintah `if` di baris 4 memeriksa 2 kondisi, yaitu apakah variabel `user` berisi string "Admin" **dan** variabel `password` berisi string "qwerty". Hanya jika keduanya menghasilkan nilai `true`, perintah `console.log()` bisa diproses.

Sebagaimana yang sudah kita ketahui, operasi **and** atau "`&&`" menghasilkan nilai `true` hanya jika kedua kondisi yang diperiksa bernilai `true`. Selain itu hasilnya `false`.

Ketika menjalankan operator logika, JavaScript menerapkan konsep yang disebut sebagai "**short circuit**". Cara kerjanya adalah, jika terdapat operator "`&&`" dan kondisi pertama sudah bernilai `false`, maka kondisi kedua **tidak akan dijalankan lagi**. Inilah kata kunci dari prinsip `short circuit`.

Pada contoh di atas, kondisi `(user === "Admin")` sudah tidak memenuhi atau `false`, maka kondisi `(password === "qwerty")` tidak akan di periksa lagi oleh JavaScript. Karena apapun hasil dari `(password === "qwerty")`, itu tidak akan mempengaruhi hasil akhir.

Lebih jauh lagi, konsep `short-circuit` bisa dipakai untuk menjalankan sebuah perintah hanya ketika suatu kondisi terpenuhi:

14.short_circuit_conditional_1.html

```
1 let user;
2
3 user = "Rudi";
4 (user === "Admin") && console.log("Welcome, admin");
5
6 user = "Admin";
7 (user === "Admin") && console.log("Welcome, admin"); // Welcome, admin
```

Di sini saya menghapus perintah `if` dan hanya menjalankan operasi logika saja. Selain itu "kondisi" kedua adalah sebuah perintah `console.log()`. Perintah `console.log()` hanya akan berjalan jika `(user === "Admin")` bernilai true. Inilah yang dimaksud dengan *short-circuit conditionals*.

Penulisan *short-circuit conditionals* bisa dipakai sebagai alternatif dari *conditional operator*. Sebagai contoh, kedua perintah berikut "nyaris" bermakna sama:

15.short_circuit_conditional_2.html

```
1 let user = "Admin";
2
3 (user === "Admin") ? console.log("Welcome, admin") : ""; // Welcome, admin
4 (user === "Admin") && console.log("Welcome, admin"); // Welcome, admin
```

Perintah di baris 3 adalah sebuah *conditional operator*. Jika `(user === "Admin")` bernilai true, maka jalankan perintah `console.log()`. Jika hasilnya *false*, maka kembalikan string kosong "".

Ini akan lebih singkat dan lebih pas jika ditulis menggunakan *short-circuit conditionals* seperti di baris 4 karena kita tidak perlu mengembalikan nilai apapun jika kondisi yang diperiksa menghasilkan *false*.

Fitur lain, *short-circuit conditionals* juga mendukung lebih dari satu pemeriksaan kondisi:

16.short_circuit_conditional_3.html

```
1 let user = "Admin";
2 let password = "qwerty";
3
4 (user === "Admin") && (password === "12345") && console.log("Welcome, admin");
5
6 (user === "Admin") && (password === "qwerty") && console.log("Welcome, admin");
7 // Welcome, admin
```

Di baris 4 dan 6, perintah `console.log()` hanya akan berjalan jika variabel `user` berisi "Admin" dan variabel `password` berisi string "qwerty".

2.6. Arrow Notation

Arrow notation adalah cara penulisan singkat untuk membuat fungsi (*function*). Selama ini, JavaScript butuh keyword `function` untuk membuat fungsi seperti contoh berikut:

17.normal_function.html

```
1 function kuadrat(a){
2     return a*a;
3 }
4 console.log(kuadrat(5)); // 25
```

Secara internal, function adalah sebuah "first-class citizens" di dalam JavaScript. Istilah ini bermaksud bahwa function bisa diproses sebagaimana layaknya sebuah nilai biasa, termasuk bisa disimpan ke dalam variabel atau dikirim sebagai argument ke function lain.

Dengan konsep ini, penulisan function juga bisa ditulis sebagai berikut:

18.normal_function_const.html

```
1 const kuadrat = function(a){  
2     return a*a;  
3 }  
4 console.log(kuadrat(5)); // 25
```

Di sini saya mengisi konstanta `kuadrat` dengan sebuah function. Tidak harus konstanta, function juga bisa disimpan ke dalam variabel yang dibuat dengan perintah `var` maupun `let`. Sepanjang kode program, `kuadrat` bisa diakses sebagaimana function biasa.

ESMAScript 6 mencoba mempersingkat penulisan ini dengan beberapa cara. Pertama, kita bisa menghapus keyword `function` lalu menambah tanda panah (*arrow*) "`=>`" untuk memulai block function:

19.arrow_notation_1.html

```
1 const kuadrat = (a) => {  
2     return a*a;  
3 }  
4 console.log(kuadrat(5)); // 25
```

Perhatikan kali ini keyword `function` sudah tidak ada lagi. Selain itu setelah penulisan argument "`(a)`" terdapat panah "`=>`" sebagai tanda untuk memulai block function. Karena tanda panah inilah penulisan di atas disebut sebagai "*arrow notation*".

Khusus untuk function yang hanya terdiri dari satu baris dan langsung mengembalikan nilai, perintah `return` dan tanda kurung kurawal "`{ }`" penanda block juga boleh dihapus:

20.arrow_notation_2.html

```
1 const kuadrat = (a) => a*a;  
2 console.log(kuadrat(5)); // 25
```

Dengan ini penulisan function menjadi lebih ringkas. Namun merasa masih belum cukup singkat, ES6 mengizinkan kita menghapus tanda kurung penanda parameter "`()`" untuk function yang hanya punya 1 parameter saja:

21.arrow_notation_3.html

```
1 const kuadrat = a => a*a;  
2 console.log(kuadrat(5)); // 25
```

Inilah cara tersingkat untuk membuat function di JavaScript.

Namun ada beberapa catatan tambahan. Jika ternyata function yang ditulis tidak butuh parameter sama sekali, tanda kurung "()" tetap harus ditulis:

22.arrow_notation_4.html

```
1 const generateRandom = () => Math.floor(Math.random() * 10) + 1;
2 console.log(generateRandom()); // 7
```

Kali ini konstanta `generateRandom` saya isi dengan function yang tidak punya parameter, namun tanda kurung "()" tetap harus ditulis.

Function `generateRandom()` di atas akan menghasilkan angka acak antara 1 – 10 hasil dari method `Math.floor()` dan `Math.random()` bawaan JavaScript. Rumus ini diperlukan karena method `Math.random()` mengembalikan angka acak desimal antara 0 – 1.

Jika function yang ingin ditulis butuh 2 parameter atau lebih, maka parameter tersebut juga harus ditulis dalam tanda kurung:

23.arrow_notation_5.html

```
1 const generateRandom = (start, end) =>
2   Math.floor(Math.random() * (end - start + 1)) + start;
3
4 console.log(generateRandom(10, 20)); // 14
```

Kali ini fungsi `generateRandom()` perlu 2 parameter yang disimpan ke dalam variabel `start` dan `end`. Kedua parameter dipakai untuk menentukan batas awal dan akhir angka random.

Sama seperti function normal, penulisan *arrow notation* juga bisa menerima *default argument*:

24.arrow_notation_6.html

```
1 const generateRandom = (start = 1, end = 10) =>
2   Math.floor(Math.random() * (end - start + 1)) + start;
3
4 console.log(generateRandom());           // 8
5 console.log(generateRandom(5));          // 6
6 console.log(generateRandom(10, 20));     // 17
```

Dengan deklarasi ini maka ketika fungsi `generateRandom()` di panggil tanpa argument, nilai default-lah yang dipakai, yakni 1 untuk `start` dan 10 untuk `end`.

Itulah cara penulisan function dengan format *arrow notation*. Dalam kebanyakan kasus, kita bebas apakah ingin mendeklarasikan function dengan cara biasa atau lewat *arrow notation*. Namun mayoritas programmer JavaScript saat ini lebih menyukai *arrow notation* dibandingkan cara penulisan function biasa.

Sebenarnya ada beberapa perbedaan mendasar antara penulisan function biasa dan *arrow notation*. Salah satunya berkaitan dengan nilai variabel `this`. Ini akan kita bahas

dalam penjelasan tentang JavaScript object sesaat lagi.

2.7. Function Sebagai First-Class Citizens

Di awal pembahasan tentang *arrow notation*, saya sempat menulis function sebagai **first-class citizens**. Selain bisa disimpan ke dalam variabel, sebuah function juga bisa dikirim sebagai argument. Praktek dari konsep ini memang cukup aneh dan membingungkan, namun akan banyak kita temui di React.

Untuk memulai bahasan, silahkan pelajari sejenak maksud dari kode berikut:

```
24.function_first_class_1.html

1  function total (a, b, cari) {
2      return cari(a, b);
3  }
4
5  function tambah (x, y) {
6      return x + y
7  }
8
9  console.log( total(5,10,tambah) ) // 15
```

Bisakah anda jelaskan bagaimana angka 15 tampil sebagai hasil dari pemanggilan fungsi `total(5,10,tambah)`?

Kita berangkat dari pemanggilan `total(5,10,tambah)` di baris 9. Dari pemanggilan ini dapat disimpulkan kalau fungsi `total()` bisa menampung 3 buah argument, yakni angka 5, 10, dan `tambah`.

Tapi apa isi dari variabel `tambah`? Ternyata itu adalah sebuah function yang dideklarasikan di baris 5 – 6. Function `tambah()` ini butuh 2 argument dan mengembalikan angka hasil penambahan kedua argument tersebut.

Ketika pertama kali bertemu kode seperti ini, saya harus berhenti cukup lama untuk bisa memahami bahwa argument `tambah` yang dikirim ke fungsi `total()` adalah sebuah function, bukan nilai atau variabel biasa.

Lanjut ke deklarasi fungsi `total()` di baris 1 – 3. Seperti yang bisa di tebak, function ini punya 3 parameter. Angka 5 akan ditampung oleh parameter `a`, angka 10 ditampung parameter `b`, dan yang perlu menjadi perhatian, function `tambah()` diterima oleh parameter `cari`.

Isi function `total()` sendiri hanya 1 perintah, yakni `return cari(a, b)`. Maka secara tidak langsung yang dijalankan adalah `tambah(5, 10)` dan hasilnya 15.

Konsep mengirim function sebagai argument memang cukup rumit, tapi mau tidak mau harus bisa dipahami.

Masih belum cukup, kode JavaScript modern sering mendeklarasikan function dengan *arrow notation* yang membuat kodennya menjadi lebih singkat:

```
25.function_first_class_2.html  
1 const total = (a, b, cari) => cari(a, b);  
2 const tambah = (x, y) => x + y;  
3  
4 console.log( total(5,10,tambah) ) // 15
```

Makna kode ini sama persis seperti sebelumnya, hanya saja kali ini setiap function ditulis dengan *arrow notation*.

Agar makin ringkas, kita juga bisa menulis deklarasi function langsung di dalam argument:

```
26.function_first_class_3.html  
1 const total = (a, b, cari) => cari(a, b);  
2  
3 console.log( total(5,10,(x, y) => x + y) ) // 15
```

Kali ini fungsi `total()` dipanggil dengan perintah `total(5,10,(x, y) => x + y)`. Di argument ketiga, itu adalah deklarasi function, bukan lagi nama sebuah fungsi.

Memahami kode yang mengirim function sebagai argument memang tidak mudah. Jika masih kurang paham, silahkan baca beberapa kali atau lompati saja untuk sementara dan kembali lagi ketika bertemu praktek dari konsep ini.

2.8. Class dan Object

JavaScript merupakan bahasa pemrograman berorientasi object. Namun konsep object di JavaScript memang berbeda dengan bahasa full OOP seperti Java.

Pada awalnya JavaScript menggunakan konsep *prototype* untuk membuat object, bukan melalui class (*prototype based object*). Namun ES6 memperkenalkan cara membuat object dari class sehingga sudah mirip seperti bahasa pemrograman lain.

Di sini saya tidak akan bahas OOP JavaScript secara mendalam karena bisa sangat panjang dan materi itu sudah ada di buku **JavaScript Uncover**. Namun untuk sekedar penyegaran, berikut contoh cara membuat class dan instansiasi object di JavaScript:

```
27.class.html  
1 class Laptop {  
2   constructor (milik, merk){  
3     this.milik = milik  
4     this.merk = merk  
5   }  
6   hidupkanLaptop(){
```

```
7     return `Hidupkan laptop ${this.merk} milik ${this.milik}`;
8 }
9 }
10
11 let laptopRudi = new Laptop("Rudi", "Asus");
12 console.log(laptopRudi.merk);           // Asus
13 console.log(laptopRudi.milik);          // Rudi
14 console.log(laptopRudi.hidupkanLaptop()); // Hidupkan laptop Asus milik Rudi
```

Antara baris 1 – 9 merupakan kode untuk deklarasi class Laptop. Di dalam class laptop terdapat constructor dengan 2 parameter yang saya simpan ke dalam variabel `milik` dan `merek`. Isi constructor sendiri hanya memindahkan kedua parameter ke dalam property `this.milik` dan `this.merk`.

Class Laptop juga memiliki method `hidupkanLaptop()` untuk menampilkan string. String ini dibuat dari *template string* dan mengakses property `this.milik` dan `this.merk`.

Di baris 10, class Laptop saya instansiasi ke dalam variabel `laptopRudi` untuk kemudian mengakses semua property dan method yang ada di dalamnya menggunakan perintah `console.log()`. Inilah cara pembuatan object sederhana di dalam JavaScript.

Fitur lain dari OOP JavaScript adalah, sebuah object bisa dibuat tanpa class:

28.object_1.html

```
1 let laptopRudi = {
2   milik: "Rudi",
3   merk: "Asus",
4   hidupkanLaptop(){
5     return `Hidupkan laptop ${this.merk} milik ${this.milik}`;
6   }
7 }
8
9 console.log(laptopRudi.merk);           // Asus
10 console.log(laptopRudi.milik);          // Rudi
11 console.log(laptopRudi.hidupkanLaptop()); // Hidupkan laptop Asus milik Rudi
```

Sekarang variabel `laptopRudi` langsung saya isi dengan object. Jika menggunakan cara seperti ini, property object bisa langsung ditulis tanpa `this` (baris 2-3), akan tetapi jika ingin mengakses property dari dalam method, tetap butuh keyword `this`, yakni `this.merk` dan `this.milik` seperti di baris 5.

Sebagai tambahan, penulisan method di dalam object juga bisa dibuat dari *arrow notation*:

29.object_2.html

```
1 let laptopRudi = {
2   milik: "Rudi",
3   merk: "Asus",
4   hidupkanLaptop : () => `Hidupkan laptop Asus milik Rudi`
5 }
```

```

6
7 console.log(laptopRudi.merk);           // Asus
8 console.log(laptopRudi.milik);          // Rudi
9 console.log(laptopRudi.hidupkanLaptop()); // Hidupkan laptop Asus milik Rudi

```

Perhatikan perintah di baris 4, itulah cara mengisi method object menggunakan *arrow notation*.

Penulisan method object dengan *arrow notation* harus menjadi perhatian, karena kita sampai ke salah satu perbedaan mendasar antara penulisan method dengan cara biasa atau memakai *arrow notation*. Perbedaan itu terkait dengan nilai variabel *this*:

30.object_this.html

```

1 let laptopRudi = {
2   milik: "Rudi",
3   merk: "Asus",
4   foo () { return this },
5   bar: () => this
6 }
7
8 console.log( laptopRudi.foo() ); // {milik: "Rudi", merk: "Asus"...
9 console.log( laptopRudi.bar() ); // Window {window: Window, ...

```

Dalam konsep OOP secara umum, *this* adalah keyword khusus yang merujuk ke *instance* dari object saat ini. Method *foo()* di baris 4 hanya berisi perintah *return this* yang selanjutnya ditampilkan oleh perintah *console.log()*. Hasilnya, variabel *this* berisi object *laptopRudi*. Ini merupakan hasil yang biasanya kita inginkan.

Akan tetapi jika method object dibuat menggunakan *arrow notation* seperti method *foo()* di baris 4, keyword *this* di dalam method merujuk ke **Window object**, bukan instance object. Ini bisa mendatangkan hasil yang tidak terduga seperti contoh berikut:

30a.object_this_arrow_problem.html

```

1 let laptopRudi = {
2   milik: "Rudi",
3   merk: "Asus",
4   hidupkanLaptop : () => `Hidupkan laptop ${this.merk} milik ${this.milik}`
5 }
6
7 console.log(laptopRudi.merk);           // Asus
8 console.log(laptopRudi.milik);          // Rudi
9 console.log(laptopRudi.hidupkanLaptop());
10 // Hidupkan laptop undefined milik undefined

```

Pemanggilan method *laptopRudi.hidupkanLaptop()* di baris 9 menghasilkan "Hidupkan laptop *undefined* milik *undefined*". Alasan tampil "*undefined*" adalah karena keyword *this.merk* dan *this.milik* tidak merujuk ke object *laptopRudi*, tapi ke *Window object*.

Umumnya hasil seperti ini tidak diharapkan, oleh karena itu jika butuh mengakses *this* di

dalam method, tetap tulis dengan cara biasa dan jangan pakai *arrow notation*.

Di React nanti kita tetap bisa menggunakan arrow notation ketika membuat method di dalam object. Ini karena React punya kode internal untuk mengisi atau men-binding variabel *this* agar sesuai dengan konsep di React. Materi ini nantinya akan kita bahas pada bab tentang **React Event**.

2.9. Memproses Object

Di dalam JavaScript, property serta method sebuah object bisa ditambah dan dimodifikasi setelah di deklarasikan. Berikut contoh kode yang dimaksud:

31.object_addition.html

```
1 let mahasiswa = {  
2   nama: "Eka",  
3   umur: 19,  
4   jurusan: "Teknik Informatika",  
5   getInfo() {  
6     return `${this.nama} (${this.umur} tahun) dari jurusan ${this.jurusan}`;  
7   }  
8 }  
9  
10 console.log( mahasiswa );  
11 // {nama: "Eka", umur: "19", jurusan: "Teknik Informatika", getInfo: f}  
12  
13 mahasiswa.umur = 20;  
14 mahasiswa.tempatLahir = "Jakarta";  
15  
16 console.log( mahasiswa );  
17 // {nama: "Eka", umur: 20, jurusan: "Teknik Informatika",  
18 // tempatLahir: "Jakarta", getInfo: f}
```

Setelah membuat object mahasiswa di baris 1 – 8, saya menukar nilai property *umur* dan menambah property *tempatLahir* di baris 13 dan 14. Hasilnya, struktur object ikut berubah. Konsep mengisi object ini akan sangat berguna karena di React nanti kita cukup sering menyimpan data dalam bentuk object.

JavaScript juga menyediakan perulangan **for in** untuk mengakses setiap property dan method object. Format dasar perulangan *for of* adalah sebagai berikut:

```
for (<variabel penampung property/method> in <nama object>) {  
  ...  
}
```

Perulangan ini sangat praktis ketika butuh menampilkan isi object yang cukup besar:

32.object_for_in.html

```
1 let mahasiswa = {  
2   nama: "Eka",  
3   umur: 19,  
4   jurusan: "Teknik Informatika",  
5 }  
6  
7 for (prop in mahasiswa) {  
8   console.log(`Nilai ${prop} : ${mahasiswa[prop]}`);  
9 }  
10  
11 // Nilai nama : Eka  
12 // Nilai umur : 19  
13 // Nilai jurusan : Teknik Informatika
```

Dalam contoh ini saya menggunakan variabel `prop` untuk menampung nama property / nama method. Di dalam perulangan `for of`, variabel `prop` bisa dipakai untuk mengakses nama setiap property / method. Jika diinginkan, kita bebas ingin mengganti nama variabel `prop` dengan nama lain.

Untuk memproses object lebih jauh lagi, ECMAScript 6 menghadirkan 3 method baru:

- `Object.keys`: menampilkan semua nama key property/method.
- `Object.values`: menampilkan semua nilai property/method.
- `Object.entries`: menampilkan semua nama dan nilai property/method.

Berikut contoh penggunaannya:

33.object_method.html

```
1 let mahasiswa = {  
2   nama: "Eka",  
3   umur: 19,  
4   jurusan: "Teknik Informatika",  
5 }  
6  
7 console.log ( Object.keys(mahasiswa) ); // ["nama", "umur", "jurusan"]  
8 console.log ( Object.values(mahasiswa) ); // [ "Eka", 19, "Teknik Informatika" ]  
9  
10 console.log ( Object.entries(mahasiswa) );  
11   // [  
12   //   ["nama", "Eka"],  
13   //   ["umur", 19],  
14   //   ["jurusan", "Teknik Informatika"]  
15   // ]
```

Hasil akhir dari ketiga method berbentuk array. Khusus untuk `Object.entries`, hasilnya berbentuk array 2 dimensi karena harus menyimpan pasangan nama property dan juga nilainya.

2.10. Array dan Object Destructuring

Array dan object destructuring adalah perintah khusus untuk "membuka" nilai yang tersimpan di dalam sebuah array atau object ke dalam variabel terpisah. Penjelasan ini lebih mudah dari contoh kode program:

34.destructuring_1.html

```
1 let mahasiswa = ["Andi", "Lisa", "Eko"];
2 let [a, b, c] = mahasiswa;
3
4 console.log(a); // Andi
5 console.log(b); // Lisa
6 console.log(c); // Eko
```

Di baris 1 variabel `mahasiswa` saya input dengan array 3 element. Kemudian di baris 2 terdapat perintah *array destructuring* untuk memindahkan setiap element array `mahasiswa` ke dalam variabel `a`, `b` dan `c`.

Dengan perintah ini, string "Andi" akan tersimpan ke dalam variabel `a`, string "Lisa" ke variabel `b`, dan string "Eko" ke variabel `c`.

Perintah *destructuring* juga bisa dipakai untuk object:

35.destructuring_2.html

```
1 let mahasiswa = {
2   nama: "Eka",
3   umur: 19,
4   jurusan: "Teknik Informatika",
5 };
6
7 let {nama, umur} = mahasiswa;
8
9 console.log(nama); // Eka
10 console.log(umur); // 19
```

Untuk object, nama variabel yang dipakai saat proses *destructuring* harus sama dengan nama property / method yang ingin diambil. Sedangkan untuk array, posisi element-lah yang menentukan.

Dalam contoh di atas, perintah di baris 7 akan mengisi variabel `nama` dengan nilai property `nama` milik object `mahasiswa`. Begitu juga untuk variabel `umur`.

Sedikit perbedaan ketika men-*destructuring* array dan object ada di penggunaan tanda kurung. Untuk *array destructuring*, tanda yang dipakai adalah kurung siku "`[] = ...`", sedangkan untuk *object destructuring*, menggunakan tanda kurung kurawal "`{ } = ...`".

Di dalam React nanti, *destructuring* ini banyak kita pakai ketika memproses hasil pemanggilan function. Berikut ilustrasinya:

36.destructuring_3.html

```
1 const dataMahasiswa = () => {
2   return {nama: "Eka", umur: 19, jurusan: "Teknik Informatika"}
3 }
4
5 let {nama, umur} = dataMahasiswa();
6
7 console.log(nama); // Eka
8 console.log(umur); // 19
```

Di baris 1 - 3 terdapat deklarasi function `dataMahasiswa()` yang mengembalikan hasil dalam bentuk object. Agar lebih praktis, saat menjalankan fungsi tersebut di baris 5 langsung saya input ke dalam object destructuring. Hasilnya, variabel `nama` dan `umur` akan berisi nilai dari property `nama` dan property `umur`.

Object destructuring juga mengizinkan kita membuat alias atau nama pengganti dari property yang ingin diakses:

37.destructuring_4.html

```
1 const dataMahasiswa = () => {
2   return {nama: "Eka", umur: 19, jurusan: "Teknik Informatika"}
3 }
4
5 let {nama: namaMahasiswa, umur: umurMahasiswa} = dataMahasiswa();
6
7 console.log(namaMahasiswa); // Eka
8 console.log(umurMahasiswa); // 19
```

Di baris 5 saya membuat alias dari property `nama` menjadi `namaMahasiswa`, serta `umur` menjadi `umurMahasiswa`. Ini sangat berguna untuk menghindari bentrok nama variabel karena mungkin saja kita sudah memiliki variabel `nama` dan `umur` di tempat lain.

2.11. Spread Operator

ES6 menghadirkan operator baru dengan karakter tanda titik tiga kali: " ... ". Uniknya, operator ini mewakili 2 fitur, yakni **spread operator** dan **rest parameter**. Perbedaan dari keduanya terletak di kapan tanda " ..." digunakan.

Kita akan bahas *spread operator* terlebih dahulu. Dalam bahasa inggris, kata "spread" berarti sebaran atau menyebarluaskan. Sesuai dengan makna ini, *spread operator* dipakai untuk "menyebarluaskan" nilai tipe data yang memiliki banyak element seperti array atau object.

Berikut contoh dari penggunaan *spread operator* untuk tipe data array:

38.spread_operator_array.html

```
1 let arr1 = [10, 20, 30];
2 let arr2 = [100, 200, 300];
```

```
3
4 let arr3 = [...arr1,...arr2]
5 console.log(arr3); // [10, 20, 30, 100, 200, 300]
6
7 let arr4 = [...arr1, 40, 50]
8 console.log(arr4); // [10, 20, 30, 40, 50]
9
10 let arr5 = [30, 40, 50, ...arr1]
11 console.log(arr5); // [30, 40, 50, 10, 20, 30]
```

Di baris 1 dan 2 saya mengisi variabel `arr1` dan `arr2` dengan array yang terdiri dari beberapa element.

Di baris 4, itu adalah cara penggunaan *spread operator* untuk menggabungkan semua element `arr1` dan `arr2` menjadi array baru. Kita menulis tanda titik tiga sebelum nama variabel seperti `...arr1`.

Di sini, *spread operator* berfungsi untuk "membuka" atau "menyebarluaskan (spread)" isi array agar bisa diproses seolah-oleh setiap element ditulis secara terpisah. Sebagai perbandingan, jika `arr1` dan `arr2` digabung tanpa *spread operator*, `arr3` akan berisi array 2 dimensi:

```
let arr3 = [arr1, arr2]
console.log(arr3); // [[10, 20, 30], [100, 200, 300]]
```

Tidak hanya dari variabel, ketika mengisi `arr4` dan `arr5` di baris 7 dan 10 saya menggabung `...arr1` dengan element baru yang ditulis langsung. Hasilnya, `arr4` dan `arr5` berisi semua element dari `arr1` serta element tambahan lain. Posisi penulisan *spread operator* ikut menentukan urutan element.

Penggunaan *spread operator* ini juga berfungsi untuk men-copy nilai sebuah array ke array lain.

Kita juga bisa menerapkan *spread operator* untuk object:

39.spread_operator_object.html

```
1 let mahasiswa = {
2   nama: "Eka",
3   umur: 19,
4   jurusan: "Teknik Informatika"
5 };
6
7 let mahasiswa1 = {
8   ...mahasiswa,
9   umur: 20,
10  tempatLahir: "Jakarta"
11 }
12
13 console.log(mahasiswa1);
14 // {nama: "Eka", umur: 20, jurusan: "Teknik Informatika",
15 // tempatLahir: "Jakarta"}
```

Di baris 8 terdapat perintah `...mahasiswa`. Perintah ini akan "menyebarluaskan" seluruh property dan juga method (jika ada) milik object `mahasiswa`. Hasilnya, object `mahasiswa1` akan berisi semua nilai yang ada di object `mahasiswa` plus tambahan property `tempatLahir`.

Jika pada saat penggabungan terdapat property yang bernama sama, nilai baru akan menimpa nilai lama. Karena itulah property `umur` yang sebelumnya berisi angka 19 di object `mahasiswa`, ditimpak menjadi angka 20 di object `mahasiswa1`.

Sedikit tambahan, nilai baru ini juga bisa berasal dari variabel:

39a.spread_operator_addition_1.html

```
1 let mahasiswa = {  
2   nama: "Eka",  
3   umur: 19,  
4   jurusan: "Teknik Informatika"  
5 };  
6  
7 let umur = 20;  
8 let tempatLahir = "Jakarta";  
9  
10 let mahasiswa1 = {  
11   ...mahasiswa,  
12   umur: umur,  
13   tempatLahir: tempatLahir  
14 }  
15  
16 console.log(mahasiswa1);  
17 // {nama: "Eka", umur: 20, jurusan: "Teknik Informatika",  
18 // tempatLahir: "Jakarta"}
```

Di baris 7 dan 8 saya membuat variabel `umur` dan `tempatLahir` yang kemudian dipakai sebagai nilai property di baris 12 dan 13.

Alternatif penulisan yang lebih singkat adalah dengan langsung menulis nama variabel saja:

39b.spread_operator_addition_2.html

```
1 ...  
2 let umur = 20;  
3 let tempatLahir = "Jakarta";  
4  
5 let mahasiswa1 = {  
6   ...mahasiswa,  
7   umur,  
8   tempatLahir  
9 }  
10 ...
```

Di baris 7 dan 8 saya tidak perlu menulis pasangan nama property dan nilainya, tapi cukup nama variabel saja. Secara otomatis JavaScript akan menjadikan nama variabel sebagai nama property.

2.12. Rest Parameter

Rest parameter memakai tanda titik tiga yang sama seperti *spread operator*, namun kali ini fungsinya berkebalikan. Jika sebelumnya spread operator bertugas "menyebarluaskan" element array atau object, maka *rest parameter* akan "mengumpulkan" element tersebut.

Berikut contoh praktik *rest parameter* untuk mengumpulkan parameter function:

40.rest_parameter_1.html

```

1  function foo(...args) {
2    return args;
3  }
4
5  console.log( foo(1, 2, 3) )           // [1, 2, 3]
6  console.log( foo(3, 5, 7, 9) )         // [3, 5, 7, 9]
7  console.log( foo(5, 10, 15, 20, 25, 30) ) // [5, 10, 15, 20, 25, 30]
```

Pada saat deklarasi function `foo()`, saya menulis parameter sebagai `...args`. Di sini variabel `args` akan mengumpulkan semua argument yang dikirim pada saat pemanggilan fungsi. Berapa pun jumlah argument di baris 5 -7, semuanya dikumpulkan ke dalam array `args`.

Sesampainya di dalam function, kita bisa memproses `args` seperti array pada umumnya:

41.rest_parameter_2.html

```

1  function tambah(...args) {
2    let total = 0;
3    for (let angka of args) {
4      total += angka;
5    }
6    return total;
7  }
8
9  console.log( tambah(1, 2, 3) )           // 6
10 console.log( tambah(3, 5, 7, 9) )          // 24
11 console.log( tambah(5, 10, 15, 20, 25, 30) ) // 105
```

Kali ini terdapat perulangan `for of` di dalam function `tambah()` yang di pakai untuk menghitung hasil penjumlahan semua element `args`.

Meskipun menggunakan rest parameter, kita tetap bisa menulis parameter normal ke dalam deklarasi function. Syaratnya, rest parameter harus ditempatkan paling akhir:

42.rest_parameter_3.html

```

1  const tambah = (a, b, ...args) => {
2    let total = 0;
3    for (const angka of args) {
4      total += angka;
5    }
6    return total;
```

```
7  }
8
9  console.log( tambah(3, 5, 7, 9) ) // 16
```

Sekarang deklarasi function `tambah()` saya tulis memakai format *arrow notation*. Hasil akhir pemanggilan fungsi `tambah(3, 5, 7, 9)` adalah 16, bukan 24. Ini karena angka 3 dan 5 sudah ditampung ke dalam parameter `a` dan `b` sehingga tidak ikut menjadi element array `args`.

Nama array *rest parameter* tidak harus `args`, tapi bisa diganti dengan nama lain sebagaimana layaknya variabel biasa.

Fungsi lain dari *rest parameter* adalah memilih sebagian nilai saat melakukan array/object destructuring:

43.rest_parameter_descctructuring.html

```
1  const arr1 = [10, 20, 30, 40, 50];
2  [a, b, ...sisa] = arr1;
3
4  console.log(a); // 10
5  console.log(b); // 20
6  console.log(sisa); // [30, 40, 50]
```

Dalam contoh ini variabel `sisa` akan berisi semua element `arr1` mulai dari element ke-3 dan seterusnya. Sebab, dua element pertama sudah disimpan ke dalam variabel `a` dan `b`.

2.13. Array Method: `forEach()` dan `map()`

Array menjadi salah satu tipe data yang sangat sering kita pakai di React. Oleh karena itu memahami cara pemrosesan array menjadi topik yang penting, terutama cara mengakses setiap element array.

Salah satunya adalah dengan perulangan `for`, dimana kita bisa mengakses satu per satu element array dalam setiap iterasi. Cara ini tidak salah, namun kodennya bisa lebih singkat dengan memanfaatkan method bawaan JavaScript.

JavaScript menyediakan cukup banyak method untuk memproses array. Di buku **JavaScript Uncover** saya menyediakan satu bab khusus terkait materi ini (belasan method). Sekedar penyegaran, kali ini kita akan bahas 2 diantaranya: `Array.forEach()` dan `Array.map()`.

Method `Array.forEach()` sangat praktis untuk mengakses setiap element array. Method ini butuh satu argument berbentuk function:

44.array.foreach.html

```
1  let arr = [10, 20, 30, 40];
2
3  arr.forEach( function(val, key) {
```

```
4   console.log(`Nilai array di index-${key} adalah ${val}`);
5 );
6
7 // Nilai array di index-0 adalah 10
8 // Nilai array di index-1 adalah 20
9 // Nilai array di index-2 adalah 30
10 // Nilai array di index-3 adalah 40
```

Function yang diletakkan sebagai argument dikenal juga dengan istilah **callback**. Callback untuk method `forEach()` sebenarnya bisa menerima 3 argument:

- Argumen pertama berupa nilai element yang sedang di proses (*value array*).
- Argumen kedua berupa index element yang sedang di proses (*key array*).
- Argumen ketiga berisi seluruh array asal.

Dalam contoh di atas saya hanya memakai 2 argument saja, yakni `val` dan `key` (baris 3). Di dalam callback, keduanya bisa diakses untuk menampilkan `key` serta `value` dari element yang saat ini sedang di proses.

Supaya lebih singkat, callback juga bisa ditulis dalam bentuk *arrow notation*:

45.array_FOREACH_arrow.html

```
1 let arr = [10,20,30,40];
2
3 arr.forEach(
4   (val, key) => console.log(`Nilai array di index-${key} adalah ${val}`)
5 );
```

Hasil akhir dari kode ini sama seperti contoh sebelumnya.

Salah satu pemrosesan array yang sering dilakukan di React adalah men-generate array baru berdasarkan array yang sudah ada. Sebagai contoh, dalam kode berikut saya ingin membuat array `arrKuadrat` yang berasal dari nilai kuadrat setiap element array `arr`:

46.array_FOREACH_kuadrat.html

```
1 let arr = [10,20,30,40];
2 arrKuadrat = [];
3
4 arr.forEach( (val) => arrKuadrat.push(val*val) );
5
6 console.log(arrKuadrat); // [100, 400, 900, 1600]
```

Di dalam *callback*, perintah `arrKuadrat.push(val*val)` akan menambah satu per satu hasil `val*val` ke dalam array `arrKuadrat`.

Untuk keperluan ini, JavaScript sebenarnya sudah menyediakan method yang lebih praktis, yakni **Array.map()**. Cara penggunaan method `map()` mirip seperti `forEach()`, yakni perulangan untuk setiap element array, namun dalam setiap iterasi method `map()` akan langsung

mengembalikan nilai baru.

Berikut cara mencari kuadrat dengan memakai method `Array.map()`:

47.array_map.html

```
1 let arr = [10, 20, 30, 40];
2 let arrKuadrat = arr.map( (val) => val*val );
3
4 console.log(arrKuadrat); // [100, 400, 900, 1600]
```

Sekarang kita tidak perlu lagi method `push()` di dalam callback, cukup tampung hasil pemanggilan `arr.map()` ke dalam array `arrKuadrat`.

Fungsi `callback` untuk `Array.map()` juga bisa menerima 3 buah argument yang sama seperti di method `Array.forEach()`, yakni value, key dan seluruh array. Kebetulan dalam contoh di atas saya hanya butuh value saja.

Dalam praktek di React nanti, array asal biasanya akan berisi kumpulan object, lalu kita melakukan perulangan untuk men-generate struktur HTML berdasarkan array tersebut. Berikut simulasi dari kasus ini:

48.array_map_object.html

```
1 const mahasiswa = [
2   {
3     nama: "Eka",
4     umur: 19,
5     jurusan: "Teknik Informatika"
6   },
7   {
8     nama: "Lisa",
9     umur: 18,
10    jurusan: "Sistem Informasi"
11  },
12  {
13    nama: "Rudi",
14    umur: 19,
15    jurusan: "Teknik Elektro"
16  }
17];
18
19 const prosesMahasiswa = (mahasiswa) =>
20 `<p>${mahasiswa.nama} (${mahasiswa.umur}), ${mahasiswa.jurusan}</p>`;
21
22 const formatMahasiswa = mahasiswa.map(prosesMahasiswa);
23
24 console.log(formatMahasiswa);
25
26 // [
27 //   "<p>Eka (19), Teknik Informatika</p>",
28 //   "<p>Lisa (18), Sistem Informasi</p>",
29 //   "<p>Rudi (19), Teknik Elektro</p>"
```

```
30 // ]
```

Konstanta `mahasiswa` di baris 1 – 17 berisi array dengan 3 buah element berbentuk object. Setiap object memiliki property `nama`, `umur` dan `jurusan`.

Di baris 19 terdapat deklarasi function `prosesMahasiswa()` yang saya siapkan sebagai `callback` untuk perulangan `map()`. Sebenarnya callback ini bisa saja diitulis langsung ke dalam method `mahasiswa.map()` seperti contoh-contoh sebelumnya, pemisahan ini sekedar agar lebih rapi saja.

Di baris 22, konstanta `formatMahasiswa` akan berisi array baru hasil dari pemrosesan function `mahasiswa.map(prosesMahasiswa)`. Untuk setiap object di array `mahasiswa`, akan di format dengan perintah: `<p>${mahasiswa.nama} (${mahasiswa.umur}), ${mahasiswa.jurusan}</p>`. Isi konstanta `formatMahasiswa` kemudian ditampilkan dengan perintah `console.log()` di baris 24.

Silahkan coba pahami sejenak cara kerja dari kode di atas, karena sangat sering kita pakai di React nantinya.

Nama konstanta `mahasiswa` dan `formatMahasiswa` bukanlah salah ketik. Saya sengaja menambah satu huruf 's' di akhir kata sebagai penanda bahwa konstanta atau variabel tersebut berisi data yang banyak (plural).

Ini berasal dari aturan kosa kata bahasa inggris dimana kata jamak selalu diakhiri dengan huruf 's'. Misal, `student` adalah tunggal, tapi `students` adalah jamak (banyak `student`), atau `item` adalah tunggal, tapi `items` adalah jamak (banyak `item`).

Salah satu masalah klasik yang selalu dihadapi setiap programmer adalah bagaimana memilih nama variabel. Jadi daripada ribet, saya tambah saja huruf 's' sebagai penanda sedang memproses data array.

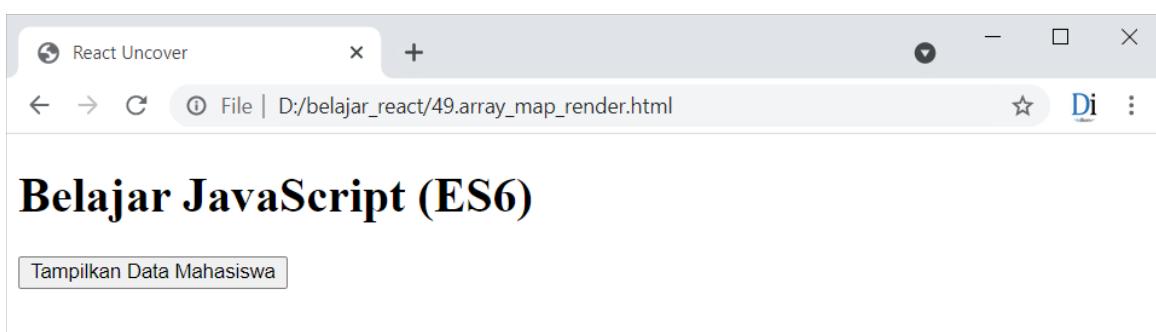
Alternatif lain yang lebih banyak dipakai adalah menggunakan kata bahasa inggris untuk semua variabel, sehingga kita tidak merasa janggal dengan tambahan huruf 's' di akhir kata. Tapi tetap saja aturan ini lebih ke pilihan, bukan hal wajib.

Agar contoh pemrosesan data sebelumnya menjadi lebih "real" lagi, saya akan input array `formatMahasiswa` ke dalam **DOM** (Document Object Model) :

49.array_map_render.html

```
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>React Uncover</title>
7 </head>
```

```
8 <body>
9   <h1>Belajar JavaScript (ES6)</h1>
10  <div id="root"></div>
11  <button onclick="renderData()">Tampilkan Data Mahasiswa</button>
12  <script>
13    const mahasiswas = [
14      {
15        nama: "Eka",
16        umur: 19,
17        jurusan: "Teknik Informatika"
18      },
19      {
20        nama: "Lisa",
21        umur: 18,
22        jurusan: "Sistem Informasi"
23      },
24      {
25        nama: "Rudi",
26        umur: 19,
27        jurusan: "Teknik Elektro"
28      }
29    ];
30
31    const prosesMahasiswa = (mahasiswa) =>
32      `<p>${mahasiswa.nama} (${mahasiswa.umur} tahun) -
33      <i>${mahasiswa.jurusan}</i></p>`;
34
35    const formatMahasiswas = mahasiswas.map(prosesMahasiswa);
36
37    const renderData = () => {
38      document.getElementById('root').innerHTML = formatMahasiswas.join("");
39    }
40  </script>
41 </body>
42 </html>
```



Gambar: Menampilkan array hasil method map()

Setelah judul tag `<h1>` di baris 9, terdapat tag `<div id="root">` di baris 10. Tag ini tidak memiliki element apapun dan akan kita isi melalui JavaScript.

Kemudian di baris 11 terdapat tag `<button>` dengan atribut `onclick="renderData()"`. Artinya pada saat tombol ini di klik, function `renderData()` akan dijalankan.

Kode dari function `renderData()` sendiri ada di baris 36 – 38. Isinya berupa perintah untuk mengisi property `innerHTML` milik `node element 'root'` dengan array `formatMahasiswa` yang sudah digabung menjadi string panjang dengan method `join("")`.

Hasilnya pada saat tombol "Tampilkan Data Mahasiswa" di klik, ketiga object mahasiswa akan tampil ke dalam tag `<div id="root">`:



Gambar: Hasil array saat tombol di klik

Teknik seperti inilah yang akan banyak kita pakai di React. Data array `mahasiswa` bisa saja berasal dari API yang kemudian diolah dan ditampilkan menggunakan React.

Sebenarnya masih banyak method JavaScript lain untuk pemrosesan data array. Namun agar tidak terlalu panjang, saya cukupkan dengan method `forEach()` dan `map()` saja. Penjelasan terkait method lain seperti `join()`, `reduce()`, `filter()`, `push()` dan `pop()` bisa dibaca di buku **JavaScript Uncover**.

2.14. Asynchronous JavaScript (Promise / Async - Await)

Ketika memproses kode program, normalnya JavaScript berjalan secara **synchronous**, yakni berurutan dari perintah baris pertama, kedua, dst hingga selesai. Beberapa perintah bisa dijalankan ulang saat terjadi sesuatu seperti ketika tombol di klik (konsep *event*), namun tetap saja kode itu diproses secara berurutan.

Berikut contoh sederhana dari synchronous JavaScript:

```
50.synchronous_js.html
1 const getUser = () => "Rudi";
2
3 console.log("Start..."); // Start...
4 console.log(getUser()); // Rudi
5 console.log("Finish") // Finish
```

Hasilnya, string "Rudi" tampil di urutan kedua karena `console.log(getUser())` dijalankan di

antara `console.log("Start...")` dan `console.log("Finish")`.

Pada situasi tertentu, kadang kita butuh agar JavaScript berjalan secara **asynchronous**, yakni boleh tidak berurutan dan bisa melompati kode-kode tertentu. Ini diperlukan karena ada data yang belum tersedia saat kode di eksekusi.

Contohnya ketika ada perintah yang harus menunggu data external dari server back-end. Jika server sedang sibuk, data tersebut bisa saja sampai 1 atau 2 detik setelahnya, sehingga sudah terlambat untuk di proses. Berikut simulasi dari masalah ini:

51.asynchronous_js_problem_1.html

```
1 const getUser = () => {
2   setTimeout(() => "Rudi", 2000)
3 }
4
5 console.log("Start...");
6 console.log(getUser());
7 console.log("Finish")
```

Tanpa menjalankan kode ini, bisakah anda tebak apa hasil dari perintah `console.log` (`getUser()`) di baris 6?

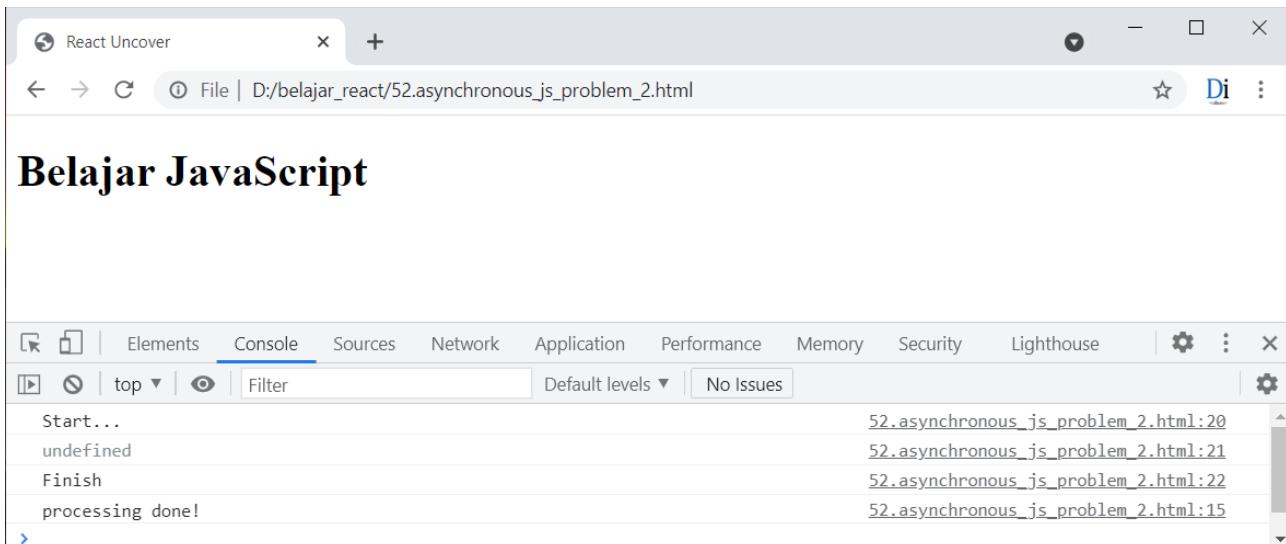
Hasilnya: `undefined`. Ini terjadi karena di dalam deklarasi `getUser()`, fungsi `setTimeout()` baru mengembalikan string "Rudi" setelah 2 detik atau 2000 milidetik kemudian. Ini sudah terlambat karena semua perintah JavaScript langsung dijalankan saat halaman di-load.

Contoh berikut bisa memperjelas apa yang terjadi:

52.asynchronous_js_problem_2.html

```
1 const getUser = () => {
2   setTimeout(() => {
3     console.log("processing done!");
4     return "Rudi";
5   }, 2000)
6 }
7
8 console.log("Start...");
9 console.log(getUser());
10 console.log("Finish")
```

Perbedaan dari kode sebelumnya ada di baris 3, dimana saya menambah satu perintah `console.log()` ke dalam fungsi `setTimeout()`. Berikut hasil yang didapat:



Gambar: Menjalankan asynchronous JavaScript

Teks "processing done!" tampil di baris 4 tepat setelah 2 detik halaman di proses. Ini membuktikan kalau fungsi `getUser()` sudah berjalan tapi hasilnya baru datang 2 detik kemudian.

Sebagai solusi, terdapat 3 cara untuk memproses perintah asynchronous di JavaScript, yakni salah satu dari **callback**, **promise**, atau **sync / await**.

Agar bahasan kita tidak terlalu panjang, saya akan lompati callback karena kurang praktis dan juga jarang dipakai. Kita akan fokus ke promise dan sync / await saja.

Memahami cara kerja **promise** memang lumayan rumit. Jika agak bingung, coba pelajari secara perlahan atau boleh dilompati untuk sementara dan kembali lagi ketika sudah sampai ke bab tentang pemrosesan [Fetch API](#) di akhir buku nanti.

JavaScript Promise

Promise adalah sebuah object yang menjanjikan suatu hasil di "masa depan". Ini sesuai dengan sifat perintah **asynchronous** yang hasilnya tidak kita dapat saat itu juga (butuh jeda beberapa saat). Mari bahas dengan contoh kode program berikut:

53.promise_resolve.html

```
1 const getUser = () => {
2   return new Promise((resolve) => {
3     setTimeout(() => {
4       resolve("Rudi");
5     }, 2000)
6   })
7 }
8
9 console.log("Start...");
10 getUser().then((userName) => console.log(userName));
11 console.log("Finish");
```

Kali ini fungsi `getUser()` saya modifikasi agar mengembalikan **promise object** dengan perintah `new Promise()` di baris 2. Proses instansiasi promise object butuh argument berbentuk function yang saya tulis dalam *arrow notation* antara baris 2 – 6.

Function untuk promise ini bisa menerima 2 buah argument, yakni **resolve** dan **reject**. Keduanya berbentuk *callback* yang akan dijalankan saat promise berhasil (untuk *resolve*) dan saat promise gagal (untuk *reject*).

Dalam contoh di atas saya hanya memakai satu argument saja, yakni *resolve* di akhir baris 2. Fungsi *resolve()* dijalankan dengan perintah `resolve("Rudi")` pada baris 4. Ini artinya promise object berjanji akan mengembalikan string "Rudi" sebagai nilai akhir.

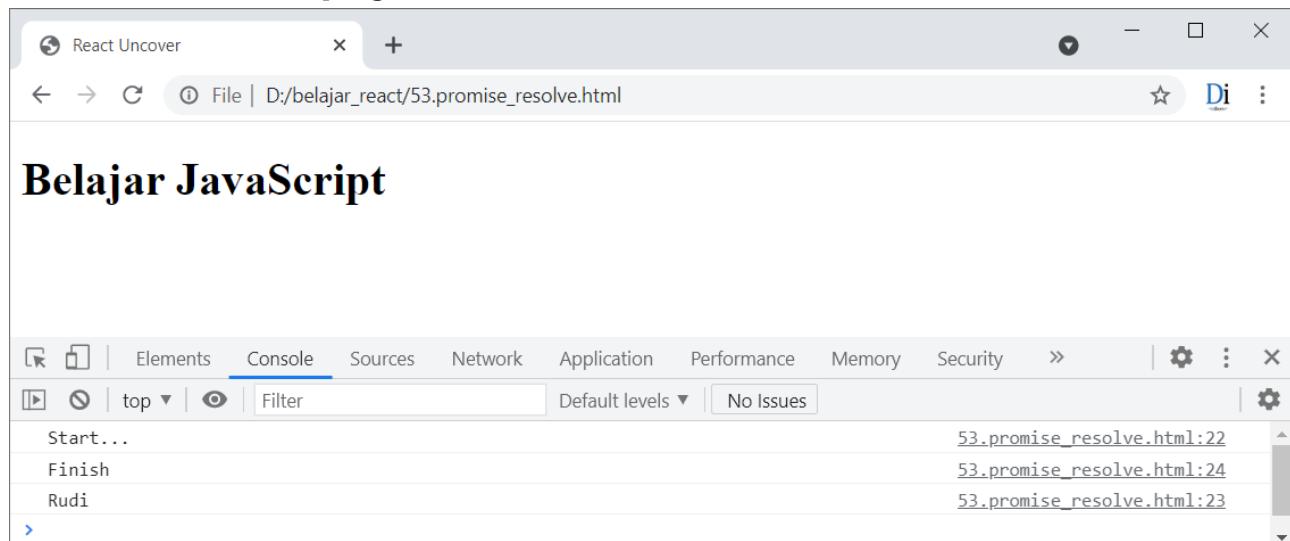
Perlu diperhatikan bahwa karena *resolve()* dipanggil dari dalam fungsi `setTimeout()`, maka string "Rudi" baru akan didapat 2 detik setelah kode berjalan.

Untuk bisa menangkap hasil promise, kita tidak bisa memanggil langsung method `getUser()`, tapi perlu tambahan method `then()` seperti di baris 10:

```
getUser().then((userName) => console.log(userName));
```

Method `then()` perlu argument dalam bentuk function, dimana argument pertama function otomatis akan diisi hasil dari callback *resolve()* milik promise. Pada saat dijalankan, argument `userName` nantinya akan berisi string "Rudi" untuk kemudian masuk ke perintah `console.log()`.

Berikut hasil dari kode program di atas:



Gambar: Menjalankan asynchronous JavaScript dengan promise

Saya sangat sarankan anda untuk jalankan langsung kode ini agar bisa melihat bahwa string "Rudi" baru muncul setelah 2 detik. String "Rudi" juga tampil setelah "Finish", padahal perintah `console.log("Finish")` ada di baris terakhir setelah pemanggilan fungsi `getUser()`.

Inilah solusi untuk menjalankan asynchronous JavaScript. Dengan memakai promise, perintah tersebut akan ditunda dan baru diproses ketika datanya sudah tersedia.

Selanjutnya saya akan modifikasi fungsi `getUser()` dengan menambah callback `reject` ke dalam promise:

54.promise_reject.html

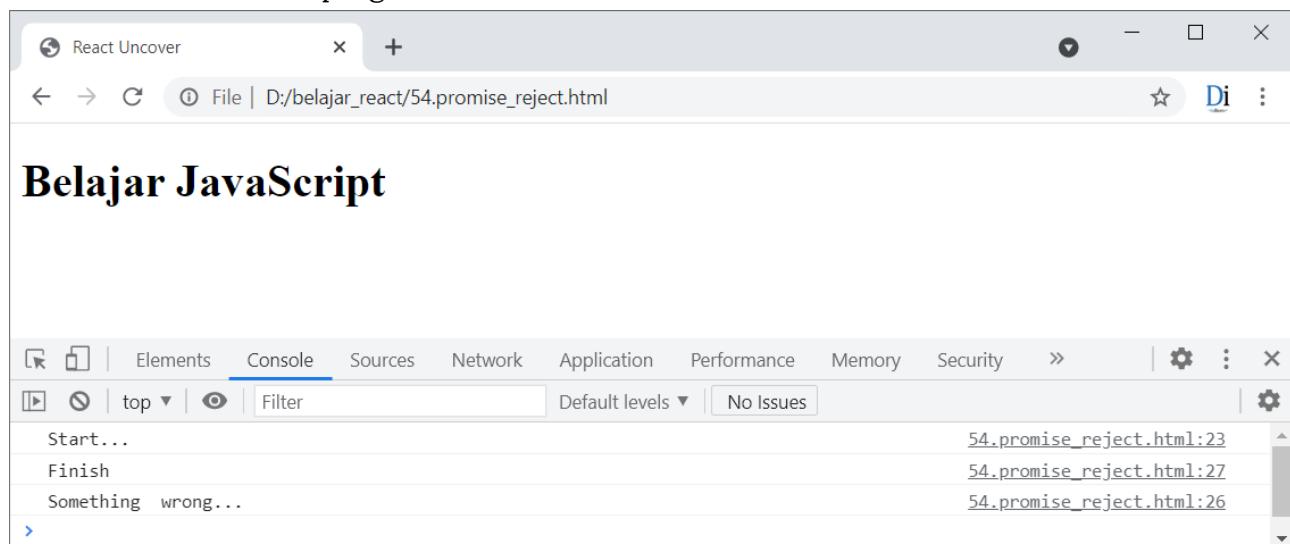
```
1 const getUser = () => {
2   return new Promise((resolve, reject) => {
3     setTimeout(() => {
4       //resolve("Rudi");
5       reject("Something wrong...");
6     }, 2000)
7   })
8 }
9
10 console.log("Start...");
11 getUser()
12 .then((userName) => console.log(userName))
13 .catch((error) => console.log(error));
14 console.log("Finish");
```

Seperti yang sempat di singgung, promise object bisa menerima 2 buah argument, yakni `resolve` yang diisi nilai saat promise berhasil serta `reject` yang akan berjalan jika promise gagal. Kedua callback ini saya tulis sebagai argument di akhir baris 2.

Agar kita bisa melihat hasil `reject`, saya menonaktifkan `resolve("Rudi")` di baris 4 dan menambah perintah `reject("Something wrong...")` di baris 5. Ini membuat promise selalu gagal karena tidak menemukan `resolve`.

Untuk "menangkap" hasil `reject()`, sambung pemanggilan method `then()` dengan `catch()` seperti di baris 13. Argument `error` di dalamnya otomatis bersisi hasil dari fungsi `reject()` yang ada di dalam promise.

Berikut hasil dari kode program di atas:



Gambar: Menampilkan hasil saat promise gagal

Kali ini setelah 2 detik akan tampil string "Something wrong..." karena promise gagal menjalankan method `resolve()`.

Async - Await

Async dan **await** bisa dipakai sebagai alternatif untuk menjalankan promise. Dalam banyak hal, penulisan dengan `async - await` menjadi lebih sederhana karena kita tidak perlu menggunakan `then()` dan `catch()` untuk memproses promise.

Berikut contoh penggunaan `async - await`:

55.await_resolve.html

```

1  const getUser = () => {
2    return new Promise((resolve, reject) => {
3      setTimeout(() => {
4        resolve("Rudi");
5        reject("Something wrong...");
6      }, 2000)
7    })
8  }
9
10 const tryGetName = async () => {
11   let userName = await getUser();
12   console.log(userName);
13 };
14
15 console.log("Start...");
16 tryGetName();
17 console.log("Finish")

```

Isi deklarasi fungsi `getUser()` tetap berisi promise, yakni mengembalikan string "Rudi" jika berhasil (`resolve`), dan mengembalikan string "Something wrong..." jika gagal (`reject`).

Yang berbeda ada di tambahan fungsi `tryGetName()` pada baris 10 – 13. Tambahan keyword `async` sebelum tanda kurung argument dipakai untuk memberitahu JavaScript bahwa fungsi ini akan berjalan secara *asynchronous*.

Perintah *asynchronous* yang dimaksud ada di baris 11, dimana variabel `userName` akan menampung hasil dari fungsi `getUser()`. Kali ini sebelum penulisan fungsi `getUser()` terdapat keyword `await` sebagai tanda kalau kita ingin menunggu hasilnya.

Hasil yang didapat akan sama seperti menggunakan `then()`:

```

Start...
Finish
Rudi (baru tampil setelah 2 detik)

```

Untuk menangkap hasil `reject`, tempatkan perintah `await` ke dalam block `try-catch`:

56.await_reject.html

```
1 const getUser = () => {
2   return new Promise((resolve, reject) => {
3     setTimeout(() => {
4       // resolve("Rudi");
5       reject("Something wrong...");
6     }, 2000)
7   })
8 }
9
10 const tryGetName = async () => {
11   try {
12     let userName = await getUser();
13     console.log(userName);
14   }
15   catch (error) {
16     console.log(error);
17   }
18 };
19
20 console.log("Start...");
21 tryGetName();
22 console.log("Finish")
```

Hasil kode program:

```
Start...
Finish
Something wrong... (baru tampil setelah 2 detik)
```

Karena baris `resolve("Rudi")` sudah tidak aktif, maka promise di dalam fungsi `getUser()` akan mengeksekusi baris `reject("Something wrong...")`. Hasil ini ditampung oleh variabel `error` di dalam blok `catch`.

Catatan tambahan, perintah `await` hanya bisa ditulis di dalam function yang sudah di set sebagai `async`. Sehingga kedua perintah ini hampir selalu berpasangan. Bagi beberapa programmer, memproses promise menggunakan `async-await` terasa lebih rapi dibandingkan method `then().catch()`, namun ini lebih ke kesukaan saja.

Dalam praktek di React, `asynchronous JavaScript` akan banyak terpakai saat memproses Fetch API, yakni mengambil dan menyimpan data ke server back-end. Materi ini akan kita bahas dalam 1 bab khusus nantinya.

2.15. JavaScript Module

JavaScript module adalah fitur untuk memecah kode program JavaScript menjadi beberapa file terpisah. File yang dipecah inilah yang disebut sebagai *modul*. Jika anda pernah belajar bahasa PHP, apa yang akan kita bahas ini mirip seperti fungsi `include()` atau `require()`.

Praktek JavaScript modul agak ribet karena terdapat mekanisme proteksi bawaan web browser. Web browser menolak mengakses file modul jika tidak berasal dari server yang sama (CORS policy). Dengan demikian, praktek JavaScript module harus dijalankan dari sebuah web server.

Salah satu web server yang biasa dipakai untuk proses belajar (terutama PHP), adalah **apache** web server bawaan **XAMPP**. Silahkan jalankan apache, lalu buat folder `js_module` di dalam folder `htdocs` (boleh juga menggunakan nama folder lain).

Jika anda tidak memiliki aplikasi XAMPP atau kurang paham cara menjalankan file dari XAMPP, itu juga tidak masalah. Materi JavaScript modul tidak terlalu rumit dan boleh dibaca tanpa di praktikkan langsung.

Pertama, saya akan buat sebuah file HTML dengan nama `main.html`:

`htdocs/js_module/main.html`

```
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>React Uncover</title>
7 </head>
8 <body>
9   <h1>Belajar JavaScript (ES6)</h1>
10  <script type="module" src="main.js"></script>
11 </body>
12 </html>
```

Ini adalah kode HTML biasa. Hanya saja di baris 10 terdapat tag `<script>` dengan atribut `type="module"` dan `src="main.js"`. Atribut `type="module"` menginstruksikan ke web browser bahwa kita ingin mengakses file JavaScript tersebut sebagai modul. Modul yang dimaksud adalah file `main.js` yang akan kita buat sesaat lagi.

Masih di folder `js_module`, silahkan buat file kedua dengan nama `main.js`:

`htdocs/js_module/main.js`

```
1 import {haloJakarta} from './haloJakarta.js';
2
3 console.log( haloJakarta() );
```

Perintah `import` di baris 1 berarti saya ingin meng-import atau mengakses suatu data bernama `haloJakarta` yang ada di file `haloJakarta.js`. File `haloJakarta.js` ini memang belum ada dan akan kita buat sebentar lagi. Perhatikan bahwa data yang di-import ditulis dalam tanda kurung kurawal " {} ", yang dalam contoh ini adalah `{haloJakarta}`.

Setelah di import, di baris 3 saya menjalankan perintah `console.log(haloJakarta())`. Ini berarti data `haloJakarta` haruslah berbentuk function, karena terdapat tambahan tanda kurung di akhir kata `haloJakarta()`.

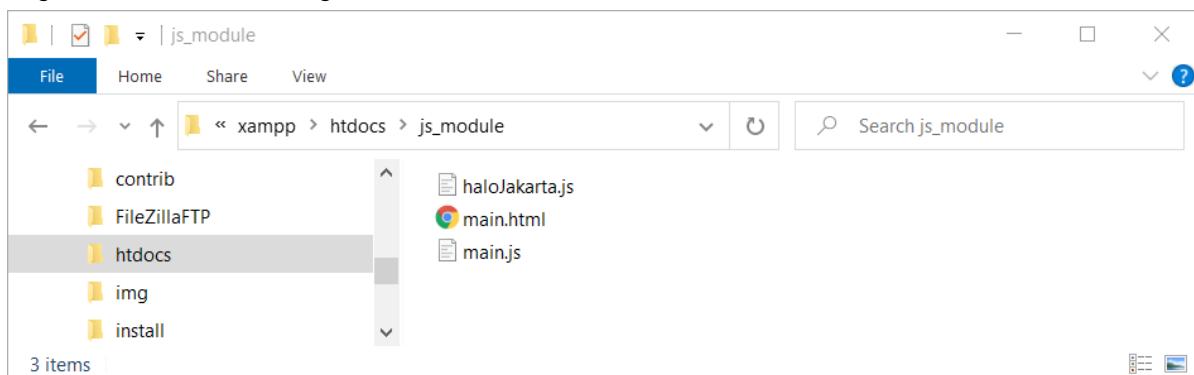
Sekarang mari buat file `haloJakarta.js` dengan kode sebagai berikut:

htdocs/js_module/haloJakarta.js

```
1 export const haloJakarta = () => {
2   return "Halo Jakarta";
3 }
```

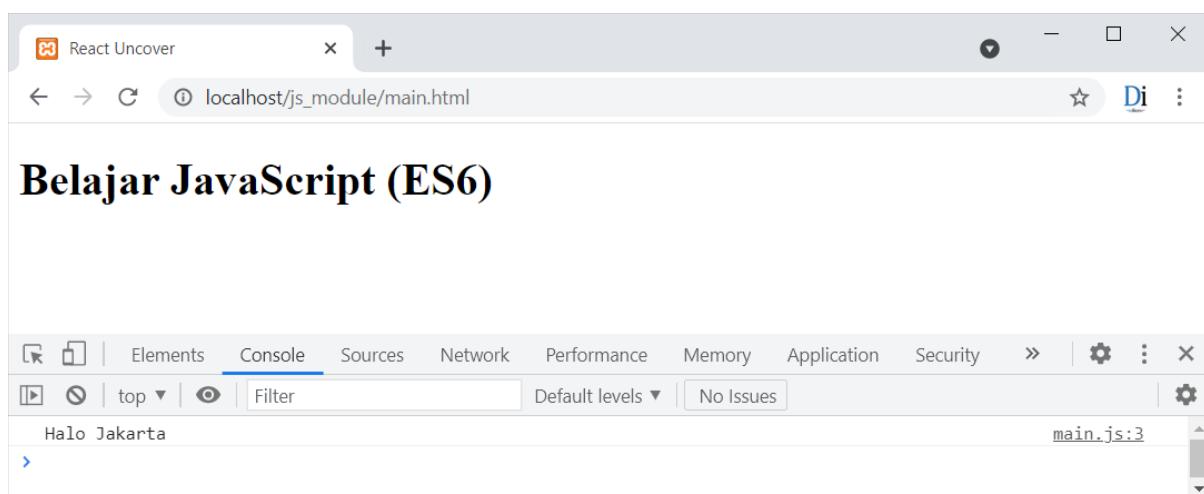
Di baris 1 terdapat perintah `export` yang langsung diikuti pendeklarasian function `halo Jakarta()`. Inilah cara meng-export sebuah function agar bisa diakses oleh perintah `import` dari file lain. Isi dari function `haloJakarta()` hanya mengembalikan string `"Halo Jakarta"`.

Baik, sampai di sini kita sudah memiliki 3 buah file di dalam folder `js_module`, yakni `main.html`, `main.js` dan `haloJakarta.js`.



Gambar: Isi folder js_module

Silahkan akses file `main.html` di web browser dengan alamat "`http://localhost/ js_module/main.html`". Kembali, JavaScript module hanya bisa berjalan jika halaman HTML di akses dari web server. Alamat URL harus `http://localhost/`, dan bukan `file:///`.



Gambar: Hasil tampilan file main.html

Jika tidak ada masalah, dalam tab Console akan tampil teks "Halo Jakarta". Teks ini berasal dari function `haloJakarta()` di file `haloJakarta.js` yang sudah kita import ke file `main.js`. Inilah konsep dasar fitur module di JavaScript.

Ketiga file di folder `js_module` saling terhubung satu sama lain. File `haloJakarta.js` berisi deklarasi fungsi `haloJakarta()`, lalu file `main.js` berisi perintah untuk menjalankan fungsi `haloJakarta()`, serta file `main.html` yang berisi kode HTML agar bisa diakses dari web browser.

Terdapat berbagai cara penulisan perintah export dan import. Misalnya di dalam file `haloJakarta.js` saya menulis perintah `export` langsung sebelum deklarasi konstanta, yakni:

```
1  export const haloJakarta = () => {
2    return "Halo Jakarta";
3 }
```

Alternatif cara penulisan lain adalah membuat perintah `export` terpisah:

```
1  const haloJakarta = () => {
2    return "Halo Jakarta";
3 }
4
5 export {haloJakarta};
```

Sekarang di baris 1 – 3 berisi pendeklarasian function saja, sedangkan perintah `export` diletakkan terpisah di baris 5.

Nama file modul tidak harus sama dengan nama function yang akan di `export`. Bisa saja kita membuat nama file `foo.js`, akan tetapi yang di `export` adalah function `bar()`.

Namun tentu lebih disarankan membuat nama file yang mencerminkan nama function.

Multiple Import / Export

JavaScript modul mengizinkan kita meng-export lebih dari satu function dalam satu file. Sebagai contoh, saya akan buat file `haloIndonesia.js` dengan kode sebagai berikut:

`htdocs\js_module\haloIndonesia.js`

```
1  const haloMedan = () => {
2    return "Halo Medan";
3 }
4
5  const haloSurabaya = () => {
6    return "Halo Surabaya";
7 }
8
9 export {haloMedan, haloSurabaya}
```

Kedua function sangat sederhana, hanya mengembalikan string ketika dipanggil. Di baris 9, terdapat perintah `export {haloMedan, haloSurabaya}` yang berarti saya ingin meng-export kedua function sekaligus.

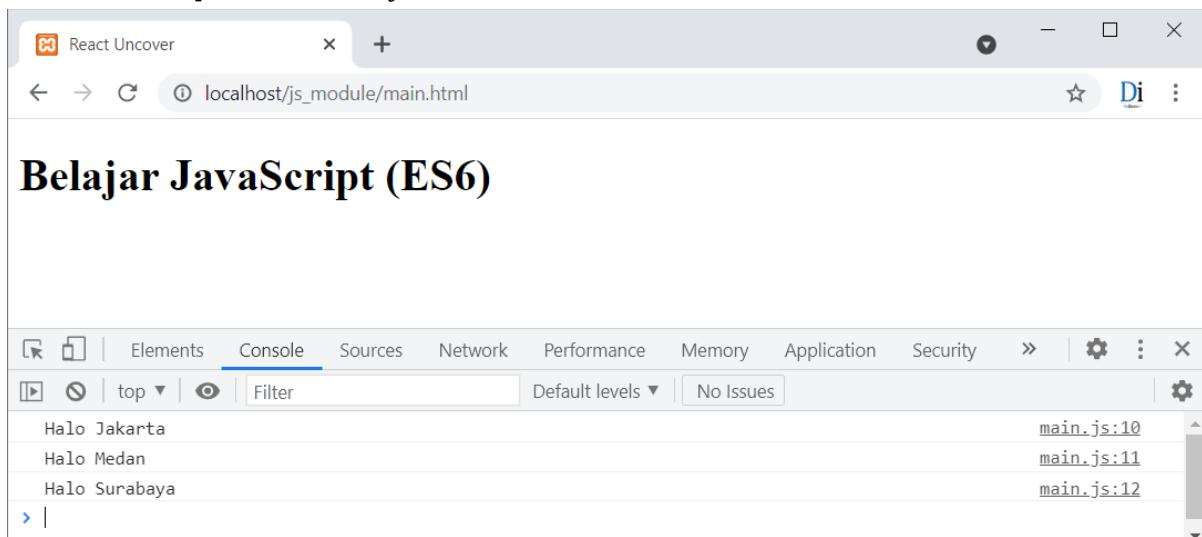
Balik ke file `main.js`, kita akan import file `haloIndonesia.js` dan menjalankannya:

htdocs/js_module/main.js

```
1 import {haloJakarta} from './haloJakarta.js';
2 import {haloMedan, haloSurabaya} from './haloIndonesia.js';
3
4 console.log( haloJakarta() );
5 console.log( haloMedan() );
6 console.log( haloSurabaya() );
```

Saya yakin anda sudah bisa memahami proses import ini, dimana kita perlu menuliskan nama function yang ingin di akses beserta nama file asal.

Berikut hasil tampilan saat menjalankan file `main.html`:



Gambar: Hasil tampilan file main.html

Export Default

Export default merupakan perintah tambahan untuk mempermudah proses export-import. Jika sebelumnya kita menulis `export {haloJakarta}` di file `haloJakarta.js`, sekarang mari ganti menjadi `export default haloJakarta`:

htdocs/js_module/haloJakarta.js

```
1 export const haloJakarta = () => {
2   return "Halo Jakarta";
3 }
4
5 export default haloJakarta;
```

Efek dari perintah `export default` di baris 5 adalah, ketika menjalankan perintah import di file `main.js`, kita bisa ganti nama function dengan nama lain, misalnya menjadi `hai`:

`htdocs/js_module/main.js`

```
1 import hai from './haloJakarta.js';
2 import {haloMedan, haloSurabaya} from './haloIndonesia.js';
3
4 console.log( hai() ); // Halo Jakarta
5 console.log( haloMedan() ); // Halo Medan
6 console.log( haloSurabaya() ); // Halo Surabaya
```

Perhatikan perintah di baris 1, saya tidak lagi menulis `import {haloJakarta} from...`, tapi `import hai from...`

Hasilnya, sepanjang kode program, `hai()` menjadi nama alias dari function `haloJakarta()`. Fitur ini bisa dipakai untuk menghindari bentrok nama function, atau jika nama function yang ingin di import terlalu panjang. Syarat lain, tanda kurung kurawal juga tidak boleh ditulis di dalam perintah import, yakni `import hai` dan bukan `import {hai}` (ini akan error).

Info tambahan, satu file modul hanya bisa memiliki 1 perintah `export default`, tidak bisa lebih. Maka jika kita ingin menambah perintah `export default` di file `haloIndonesia.js`, harus memilih satu function saja sebagai default, sedangkan function lain bisa dikirim dengan perintah `export` biasa (tanpa default).

Import Alias

Sebelumnya kita bisa membuat nama alias dari `export default`. Sebenarnya, `export` biasa juga bisa dibuat alias, namun dengan tambahan perintah `as`:

`htdocs/js_module/main.js`

```
1 import hai from './haloJakarta.js';
2 import {haloMedan as horas, haloSurabaya as halo} from './haloIndonesia.js';
3
4 console.log( hai() ); // Halo Jakarta
5 console.log( horas() ); // Halo Medan
6 console.log( halo() ); // Halo Surabaya
```

Di baris 2, saya membuat nama alias `haloMedan` menjadi `horas` dan `haloSurabaya` menjadi `halo`. Dengan demikian sepanjang kode program function `horas()` dan `halo()` lah yang akan dipakai.

Import All

Import all adalah cara penulisan singkat jika kita ingin mengimport semua function yang di export oleh suatu modul. Caranya, tulis karakter asterisk atau bintang `" * "` setelah perintah `import` serta diikuti nama variabel *namespace*:

htdocs/js_module/main.js

```
1 import hai from './haloJakarta.js';
2 import * as sapa from './haloIndonesia.js';
3
4 console.log( hai() ); // Halo Jakarta
5 console.log( sapa.haloMedan() ); // Halo Medan
6 console.log( sapa.haloSurabaya() ); // Halo Surabaya
```

Di baris, proses import dilakukan dengan perintah `import * as sapa`. Ini membuat sebuah namespace sapa untuk semua function yang di export dari `haloIndonesia.js`. Untuk mengakses setiap function, sekarang menjadi `sapa.haloMedan()` dan `sapa.haloSurabaya()`.

Bahasan tentang *module* ini mengakhiri kursus singkat kita tentang "JavaScript untuk React". Mayoritas materi akan tetap terpakai di kode JavaScript apapun, termasuk seandainya anda lanjut belajar JavaScript backend seperti NodeJS serta berbagai library JS lain.

Dengan bekal ini, kita siap masuk ke dunia React!

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

3. React dan React DOM

React mengadopsi konsep "*gradual adoption*", dimana kita bisa menggunakan react se-sedikit mungkin. Atau dengan kata lain kode program website tidak harus semuanya dibuat dengan react, cukup bagian yang diperlukan saja.

Misal, kita memiliki web yang sudah jadi berbasiskan HTML, CSS dan PHP. Kemudian ingin menambah efek interaktif di bagian tabel dengan react, maka itu tidak jadi masalah.

File react yang diperlukan hanya ada 2, yakni `react.js` dan `react-dom.js`. Dengan mengakses keduanya dari tag `<script>`, sebuah halaman web sudah bisa memproses kode-kode react.

Dalam bab ini dan beberapa bab ke depan saya ingin menyajikan fitur react yang "minimalis" ini. Kita belum butuh menginstall aplikasi tambahan apapun. Harapannya dengan materi yang sederhana, konsep dasar react menjadi lebih mudah dipahami.

Saya yakin teman-teman sudah punya pilihan teks editor sendiri. Jika belum, [VS Code](https://code.visualstudio.com)¹⁰ bisa menjadi alternatif yang sangat pas. Untuk yang lebih ringan bisa coba [Notepad++](https://notepad-plus-plus.org/)¹¹. Aplikasi teks editor lain pun tidak ada masalah karena React hanya berbentuk file JavaScript biasa.

Untuk menampilkan kode React, saya menggunakan Google Chrome versi terbaru (93.x), tapi seharusnya tetap bisa jalan di web browser modern lain.

3.1. Mengenal File `react.js` dan `react-dom.js`

Sebelumnya sudah di singgung kalau react hanya butuh dua buah file saja, yakni `react.js` dan `react-dom.js`. Apa fungsi file-file ini?

- File `react.js` berisi kode inti library react. Setiap file yang perlu menjalankan kode react wajib mengaksesnya. Disinilah terdapat class utama React seperti `React.Component`, `React.createElement()`, `React.Fragment`, dll.
- File `react-dom.js` berisi kode untuk menampilkan hasil pemrosesan react ke halaman web. Pada file inilah terdapat deklarasi method `ReactDOM.render()` dan `ReactDOM.createRoot().render()` yang hampir selalu kita butuhkan.

Jika sama-sama wajib dipakai, kenapa tidak digabung saja menjadi satu file?

10. <https://code.visualstudio.com>

11. <https://notepad-plus-plus.org/>

Pemisahan dua file ini karena tim pengembang react ingin membuat "library universal" yang memungkinkan react dipakai di luar web. Salah satu hasilnya adalah **React Native** dimana kita bisa membuat aplikasi mobile (android dan iOS) menggunakan React.

Di react native nanti, file `react.js` tetap diperlukan, akan tetapi file `react-dom.js` diganti dengan file lain yang bertugas untuk menampilkan kode React di perangkat mobile.

3.2. Membuat Template HTML

Sekarang saatnya membuat file template untuk mengakses file `react.js` dan `react-dom.js`. Terdapat 2 alternatif: mengakses file react yang ada di CDN atau mendownload kedua file untuk diakses secara offline.

File React di CDN

Salah satu keunggulan mengakses file yang sudah ada di CDN adalah tidak perlu repot-repot mendownloadnya. Cukup tulis ke dalam tag `<script>` dan kode react siap digunakan.

Syaratnya, kita harus terhubung ke internet ketika menjalankan kode program.

Dokumentasi react sudah menyediakan alamat file CDN untuk file `react.js` dan `react-dom.js`, yakni di:

- <https://unpkg.com/react@18/umd/react.development.js>
- <https://unpkg.com/react-dom@18/umd/react-dom.development.js>

Atau versi *minified* di :

- <https://unpkg.com/react@18/umd/react.production.min.js>
- <https://unpkg.com/react-dom@18/umd/react-dom.production.min.js>

Kelompok pertama adalah versi yang belum dikecilkan (*unminified*). Ini ditandai dengan kata "development" di dalam nama file. Anda bebas ingin memakai versi yang mana saja.

Versi *unminified* lebih cocok selama perancangan aplikasi karena kita bisa membaca langsung kode bawaan react. Akan tetapi karena kode itu sudah sangat advanced, saya tidak akan membahasnya dalam buku ini.

Berikut template HTML dasar yang mengakses file react di CDN:

01.template_cdn.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>React Uncover</title>
7  </head>
```

```

8 <body>
9   <h1>Belajar React</h1>
10  <script src="https://unpkg.com/react@18/umd/react.development.js"
11    crossorigin></script>
12  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
13    crossorigin></script>
14  <script>
15    //... kode JavaScript + React disini
16  </script>
17 </body>
18 </html>
```

Di baris 10 dan 12 saya mengakses file `react.development.js` dan `react-dom.development.js` menggunakan tag `<script>` biasa. Tambahan atribut `crossorigin` berfungsi agar web browser "tidak komplain" karena kita mengakses file di luar server dan bisa menampilkan pesan error yang sesuai.

Dengan kedua file ini, di baris 15 kita sudah bisa menulis kode React atau bisa juga membuat file JS terpisah untuk kemudian diakses dari tag `<script>`.

Ketika buku ini saya tulis, react versi terakhir adalah **React 18**. Sangat mungkin sudah ada versi yang lebih baru. Biasanya update versi React tidak terlalu berdampak ke kode yang sudah ada, lebih ke penambahan fitur baru saja.

Namun untuk menghindari error yang mungkin terjadi, saya sangat sarankan untuk tetap di React 18 dahulu. Barulah setelah itu bisa cari info terkait tambahan fitur baru yang ada di versi React saat ini (jika sudah rilis React 19 atau versi yang lebih tinggi).

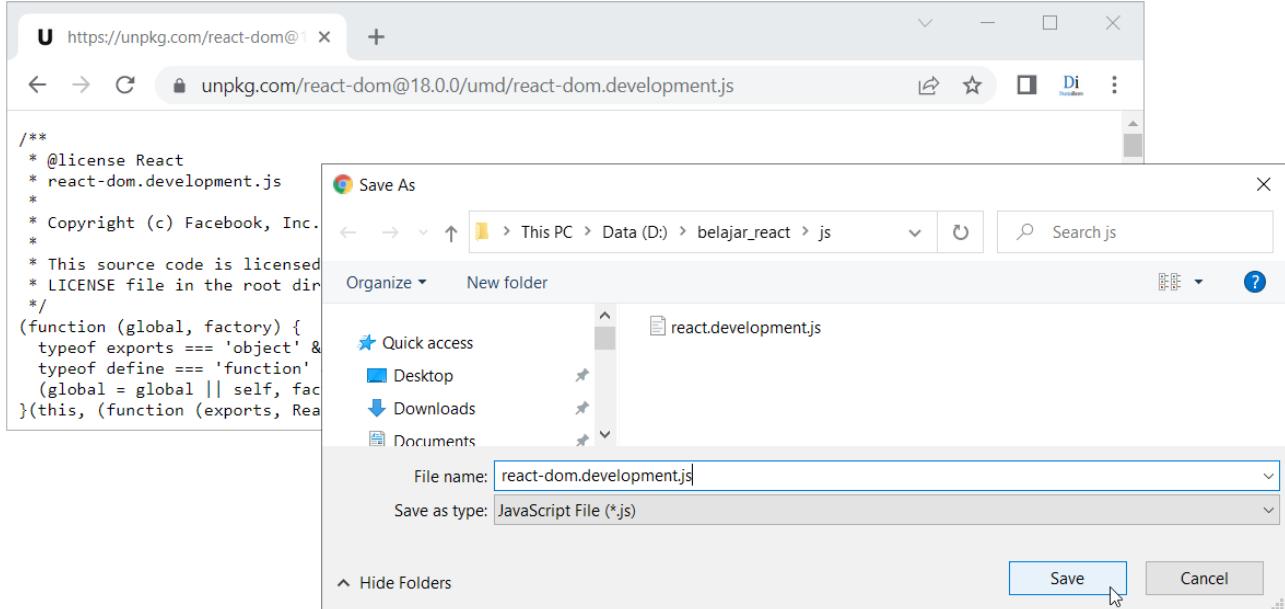
File React di Local

Template react CDN sebelumnya sudah bisa langsung dipakai. Akan tetapi saya merasa lebih baik file react tersebut di download saja untuk diakses di komputer lokal. Tujuannya agar kita bisa menjalankan kode react secara offline dan tidak pengaruh seandainya terjadi gangguan di server CDN atau di koneksi internet.

Sebagai persiapan, silahkan buat folder `belajar_react` di drive D. Kemudian buat juga folder `js` di dalamnya. Setelah itu buka alamat file react berikut di web browser, lalu save ke dalam folder `belajar_react\js\`:

- <https://unpkg.com/react@18/umd/react.development.js>
- <https://unpkg.com/react-dom@18/umd/react-dom.development.js>

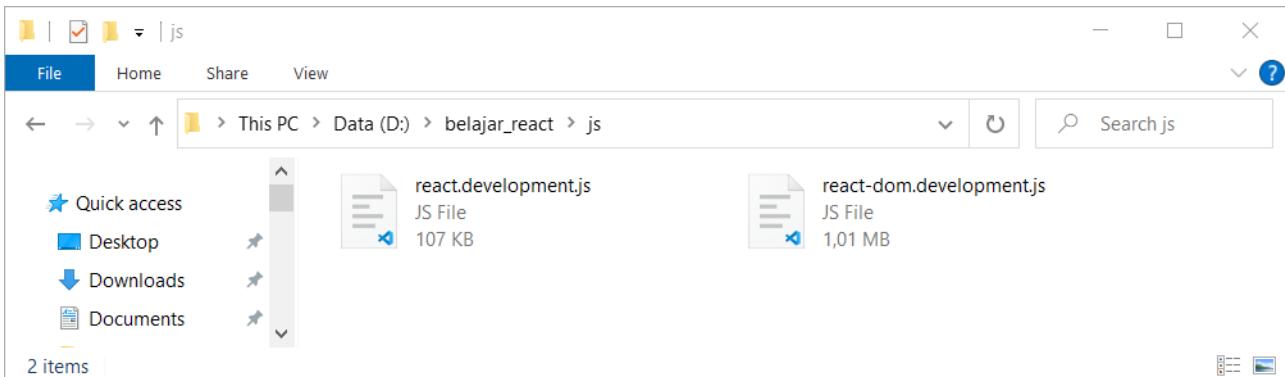
React dan React DOM



Gambar: Simpan file react ke dalam folder D:\belajar_react\js\

Setelah selesai, di dalam folder `belajar_react\js\` akan terdapat 2 buah file:

- `react.development.js`
- `react-dom.development.js`



Gambar: Isi folder D:\belajar_react\js\

Kembali ke folder `belajar_react`, buat satu file HTML yang mengakses kedua file React ini:

02.template_offline.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>React Uncover</title>
7  </head>
8  <body>
9      <h1>Belajar React</h1>
10     <script src="js/react.development.js"></script>
```

```
11 <script src="js/react-dom.development.js"></script>
12 <script>
13 //... kode JavaScript + React disini
14 </script>
15 </body>
16 </html>
```

Sip, template kode React sudah siap digunakan. Sepanjang buku ini saya akan memakai file template versi offline di atas, kecuali dinyatakan lain.

Jika anda kesulitan mendownload file React, bisa diambil dari file `belajar_react.zip` yang tersedia di Google Drive.

3.3. Method React.createElement()

Method react pertama yang akan kita bahas adalah `React.createElement()`. Method ini berfungsi untuk membuat *react element*.

React element adalah sebuah object yang dalam banyak hal mirip seperti *node element* hasil perintah `document.getElementById()`. Akan tetapi react element lebih efisien daripada *node element* biasa.

Berikut contoh penggunaan method `React.createElement()`:

03.create_element.html

```
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>React Uncover</title>
7 </head>
8 <body>
9   <h1>Belajar React</h1>
10  <script src="js/react.development.js"></script>
11  <script src="js/react-dom.development.js"></script>
12  <script>
13    const myElement = React.createElement('h1', {id:'judul'}, 'Hello React');
14    console.log(myElement);
15  </script>
16 </body>
17 </html>
```

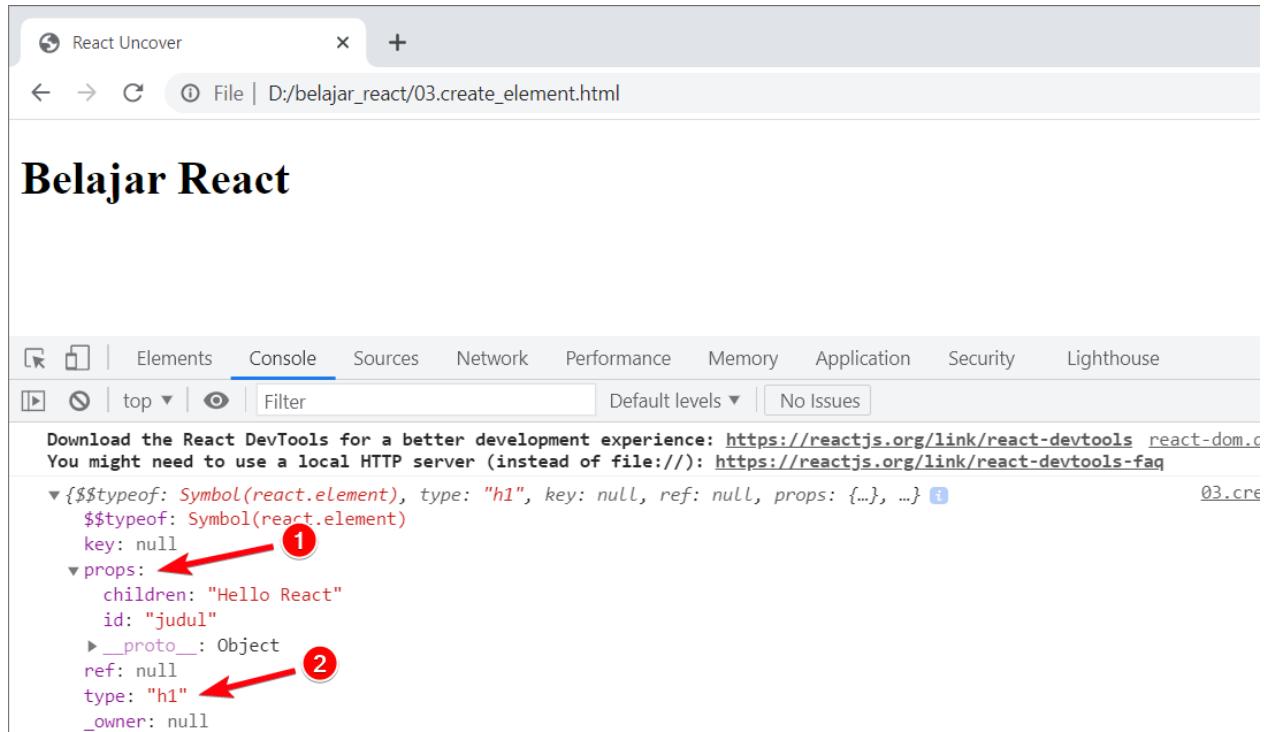
Di baris 13 saya mengisi konstanta `myElement` dengan hasil pemanggilan method `React.createElement()`. Method ini butuh 3 buah argument:

- Argument pertama diisi tipe data string dari jenis element HTML yang akan dibuat, misalnya '`p`', '`div`', '`h1`', '`article`', atau element HTML lain. Untuk penggunaan yang

lebih advanced, argument pertama ini juga bisa diisi *react component* (akan kita bahas dalam bab tersendiri).

- Argument kedua diisi atribut dari react element. Atribut ini sama seperti yang kita tulis ke dalam element HTML seperti `id`, `class`, `src`, `alt`, atau bisa juga *custom attribute* yang dibuat sendiri. Atribut ditulis dalam bentuk pasangan property object.
- Argument ketiga diisi "anak" atau *child* dari react element. Ini mirip seperti `children` pada *node element* yang bisa diisi teks atau react element lain.

Setelah membuat method `React.createElement()`, di baris 14 saya menampilkan isi konstanta `myElement` menggunakan perintah `console.log(myElement)`. Berikut hasilnya:



```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools react-dom.c
You might need to use a local HTTP server (instead of file://): https://reactjs.org/link/react-devtools-faq 03.cre
v {  
  $typeof: Symbol(react.element), type: "h1", key: null, ref: null, props: {...}, ...} ⓘ  
    $typeof: Symbol(react.element)  
    key: null  
    props:  
      children: "Hello React"  
      id: "judul"  
      __proto__: Object  
      ref: null  
      type: "h1"  
      _owner: null
  
```

Gambar: Hasil `console.log(myElement)`

Di dalam tab console, terlihat kode yang cukup njelimet. Kita tidak perlu pahami semua ini karena hanya dipakai secara internal oleh React.

Namun perhatikan terdapat nilai "props" (1) yang berisi dua item, yakni `children: "Hello React"` dan `id: "judul"`. Nilai ini berasal dari argument ketiga dan kedua saat penulisan `React.createElement()`. Sedangkan argument pertama tersimpan di property `type: "h1"` (2).

Di dalam React, kata `props` merupakan kependekan dari *properties*. Istilah `props` ini nantinya akan kita bahas secara mendalam karena menjadi bagian dari konsep inti React.

3.4. Method `ReactDOM.createRoot().render()`

Method react kedua yang akan kita bahas adalah `ReactDOM.createRoot().render()`. Sesuai

namanya, method ini bagian dari `react-dom.js`. Sedangkan method `React.createElement()` sebelumnya adalah milik file `react.js`.

Method `ReactDOM.createRoot().render()` bertugas untuk menampilkan *react element* ke dalam halaman HTML, atau lebih tepatnya ke dalam struktur DOM.

Sebagaimana yang kita pelajari pada materi JavaScript dasar (misalnya dari buku *JavaScript Uncover*). **DOM** atau *document object model* adalah struktur internal dari semua element yang ada di dalam halaman HTML.

Ketika file HTML tampil pertama kali, struktur DOM akan dibuat. Setelah itu jika kita ingin menampilkan element baru atau memodifikasi tampilan, struktur DOM ini juga harus diperbaharui.

Di dalam JavaScript biasa (native), element HTML bisa dibuat dari method `document.createElement()`, lalu kita mengisi teks ke dalam element tersebut untuk kemudian di-inject ke struktur DOM menggunakan method `appendChild()` atau property `innerHTML`.

React memiliki pendekatan berbeda. Agar lebih efisien, semua element HTML di petakan ke dalam **virtual DOM** terlebih dahulu. Kode internal React kemudian memproses dan menampilkan hasilnya ke DOM web browser. Inilah fungsi dari method `ReactDOM.createRoot().render()`, yakni me-render virtual DOM milik React ke DOM web browser agar hasilnya bisa terlihat.

Dalam kode sebelumnya, method `React.createElement('h1', {id:'judul'}, 'Hello React')` akan membuat suatu tag HTML dengan bentuk berikut:

```
<h1 id='judul'>'Hello React'</h1>
```

Method `ReactDOM.createRoot().render()` akan "menyisipkan" element ini ke dalam struktur DOM. Berikut kode yang diperlukan:

04.dom-render.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>React Uncover</title>
7      <style>
8          #judul {
9              color:blueviolet;
10             text-shadow: 1px 1px 2px ;
11             font-family:Verdana, Geneva, Tahoma, sans-serif
12         }
13     </style>
14 </head>
15 <body>
16     <h1>Belajar React</h1>
```

```
17 <div id="root"></div>
18
19
20 <script src="js/react.development.js"></script>
21 <script src="js/react-dom.development.js"></script>
22 <script>
23   const myElement = React.createElement('h1', {id:'judul'}, 'Hello React');
24   ReactDOM.createRoot(document.getElementById('root')).render(myElement);
25 </script>
26 </body>
27 </html>
```

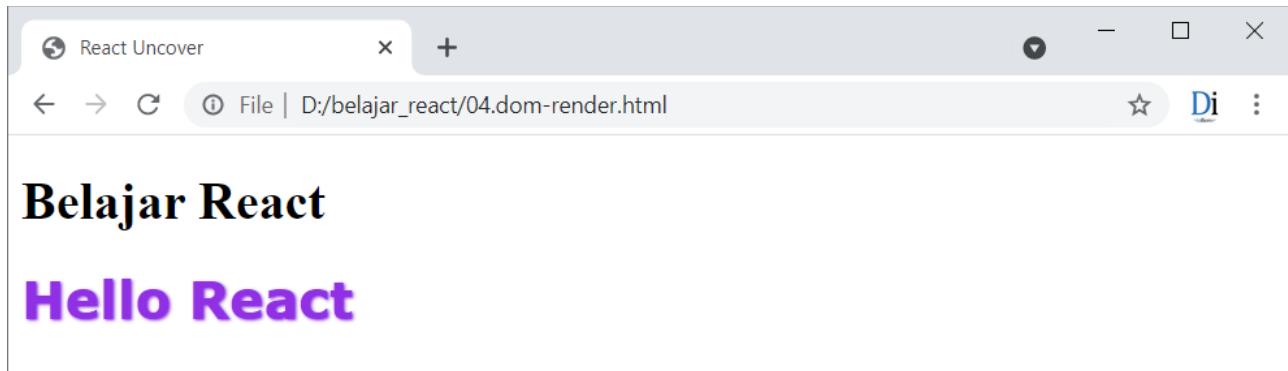
Pemanggilan method `ReactDOM.createRoot().render()` ada di baris 24. Perintah ini sebenarnya perpaduan dari 2 buah method:

- `ReactDOM.createRoot()`, berguna untuk mencari node element dari struktur HTML yang sudah ada. Disinilah tempat element akan disisipkan.
- `render()`, diisi dengan *react element* yang ingin ditampilkan.

Sebagai argument dari method `ReactDOM.createRoot()`, saya isi dengan perintah bawaan JavaScript, yakni `document.getElementById('root')`. Method ini akan mencari satu element HTML di dalam struktur DOM yang memiliki `id = root`. Element yang dimaksud berupa tag `<div id="root">` di baris 18.

Untuk method `render()`, argument-nya saya input dengan konstanta `myElement`. Konstanta ini berisi react element yang dihasilkan dari method `React.createElement()` di baris 23.

Berikut hasil dari kode program di atas:

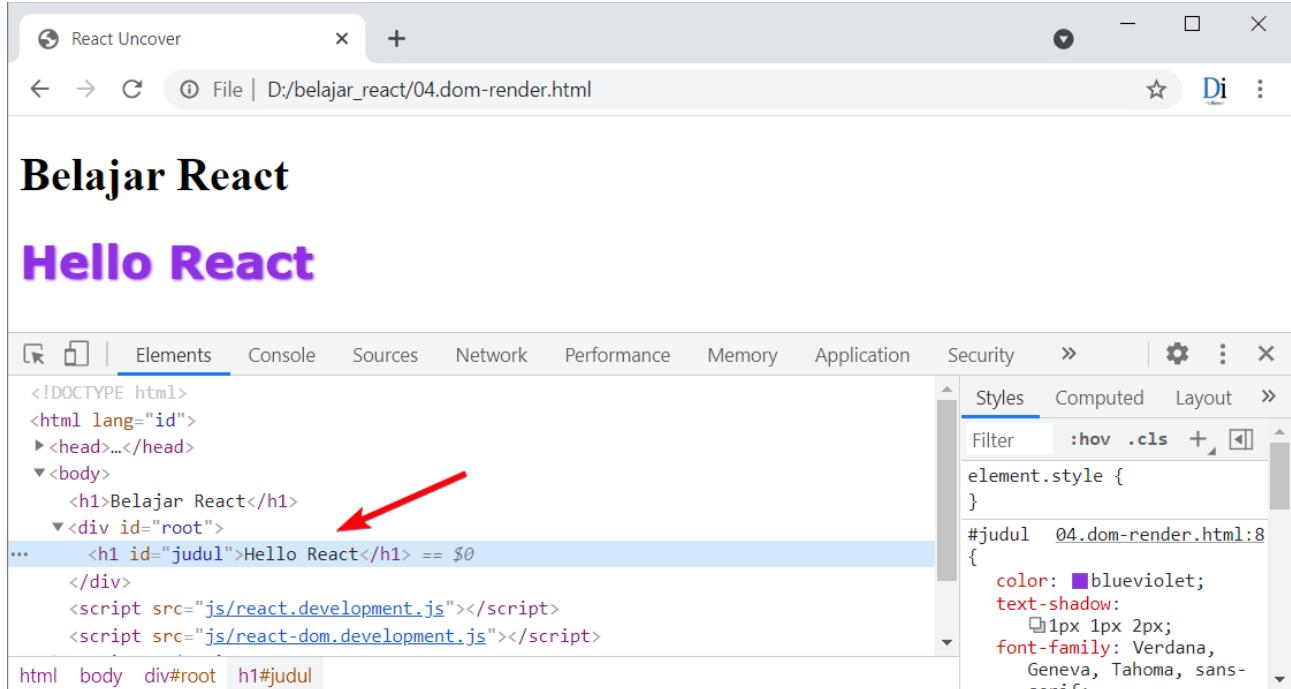


Gambar: Menampilkan react element ke halaman web

Dengan tampilnya teks "Hello React" di web browser, maka kita sudah sukses memproses react element.

Teks "Hello React" berwarna abu-abu dan mendapat efek shadow dari tambahan kode CSS di baris 8 – 12. Ini memperlihatkan bahwa element hasil react tidak berbeda seperti tag HTML biasa.

Jika inspect element dibuka, akan tampil struktur DOM yang sudah di update:



Gambar: Tag `<h1>` yang berasal dari React

Terlihat tag `<h1 id='judul'>'Hello React'</h1>` menjadi child dari tag `<div id="root">`. Inilah hasil dari method `ReactDOM.createRoot().render()`. Jika di dalam tag `<div id="root">` terdapat element lain, isinya akan ditimpak oleh `ReactDOM.createRoot().render()`.

Jika diperlukan, penulisan perintah `ReactDOM.createRoot().render()` bisa kita pisah ke dalam 2 baris seperti contoh berikut:

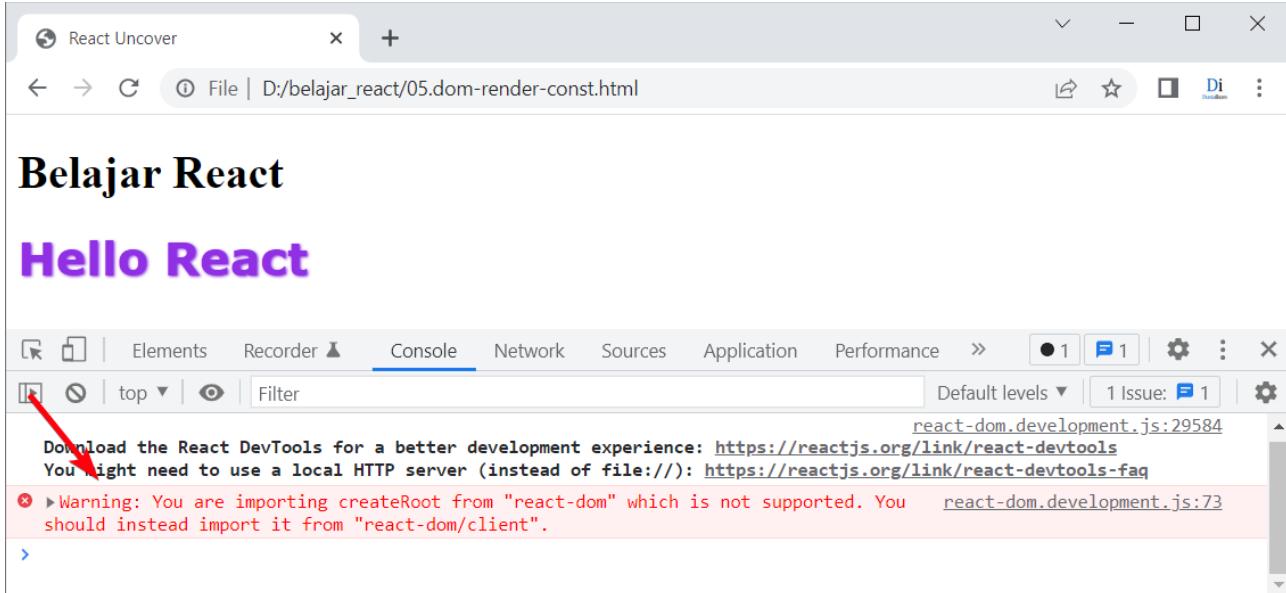
05.dom-render-const.html

```
1 ...
2   const myElement = React.createElement('h1', {id:'judul'}, 'Hello React');
3   const root = ReactDOM.createRoot(document.getElementById('root'));
4   root.render(myElement);
```

Di baris 3 saya membuat konstanta `root` yang berisi hanya hasil dari perintah `ReactDOM.createRoot()`. Barulah di baris 4 konstanta `root` disambung dengan method `render()`. Ini sekedar variasi penulisan saja, anda bebas memilih apakah ingin menggabungnya dalam 1 baris, atau memisahkan penulisan `ReactDOM.createRoot()` dengan `render()`.

Pesan Warning di Tab Console

Jika kita membuka tab console di developer tools, akan muncul pesan warning berikut: `You are importing createRoot from "react-dom" which is not supported. You should instead import it from "react-dom/client".`



Gambar: Pesan warning terkait import react-dom/client

Pesan ini terkait dengan proses import library ReactDOM. React 18 mewajibkan proses import ReactDOM dari "react-dom/client", akan tetapi ini belum bisa kita lakukan jika file library di import menggunakan tag <script>.

Proses import react-dom/client baru bisa dilakukan dari modul `create react app` yang akan kita pelajari di pertengahan buku nanti. Jadi untuk sementara, pesan error ini boleh diabaikan.

3.5. Method ReactDOM.render()

Method `ReactDOM.createRoot().render()` yang baru saja kita pelajari merupakan salah satu fitur baru di React 18. Sebelumnya di React 17 ke bawah, method yang dipakai untuk merender tampilan adalah `ReactDOM.render()`.

Karena React 18 masih relatif baru, mayoritas tutorial yang ada di internet masih memakai `ReactDOM.render()`, sehingga tidak ada salahnya kita bahas sekilas. Terlebih perintah ini juga masih bisa dipakai di React 18.

Cara penggunaan `ReactDOM.render()` sangat mirip seperti `ReactDOM.createRoot().render()`, malah lebih sederhana. Berikut contoh kode program "Hello React" sebelumnya yang kali ini memakai method `ReactDOM.render()`:

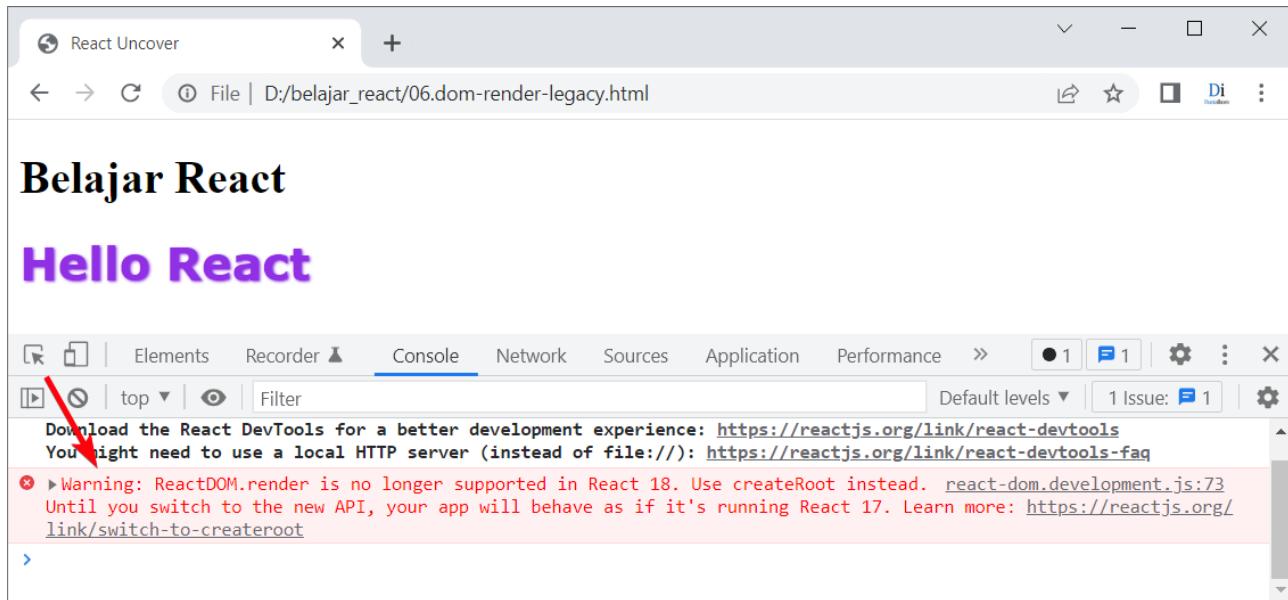
06.dom-render-legacy.html

```
1 ...
2 <body>
3   <h1>Belajar React</h1>
4
5   <div id="root"></div>
6
```

```
7 <script src="js/react.development.js"></script>
8 <script src="js/react-dom.development.js"></script>
9 <script>
10   const myElement = React.createElement('h1', {id:'judul'}, 'Hello React');
11   ReactDOM.render(myElement, document.getElementById('root'));
12 </script>
13 </body>
```

Pemanggilan method `ReactDOM.render()` ada di baris 11. Method ini butuh 2 buah argument, argument pertama untuk react element yang ingin ditampilkan, dan argument kedua berupa lokasi tempat react element akan diinput.

Hasil akhir dari kode di atas sama persis seperti versi `ReactDOM.createRoot().render()`, hanya saja di bagian tab console akan muncul pesan warning baru:



Gambar: Pesan warning terkait `ReactDOM.render`

Peringatan ini berisi pesan bahwa method `ReactDOM.render()` tidak lagi di dukung oleh React 18. Kita disarankan beralih ke `ReactDOM.createRoot()`. Selain itu ada pemberitahuan bahwa kode yang ada akan di proses seolah-olah berada di React 17.

Untuk program sederhana, nyaris tidak ada perbedaan antara kode untuk React 17 dan React 18. Hampir semua kode di buku ini tetap bisa berjalan di React 17, sehingga jika anda ingin memakai versi `ReactDOM.render()` juga tidak masalah.

Sebagai pembanding, kedua kode berikut memiliki makna sama, yakni menampilkan `myElement` ke dalam tag `<div id="root">`:

Versi **React 17** ke bawah:

```
ReactDOM.render(myElement, document.getElementById('root'));
```

Versi **React 18**:

```
ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Pakai File React 17 saja?

Jika anda tidak ingin melihat pesan warning terkait import `react-dom/client` saat memakai `ReactDOM.createRoot().render`, maupun pesan error `ReactDOM.render()`, bisa menggunakan file React 17 untuk sementara waktu. Setidaknya hingga kita sampai ke bab `create react app`.

Nyaris semua kode dasar React yang akan kita pelajari tetap sama di React 17 maupun React 18. Link untuk React 17 bisa didapat dari alamat berikut:

<https://unpkg.com/react@17/umd/react.development.js>
<https://unpkg.com/react-dom@17/umd/react-dom.development.js>

Silahkan save seperti sebelumnya dan simpan ke dalam folder `js`.

Jika dalam tampilan di buku nanti tidak terlihat pesan warning, itu berarti saya menggunakan file React 17, bukan file React 18.

Saya berkeyakinan masalah ini hanya sementara, menunggu tim React memberi solusi agar kita mengakses `react-dom/client` menggunakan tag `<script>`, tidak harus lewat `create react app`.

3.6. Multiple ReactDOM.createRoot().render()

Dalam praktek sebelumnya kita sudah berhasil menampilkan 1 react element. Jika ingin menampilkan element lain, tinggal jalankan method `ReactDOM.createRoot().render()` beberapa kali seperti contoh berikut:

05.multiple_render.html

```

1 ...
2 <body>
3   <h1>Belajar React</h1>
4
5   <div id="root"></div>
6   <div id="foo"></div>
7
8   <script src="js/react.development.js"></script>
9   <script src="js/react-dom.development.js"></script>
10  <script>
11    const myElement = React.createElement('h1', null, 'Hello React');
12    ReactDOM.createRoot(document.getElementById('root')).render(myElement);
13
14    const myPic = React.createElement('img', { src: 'img/landak.jpg' }, null);
15    ReactDOM.createRoot(document.getElementById('foo')).render(myPic);
16  </script>
17 </body>
```

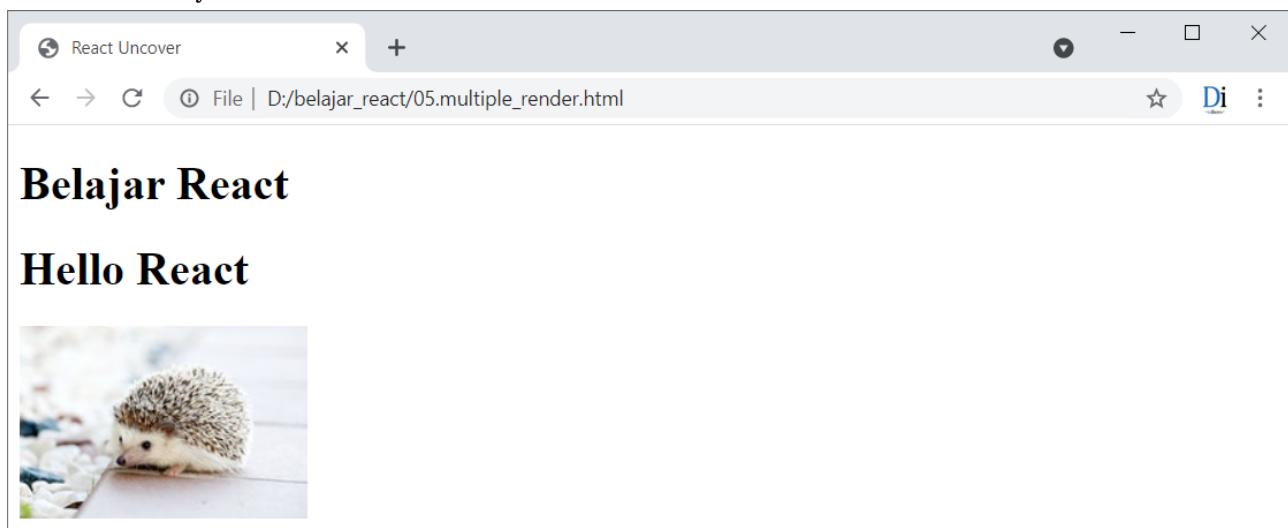
Kali ini saya membuat 2 buah *react element*, di dalam konstanta `myElement` pada baris 11 dan konstanta `myPic` pada baris 14.

Konstanta `myElement` berisi hasil method `React.createElement()` yang sama seperti praktek sebelumnya. Hanya saja sekarang argument kedua diisi nilai `null` yang berarti saya tidak ingin menambah atribut apapun. Element yang dihasilkan adalah `<h1>Hello React</h1>`.

Untuk konstanta `myPic`, diisi dengan string '`img`' sebagai argument pertama, object `src`: '`img/landak.jpg`' sebagai argument kedua, dan nilai `null` sebagai argument ketiga. Ini akan menghasilkan element ``.

Melalui method `ReactDOM.createRoot().render()`, konstanta `myElement` diinput ke dalam tag `<div id="root">` sedangkan konstanta `myPic` akan tampil ke dalam tag `<div id="foo">`. Kedua tag ini sudah saya siapkan di baris 5-6.

Berikut hasilnya:



Gambar: Hasil multiple ReactDOM.createRoot().render()

Di bawah teks "Hello React" terlihat gambar landak. Gambar tersebut diambil dari file `landak.jpg` yang ada di folder `img`. File ini sudah saya siapkan sebelumnya dan bebas jika ingin diganti dengan gambar lain.

Inti dari praktek kita adalah, method `ReactDOM.createRoot().render()` bisa dipanggil lebih dari satu kali.

3.7. Nested React Element

Untuk praktek yang lebih real, kita bisa membuat element HTML "bersarang" atau *nested element*, maksudnya ada element di dalam element. Struktur tabel, form, atau list, semuanya butuh perpaduan banyak element HTML.

Argument ketiga dari method `React.createElement()` bisa diisi dengan *child element*. Di sini

kita bisa jalankan method `React.createElement()` lain untuk membuat struktur nested element. Berikut contoh penggunaannya:

06.nested_element.html

```

1 ...
2 <body>
3   <h1>Belajar React</h1>
4
5   <div id="root"></div>
6
7   <script src="js/react.development.js"></script>
8   <script src="js/react-dom.development.js"></script>
9   <script>
10    const myMenu = React.createElement('ul', {}, [
11      [
12        React.createElement('li', {}, 'Espresso'),
13        React.createElement('li', {}, 'Cappuccino'),
14        React.createElement('li', {}, 'Moccacino')
15      ]
16    );
17
18    ReactDOM.createRoot(document.getElementById('root')).render(myMenu);
19  </script>
20 </body>
```

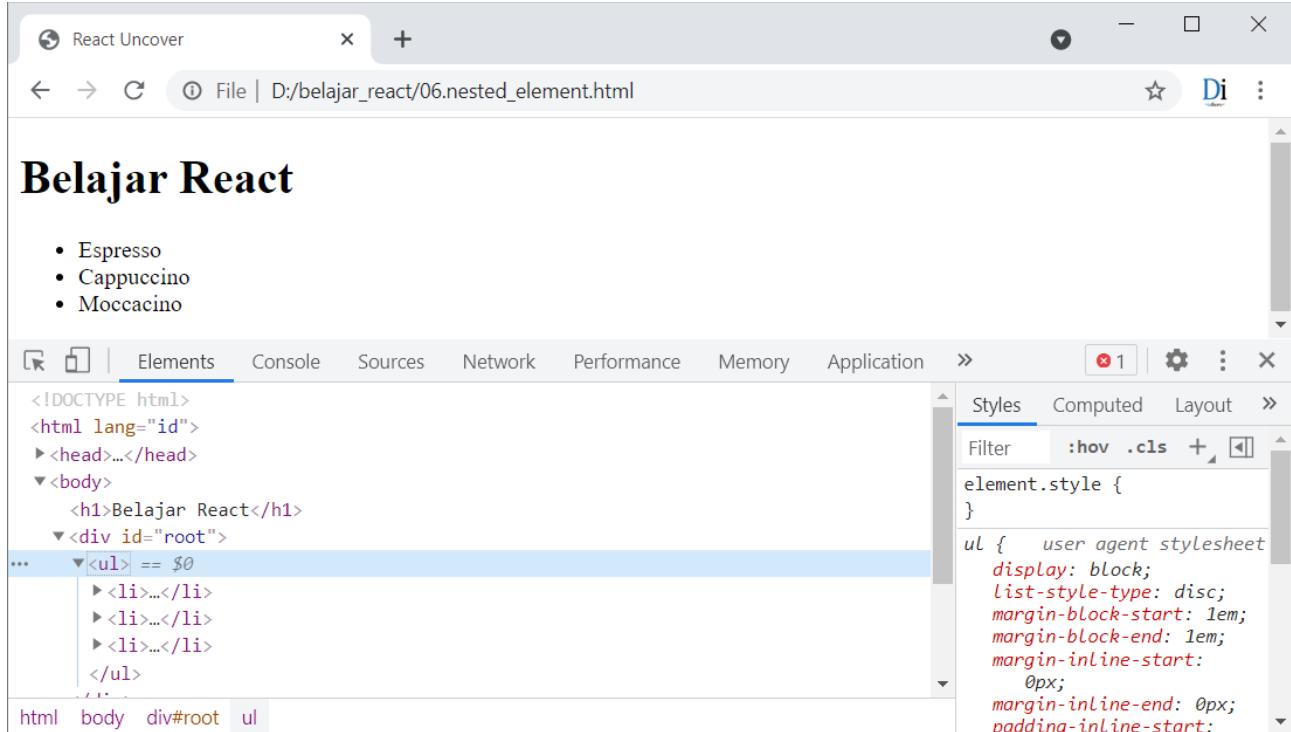
Untuk argument ketiga dari `React.createElement()` di baris 10, saya isi dengan array yang elementnya terdiri dari 3 kali pemanggilan method `React.createElement()` antara baris 12 – 14. Inilah cara membuat *nested element* di React.

Dengan kode tersebut, konstanta `myMenu` akan berisi struktur HTML berikut:

```
<ul>
  <li>Espresso</li>
  <li>Cappuccino</li>
  <li>Moccacino</li>
</ul>
```

Sebagai tambahan, argument kedua dari setiap method `React.createElement()` di isi dengan object kosong " {} ". Ini adalah cara lain untuk menandakan kita tidak butuh atribut apa-apa.

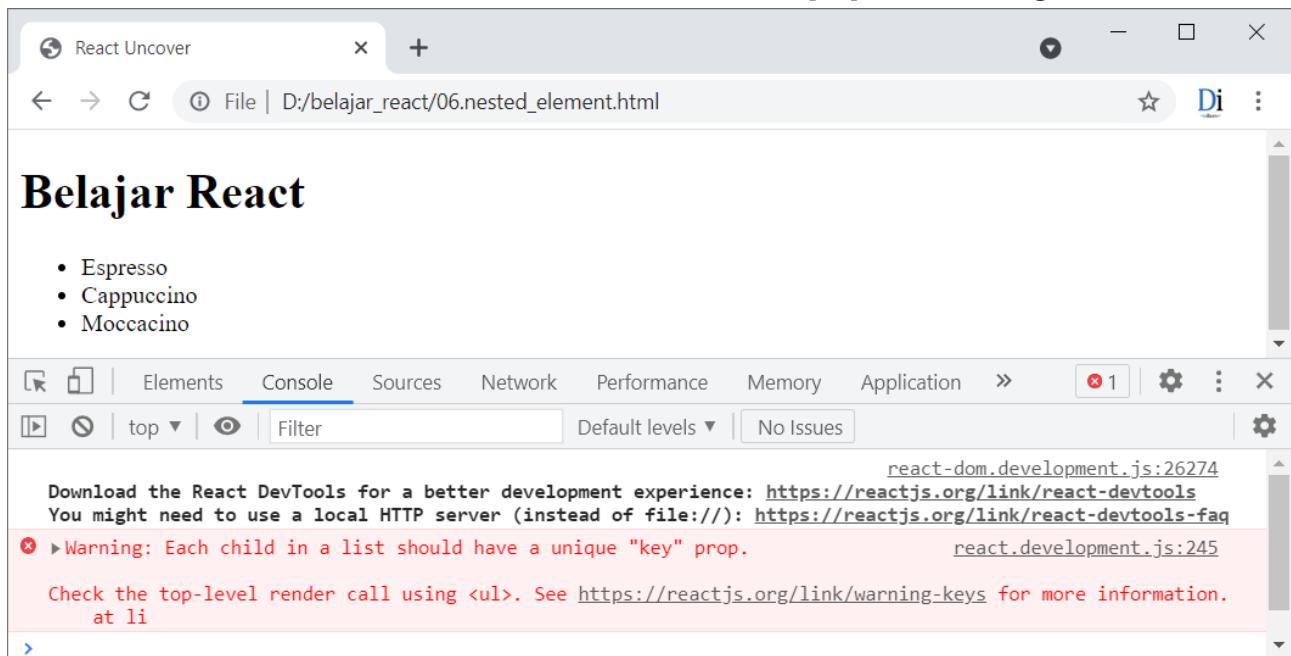
Berikut hasil dari kode di atas:



Gambar: Hasil nested React.createElement()

Konstanta `myMenu` sukses ditampilkan dan berbentuk sebuah unordered list.

Namun ada sedikit masalah. Ketika tab console dibuka, tampil pesan warning berikut:



Gambar: Pesan warning dari react

Yang perlu menjadi perhatian, ini adalah pesan level warning, bukan error. Sebenarnya tidak ada yang salah dari kode program kita, akan tetapi React butuh suatu tambahan optimasi ketika memproses data berulang seperti list.

Tim pengembang react ingin membuat proses manipulasi DOM se-efisien mungkin. Salah satu cara adalah dengan mewajibkan setiap react element memiliki identitas khusus agar mudah membedakan satu element dengan element lain. Untuk tag berulang seperti ``, React butuh tambahan atribut **key**.

Inilah maksud dari pesan "Warning: Each child in a list should have a unique "key" prop" di tab console. Pesan warning ini akan sering kita temui jika lupa menambah atribut key saat menampilkan data berulang.

Untungnya, yang diinginkan React cukup mudah dibuat. Cukup tambah atribut key dengan nilai unik ke dalam setiap *react element*. Nilai atribut key boleh bebas, entah itu angka, huruf, atau kombinasi keduanya selama tidak ada yang berulang.

Berikut salah satu solusi yang bisa kita gunakan:

07.nested_element_key.html

```

1  ...
2  <body>
3      <h1>Belajar React</h1>
4
5      <div id="root"></div>
6
7      <script src="js/react.development.js"></script>
8      <script src="js/react-dom.development.js"></script>
9      <script>
10         const myMenu = React.createElement('ul', {}, [
11             [
12                 React.createElement('li', {key: 'a'}, 'Espresso'),
13                 React.createElement('li', {key: 'b'}, 'Cappuccino'),
14                 React.createElement('li', {key: 'c'}, 'Moccacino')
15             ]
16         );
17
18         ReactDOM.createRoot(document.getElementById('root')).render(myMenu);
19     </script>
20 </body>

```

Sebagai argument kedua dari method `React.createElement()`, saya isi dengan atribut key dengan nilai huruf 'a', 'b' dan 'c'. Hasilnya, pesan warning terkait key sudah tidak keluar lagi.

Exercise

Sebagai penutup bab, saya ingin membuat latihan sederhana. Silahkan modifikasi kode program list menu di atas, lalu generate react element dengan link di setiap list.

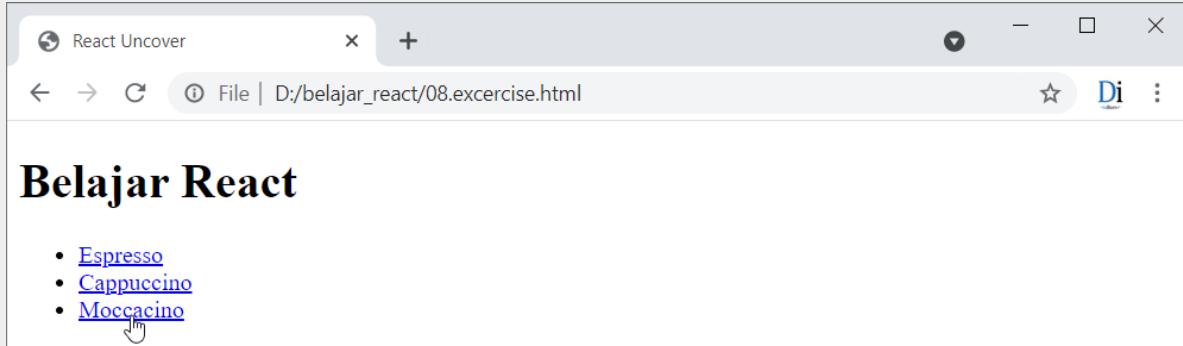
Hasil akhir struktur HTML yang diinginkan adalah sebagai berikut:

```

1 <div id="root">
2   <ul>

```

```
3   <li><a href="https://id.wikipedia.org/wiki/Espreso">Espresso</a></li>
4   <li><a href="https://id.wikipedia.org/wiki/Kapuccino">Cappuccino</a></li>
5   <li><a href="https://id.wikipedia.org/wiki/Moka">Moccacino</a></li>
6 </ul>
7 </div>
```



Gambar: Hasil exercise react element

Sedikit tips, react element 'a' nantinya akan menjadi *child* dari setiap element 'li'.

Solution

Berikut kode yang diperlukan:

08.exercise.html

```
1  <script>
2    const myMenu = React.createElement('ul', {},
3      [
4        React.createElement('li', {key:'a'},
5          React.createElement('a',
6            {href:'https://id.wikipedia.org/wiki/Espreso'}, 'Espresso')
7        ),
8        React.createElement('li', {key:'b'},
9          React.createElement('a',
10            {href:'https://id.wikipedia.org/wiki/Kapuccino'}, 'Cappuccino')
11        ),
12        React.createElement('li', {key:'c'},
13          React.createElement('a',
14            {href:'https://id.wikipedia.org/wiki/Moka'}, 'Moccacino')
15        )
16      ]
17    );
18
19  ReactDOM.createRoot(document.getElementById('root')).render(myMenu);
20 </script>
```

Selain mengisi react element 'a' sebagai child dari react element 'li', kita juga harus menulis atribut href serta memindahkan teks ke dalam element 'a'.

Sepanjang bab ini kita telah membahas tentang file `react.js` dan `react-dom.js` yang menjadi syarat untuk bisa menjalankan kode-kode React. Dengan method `React.createElement()` dan `ReactDOM.createRoot().render()`, kita juga berhasil men-generate `react element` untuk kemudian ditampilkan ke dalam halaman web.

Akan tetapi kode yang diperlukan untuk `React.createElement()` memang sangat panjang, terlebih jika ingin membuat struktur DOM yang bertingkat seperti list, tabel, atau form. Untungnya, React sudah menyediakan cara penulisan yang lebih mudah, dan itulah yang akan kita bahas dalam bab selanjutnya, yakni tentang **JSX**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

=====

4. JSX (JavaScript XML)

Salah satu struktur terpenting di React adalah *react element*. Semua materi yang akan kita bahas sampai akhir buku nanti dipakai untuk memanipulasi element ini.

Dalam bab sebelumnya telah dibahas bahwa React element dibuat menggunakan method `React.createElement()`. Untuk element yang sederhana, penggunaannya masih terasa mudah. Akan tetapi bisa menjadi sangat rumit untuk halaman web yang perlu ratusan tag HTML. Sebagai alternatif yang lebih praktis, React mengadopsi cara penulisan **JSX**.

4.1. Pengertian JSX

JSX (singkatan dari **JavaScript XML**) adalah sebuah extension JavaScript untuk memudahkan pembuatan react element. Penulisan JSX sangat mirip seperti kode HTML biasa. Sebagai contoh, sebelumnya kita membuat react element dengan kode berikut:

```
const myElement = React.createElement('h1', {id:'judul'}, 'Hello React');
```

Dengan JSX, bisa ditulis menjadi:

```
const myElement = <h1 id='judul'>Hello React</h1>;
```

Terlihat JSX nyaris sama seperti kode HTML biasa. Namun perhatikan bahwa teks yang di input ke dalam konstanta `myElement` bukanlah sebuah string, tapi perintah JavaScript. Oleh karena itu tidak ditulis dalam tanda kutip.

Dengan JSX, kita bisa membuat struktur nested HTML dengan lebih mudah:

```
1 const myElement = (
2   <ul>
3     <li><a href="https://id.wikipedia.org/wiki/Espreso">Espresso</a></li>
4     <li><a href="https://id.wikipedia.org/wiki/Kapuccino">Cappuccino</a></li>
5     <li><a href="https://id.wikipedia.org/wiki/Moka">Moccacino</a></li>
6   </ul>
7 )
```

Kode ini akan menghasilkan *react element* yang sama seperti struktur menu di *exercise* bab sebelumnya.

Dalam dunia programming, JSX ini termasuk kelompok fitur yang disebut sebagai [syntactic](#)

sugar¹² atau "pemanis", yakni fitur tambahan untuk memudahkan penulisan kode program dari kode yang sudah ada.

Selain penulisan yang lebih mudah, JSX juga memiliki aturan, fitur dan batasan tersendiri. Kita akan bahas satu per satu dengan lebih detail.

Kenapa "JavaScript XML" dan bukannya "JavaScript HTML"?

XML pada awalnya merupakan "induk" atau *superset* dari HTML. Istilah "tag" serta cara penulisan kode HTML dengan tanda kurung siku "< />" berasal dari XML. Dengan kata lain, HTML adalah bagian atau *subset* dari XML.

Saat ini HTML5 tidak lagi mengikuti aturan baku XML dan menjadi bahasa markup terpisah. Salah satu aturan XML yang "dilanggar" HTML adalah tidak butuh tag penutup seperti
 atau <hr>. Akan tetapi di XML semua tag harus ditutup seperti
 dan <hr />.

Tim pengembang JSX memilih XML agar penggunaannya lebih luas dari sekedar HTML. Ketika merancang tampilan aplikasi android, itu juga menggunakan XML (JSX nantinya juga dipakai untuk React Native).

Selain itu tentu saja singkatan "JSX" terdengar lebih keren daripada "JSH" (JavaScript HTML).

4.2. Mengakses File Babel

Agar bisa menggunakan JSX, kita perlu bantuan library khusus karena JSX bukanlah bagian dari JavaScript native. Library yang dimaksud adalah Babel¹³. Babel sering dipakai untuk "menerjemahkan" kode-kode JavaScript terbaru agar bisa berjalan di web browser lama.

Sebagai contoh, fitur ES6 seperti *template string*, *arrow function*, atau *spread operator* bisa dipastikan tidak akan berjalan di web browser tua. Jika web kita diakses, bisa jadi tampil berantakan atau tidak bisa di akses sama sekali.

Daripada menulis ulang kode JavaScript, Babel bisa menerjemahkan kode ES6 menjadi kode JS biasa (ES5). Ini dikenal dengan istilah *transcompiler*.

Selain tugas utama sebagai transcompiler, Babel juga mendukung proses konversi kode JSX menjadi method `React.createElement()`. Betul, secara internal Babel akan membaca kode-kode JSX untuk diterjemahkan menjadi method `React.createElement()`.

Sama seperti file `react.js` dan `react-dom.js`, file `babel.js` juga tersedia di CDN berikut:

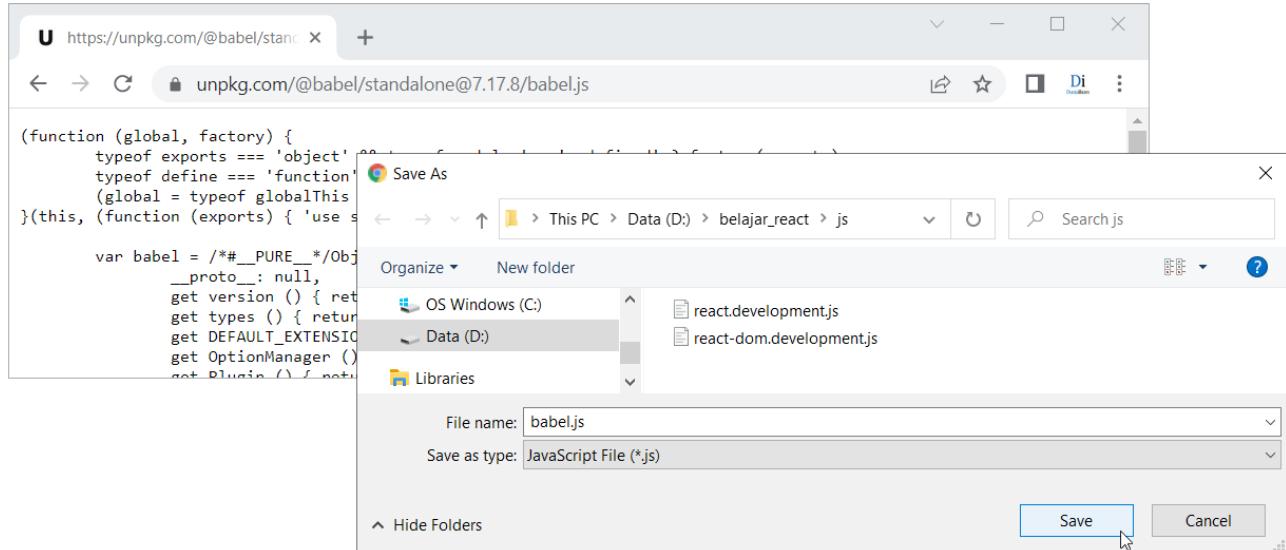
<https://unpkg.com/@babel/standalone/babel.js>

12 https://en.wikipedia.org/wiki/Syntactic_sugar

13 <https://babeljs.io>

JSX (JavaScript XML)

Agar lebih praktis, silahkan download file tersebut lalu simpan ke folder belajar_react/js, yakni lokasi yang sama dengan file react.js dan react-dom.js:



Gambar: Simpan file babel.js ke folder belajar_react/js

Sekarang kita sudah bisa membuat file template untuk menjalankan JSX:

01.babel_template.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>React Uncover</title>
7  </head>
8  <body>
9      <div id="root"></div>
10
11     <script src="js/react.development.js"></script>
12     <script src="js/react-dom.development.js"></script>
13     <script src="js/babel.js"></script>
14     <script type="text/babel">
15         //... kode JavaScript + React disini
16     </script>
17 </body>
18 </html>
```

Saya mengakses 3 file JavaScript antara baris 11 - 13, yakni js/react.development.js, js/react-dom.development.js, dan js/babel.js.

Selain itu perhatikan baris 14, terdapat tambahan atribut `type="text/babel"` ke dalam tag `<script>`. Atribut ini wajib ditambah agar kode JavaScript bisa diproses oleh Babel.

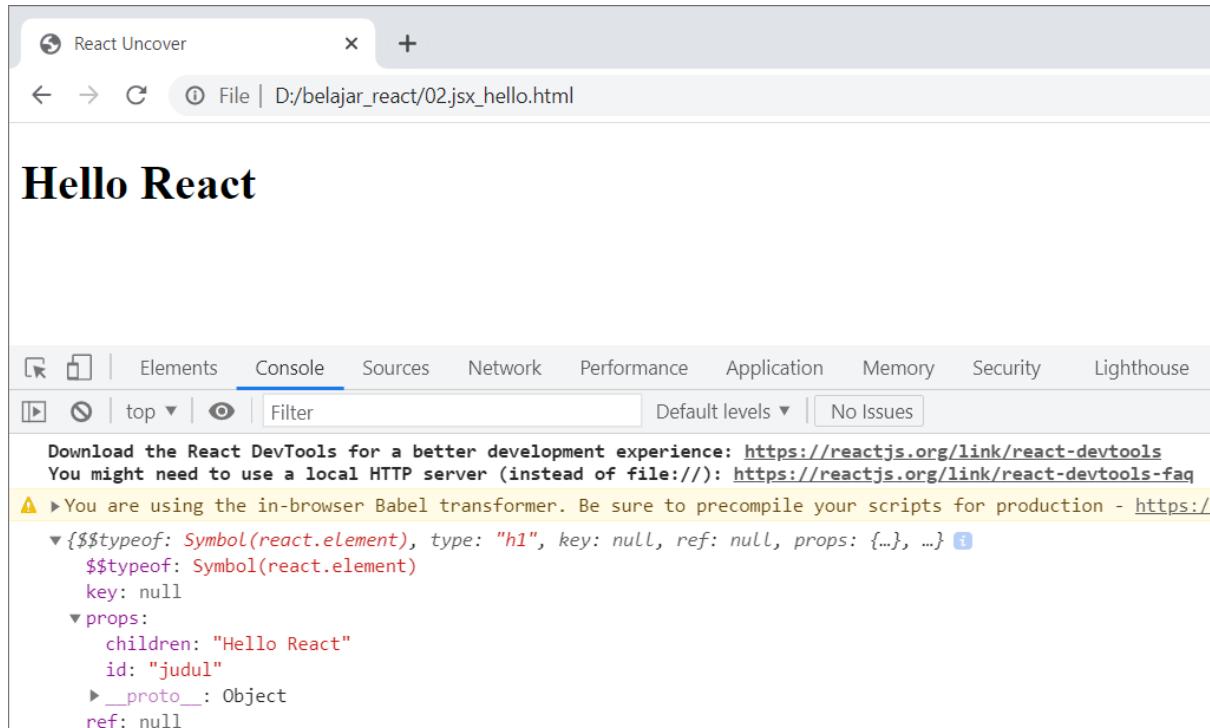
Di baris 9 juga terdapat satu tag `<div id="root">` yang saya siapkan sebagai nilai input untuk method `ReactDOM.createRoot().render()`. Disinilah `react element` akan ditampilkan.

Persiapan sudah selesai, mari kita coba:

02.jsx_hello.html

```
1 const myElement = <h1 id='judul'>Hello React</h1>;
2 console.log(myElement);
3
4 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Tempatkan kode di atas ke dalam tag `<script type="text/babel">`, dan berikut hasilnya:



Gambar: Hasil menjalankan JSX

Sukses!, kode JSX sudah tampil di web browser. Hasil perintah `console.log(myElement)` juga mengkonfirmasi bahwa konstanta `myElement` berisi sebuah `react element`.

Pesan warning "You are using the in-browser Babel transformer..." boleh diabaikan karena itu sekedar informasi tambahan dari babel.

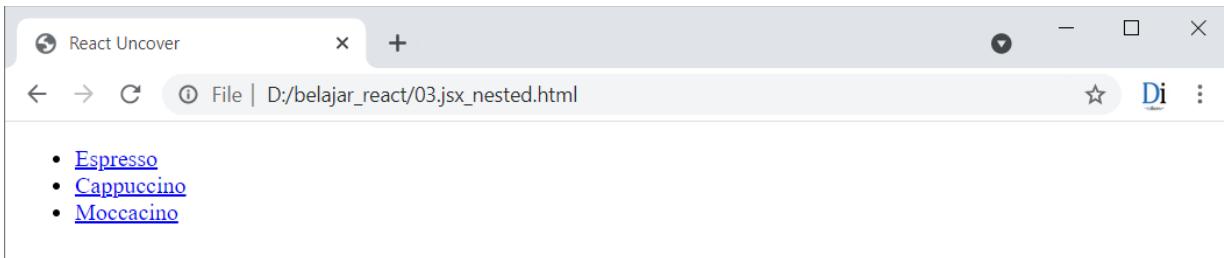
Percobaan kedua, kode berikut akan menampilkan menu kopi yang menjadi `exercise` di bab sebelumnya:

03.jsx_nested.html

```
1 const myElement = (
2   <ul>
3     <li><a href="https://id.wikipedia.org/wiki/Espreso">Espresso</a></li>
4     <li><a href="https://id.wikipedia.org/wiki/Kapucino">Cappuccino</a></li>
5     <li><a href="https://id.wikipedia.org/wiki/Moka">Moccacino</a></li>
6   </ul>
```

JSX (JavaScript XML)

```
7      )
8
9      ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



Gambar: Menampilkan nested element dengan JSX

Kode JSX ini jauh lebih singkat dan lebih mudah ditulis daripada menggunakan method `React.createElement()`.

Agar tidak perlu menulis lengkap kode HTML, sepanjang sisa bab saya hanya menampilkan kode JavaScript saja. Agar bisa berjalan, silahkan copy atau ketik ulang kode tersebut ke dalam tag `<script type="text/babel">`.

4.3. Aturan Dasar Penulisan JSX

Kode JSX sangat mirip seperti tag HTML biasa, tapi itu tetaplah perintah JavaScript yang memiliki aturan penulisan tersendiri.

Seperti contoh yang kita praktekkan, JSX bukanlah sebuah string sehingga jangan tambah tanda kutip di awal dan di akhir:

```
const myElement = "<h1 id='judul'>Hello React</h1>"; // salah
const myElement = <h1 id='judul'>Hello React</h1>; // benar
```

Jika kode JSX lebih dari 1 baris, tempatkan di dalam tanda kurung " () ":

```
// ini akan error
const myElement =
  <ul>
    <li><a href="https://id.wikipedia.org/wiki/Espresoa">Espresso</a></li>
    <li><a href="https://id.wikipedia.org/wiki/Kapucinoa">Cappuccino</a></li>
    <li><a href="https://id.wikipedia.org/wiki/Mokaa">Moccacino</a></li>
  </ul>

// benar
const myElement = (
  <ul>
    <li><a href="https://id.wikipedia.org/wiki/Espresoa">Espresso</a></li>
    <li><a href="https://id.wikipedia.org/wiki/Kapucinoa">Cappuccino</a></li>
    <li><a href="https://id.wikipedia.org/wiki/Mokaa">Moccacino</a></li>
  </ul>
)
```

JSX (JavaScript XML)

Namun jika kita hanya menulis 1 baris saja, tanda kurung ini boleh tidak ditulis.

JSX mengikuti aturan XML, sehingga setiap tag harus memiliki pasangan tag penutup. Khusus untuk tag yang berdiri sendiri, akhiri dengan tanda "/" yang sama seperti penulisan *self closing tag* di HTML:

```
const myElement = ;      // ini akan error
const myElement = ;     // benar
```

Blok JSX hanya bisa memiliki 1 tag terluar (1 *root element* atau 1 *top element*). Jika ada 2 tag yang saling berdampingan, hasilnya akan error:

```
// hanya bisa 1 root element, ini akan error
const myElement = (
  <p>Belajar JavaScript</p>
  <p>Belajar React</p>
);
```

Solusi yang paling mudah adalah dengan menambah satu tag `<div>` sebagai root atau "container" JSX (bisa juga tag lain) :

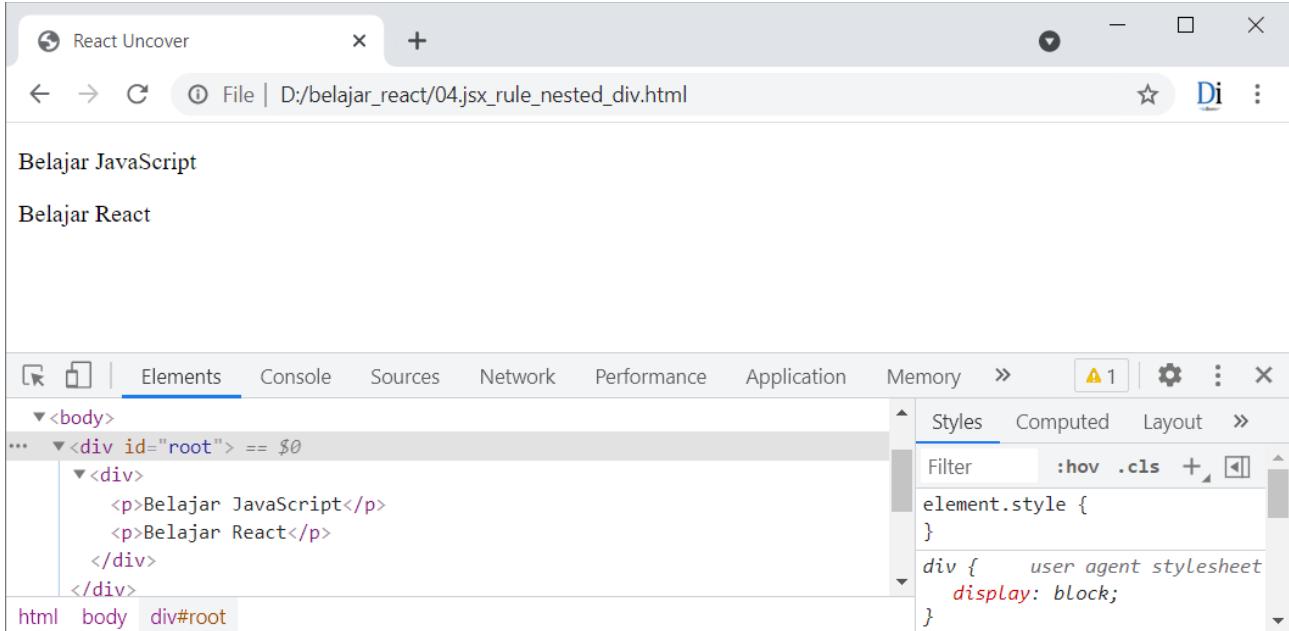
```
const myElement = (
  <div>
    <p>Belajar JavaScript</p>
    <p>Belajar React</p>
  </div>
);
```

Akan tetapi dalam beberapa situasi, tambahan tag `<div>` ini kadang tidak kita inginkan. Tag `<div>` tersebut bisa menambah "kedalaman" struktur DOM dan menjadi masalah untuk selector CSS. Berikut contoh kasusnya:

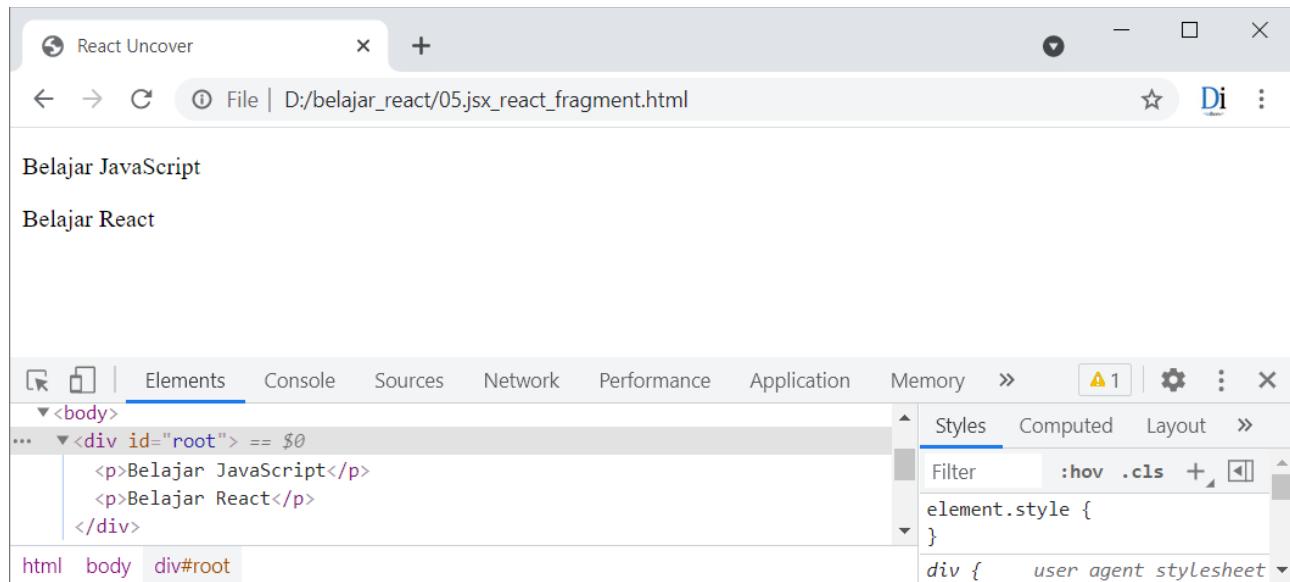
04.jsx_rule_nested_div.html

```
1 const myElement = (
2   <div>
3     <p>Belajar JavaScript</p>
4     <p>Belajar React</p>
5   </div>
6 );
7
8 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

JSX (JavaScript XML)



JSX (JavaScript XML)



Gambar: Hasil penggunaan <React.Fragment>

Sekarang di bawah tag `<div id="root">` sudah langsung berisi 2 buah tag `<p>`. Ini membuat struktur DOM menjadi lebih rapi tanpa perlu container `<div>` yang tidak kita perlukan.

Atribut HTML juga bisa ditulis seperti biasa ke dalam JSX. Pengecualianya adalah untuk atribut `class` dan `for`. Kedua kata ini menjadi keyword khusus di dalam JavaScript. JSX mengatasinya dengan mengganti penulisan atribut `class` menjadi `className`, serta atribut `for` menjadi `htmlFor`:

```
const myElement = <h1 class="judul">Belajar React</h1>;           // ini akan error
const myElement = <h1 className="judul">Belajar React</h1>;         // benar

const myElement = (
  <div>
    <label for="username">Username: </label>                         // ini akan error
    <input type="text" name="username" id="username" />
  </div>
);

const myElement = (
  <div>
    <label htmlFor="username">Username: </label>                     // benar
    <input type="text" name="username" id="username" />
  </div>
);
```

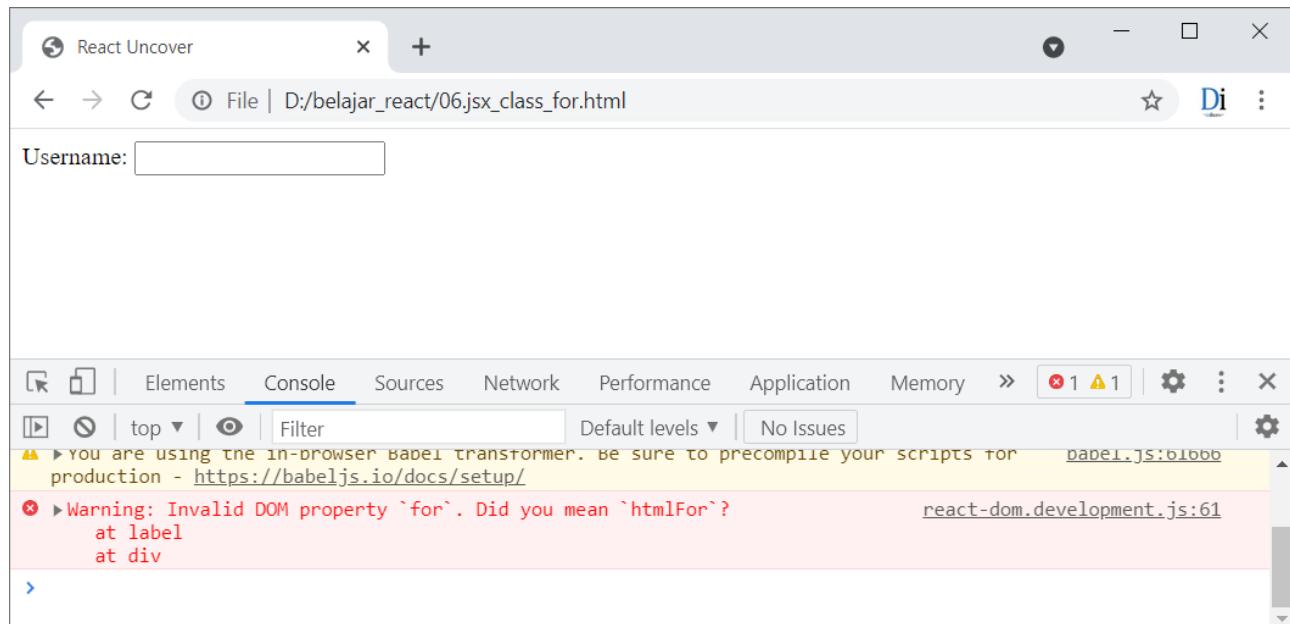
Berikut contoh error yang tampil:

06.jsx_class_for.html

```
1 const myElement = (
2   <div>
3     <label for="username">Username: </label>
4     <input type="text" name="username" id="username" />
```

JSX (JavaScript XML)

```
5      </div>
6  );
7
8 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



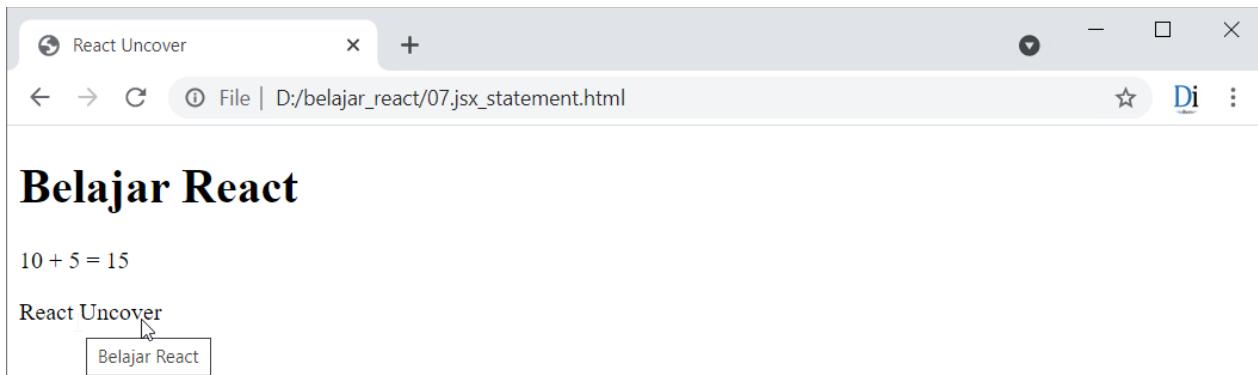
Gambar: Error karena penggunaan atribut for

Di baris 3 saya menulis atribut `for="username"` ke dalam tag `<label>`. Hasilnya, tampil pesan `Warning: Invalid DOM property 'for'. Did you mean 'htmlFor'?`. Untungnya, React memberi pesan warning yang sangat jelas bahwa atribut `for` harus diganti menjadi `htmlFor`.

4.4. Statement di Dalam JSX

JSX mengizinkan kita meng-eksekusi `expression` di antara kode-kode yang ada. Ini mirip seperti cara penulisan `expression` milik template string. Bedanya, di JSX kita cukup memakai tanda kurung kurawal tanpa tanda dollar " { } ":

```
07.jsx_statement.html
1 let myText="Belajar React";
2
3 const myElement = (
4   <div>
5     <h1>{myText}</h1>
6     <p>10 + 5 = {10 + 5}</p>
7     <span title={myText}>React Uncover</span>
8   </div>
9 );
10
11 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



Gambar: Hasil menjalankan statement di dalam JSX

Dalam kode ini saya mengakses variabel `myText` didalam tag `<h1>`, menjalankan perhitungan `10 + 5` di dalam tag `<p>`, serta menampilkan variabel `myText` sebagai nilai atribut `title` ke dalam tag ``.

Untuk penulisan atribut yang nilainya berasal dari `statement`, tidak perlu ditambah tanda kutip, cukup nilainya saja. JSX otomatis menambah tanda kutip saat ditampilkan ke dalam web browser:

```
<span title="{myText}">React Uncover</span>    // salah
<span title={myText}>React Uncover</span>    // benar
```

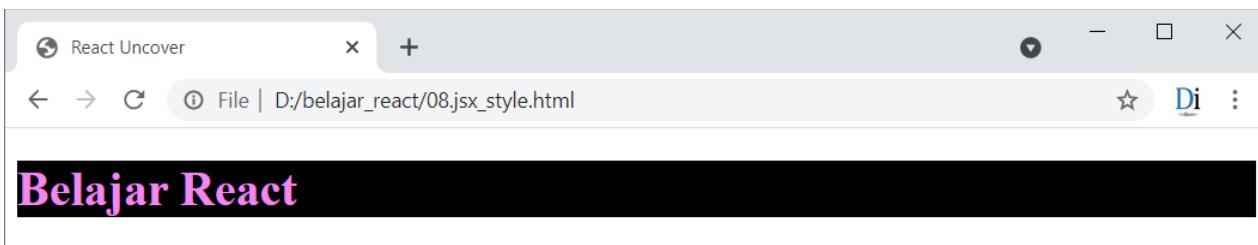
4.5. Penulisan Atribut style

Di dalam HTML, kita biasa memakai atribut `style` untuk menulis perintah CSS inline. JSX juga mendukung atribut ini, hanya saja nilainya harus ditulis dalam bentuk object. Property CSS yang memiliki tanda hubung " - " harus dikonversi ke dalam format camelCase, misalnya `background-color` menjadi `backgroundColor`.

Berikut contoh penulisan atribut `style` di JSX:

08.jsx_style.html

```
1 const myElement =
2 <h1 style={{ color: 'violet', backgroundColor: 'black' }}>Belajar React</h1>;
3
4 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



Gambar: Cara penulisan atribut style di JSX

Perhatikan cara penulisan nilai atribut `style` yang ditulis dengan 2 kali tanda kurung kurawal. Ini diperlukan karena tanda kurung pertama dipakai untuk penulisan *expression* di JSX, sedangkan tanda kurung kedua dipakai untuk membuat object JavaScript.

4.6. Percabangan Kondisi

Sepanjang pembuatan aplikasi, kita perlu membuat percabangan kondisi yang salah satunya menggunakan struktur `if else`. Sayangkan, perintah `if else` tidak bisa ditulis ke dalam JSX karena itu bukanlah JavaScript *expression*.

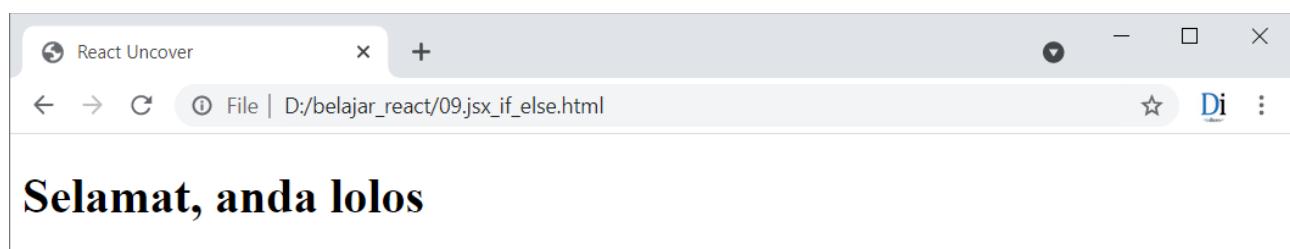
Namun dalam beberapa situasi, logika `if else` ini bisa ditempatkan di luar JSX. Berikut contoh yang dimaksud:

09.jsx_if_else.html

```

1  let nilai = 80;
2  let pesan;
3  if (nilai > 70) {
4      pesan = "Selamat, anda lolos";
5  }
6  else {
7      pesan = "Maaf, silahkan coba lagi";
8  }
9
10 const myElement = <h1>{pesan}</h1>;
11
12 ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```



Gambar: Hasil percabangan kondisi `if else` dengan JSX

Di baris 1 saya mengisi variabel `nilai` dengan angka 80, lalu membuat pemeriksaan kondisi `if else`. Jika nilai yang tersimpan lebih besar dari 70, isi variabel `pesan` dengan teks "Selamat, anda lolos", selain itu isi dengan string "Maaf, silahkan coba lagi".

Sesampainya di JSX pada baris 10, variabel `pesan` tinggal ditampilkan dengan perintah `<h1>{pesan}</h1>`. Inilah yang dimaksud dengan membuat percabangan `if else` di luar JSX.

Alternatif lain yang sangat sering dipakai dalam React adalah, membuat percabangan kondisi dengan *operator ternary* atau *conditional operator*, yakni operator dengan karakter "`: ? :`". Berikut contoh penggunaannya:

JSX (JavaScript XML)

10.jsx_ternary.html

```
1 let nilai = 80;
2
3 const myElement =
4 <h1>{(nilai > 70) ? "Selamat, anda lolos": "Maaf, silahkan coba lagi"}</h1>;
5
6 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Logika untuk kode ini sama seperti if else sebelumnya. Namun menjadi lebih singkat dengan memanfaatkan *conditional operator*.

Dalam kasus tertentu, kita hanya ingin menampilkan sebuah element jika suatu kondisi terpenuhi, selain itu jangan tampilkan apapun. Untuk logika ini, bisa juga dibuat menggunakan *short circuit* " && ":

11.jsx_short_circuit.html

```
1 let nilai = 60;
2
3 const myElement = (nilai > 70) && <h1>Selamat, anda lolos</h1>
4
5 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Saat dijalankan, konstanta `myElement` tidak berisi kode apapun karena kondisi `(nilai > 70)` sudah tidak terpenuhi.

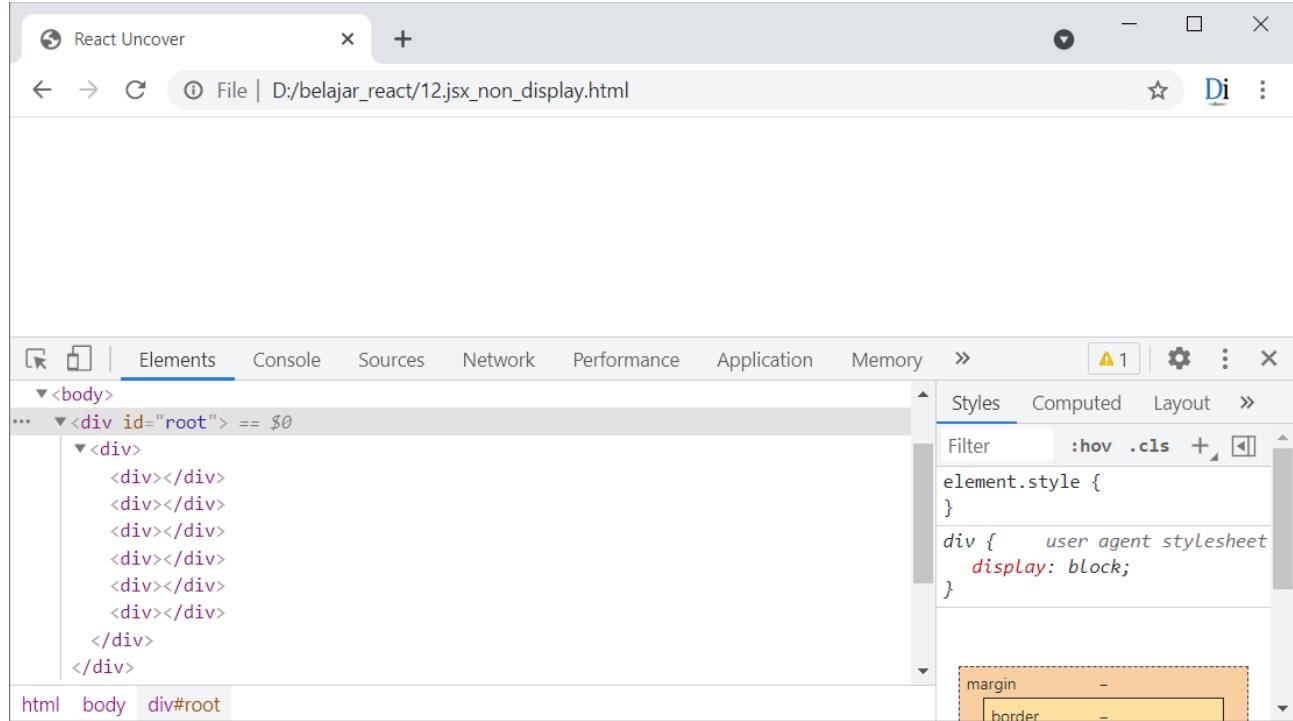
Penjelasan lebih lanjut tentang *conditional operator* dan teknik *short circuit* sudah kita bahas dalam bab 2 tentang "[JavaScript \(ES6\) untuk React](#)".

Sebagai tambahan, JSX tidak akan menampilkan nilai boolean `true`, `false`, `null` dan `undefined` ke dalam web browser:

12.jsx_non_display.html

```
1 const myElement = (
2   <div>
3     <div />
4     <div></div>
5     <div>{true}</div>
6     <div>{false}</div>
7     <div>{null}</div>
8     <div>{undefined}</div>
9   </div>
10 )
11
12 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Di dalam web browser, semua tag `<div>` di atas akan tampil sebagai tag kosong:



Gambar: JSX tidak me-render nilai true, false, null dan undefined

4.7. Array Processing

Dalam banyak situasi, kita perlu memproses nilai dalam bentuk array. Nilai ini bisa berasal dari database atau hasil pemanggilan API. Ketika ingin mengkonversi semua nilai ke dalam react element, method `Array.map()` bawaan JavaScript menjadi sangat praktis. Terlebih `Array.map()` sudah langsung memberikan hasil dalam bentuk array yang bisa diterima JSX.

Berikut contoh prakteknya:

13.jsx_map_1.html

```

1 let kopis = ["Espresso", "Cappuccino", "Moccacino"];
2
3 const myElement = (
4   <ul>
5     {kopis.map(kopi => <li>{kopi}</li>)}
6   </ul>
7 );
8
9 ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```

Di baris 1 terdapat variabel `kopis` yang berisi array 3 element. Kemudian di baris 5 array ini saya "buka" menggunakan method `kopis.map(kopi => {kopi})`. Method ini akan memproses setiap element array `kopis` untuk menghasilkan struktur berikut:

Espresso

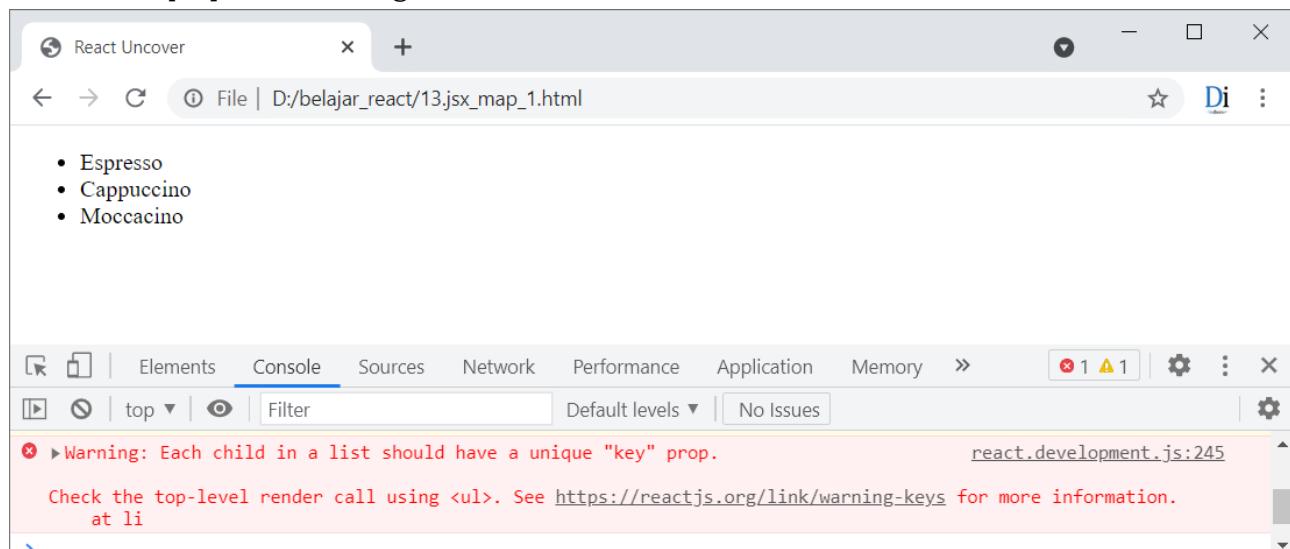
JSX (JavaScript XML)

```
<li>Cappuccino</li>
<li>Moccacino</li>
```

Dengan tambahan tag `` dan `` sebagai container konstanta `myElement`, itu sudah menjadi sebuah list element HTML yang lengkap.

Jika ragu dengan cara kerja method `Array.map()`, boleh dibaca lagi penjelasan di bab 2 tentang "[JavaScript \(ES6\) untuk React](#)". Ini sangat amat penting karena method `Array.map()` akan sering dipakai dalam React.

Tidak ada masalah dengan struktur DOM yang dihasilkan, akan tetapi begitu tab console dibuka, tampil pesan warning berikut:



Gambar: Pesan warning di tab console

Bisakah anda tebak apa maksud pesan ini? Betul, jika kita membuat element berulang, React butuh tambahan atribut **key**. Nilai key boleh bebas selama unik untuk setiap list (tidak memiliki nilai berulang).

Ketika pertama kali belajar React, saya langsung berpikir kalau nilai untuk atribut key ini bisa diambil dari argument kedua method `Array.map()`. Di bab 2 pernah kita bahas bahwa `Array.map()` memiliki callback yang argumentnya bisa men-generate urutan element (index array). Nilai inilah yang bisa dipakai untuk atribut key:

14.jsx_map_solution_1.html

```
1      let kopis = ["Espresso", "Cappuccino", "Moccacino"];
2
3      const myElement = (
4          <ul>
5              {kopis.map((kopi,i) => <li key={i}>{kopi}</li>)}
6          </ul>
7      );
```

```

8
9     ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```

Di baris 5 terdapat tambahan atribut `key={i}`, sehingga akan men-generate kode berikut:

```

<li key="0">Espresso</li>
<li key="1">Cappuccino</li>
<li key="2">Moccacino</li>

```

Hasilnya, pesan warning dari React sudah tidak ada lagi.

Setelah penelusuran lebih lanjut, cara men-generate key seperti di atas ternyata bisa menjadi bug dan tidak disarankan, terutama jika kita ingin memodifikasi list secara dinamis (dihapus, ditambah, atau diurutkan ulang). Namun jika hanya menampilkan data statis seperti contoh kita, itu tidak masalah.

Pada saat menggunakan data asli yang berasal dari database, biasanya nilai yang didapat sudah memiliki `id` atau kolom unik lain. Kolom yang bertindak sebagai *primary key* paling pas menjadi nilai untuk atribut `key`.

Penjelasan kenapa menggunakan index sebagai nilai `key` bisa menjadi bug agak rumit, jika tertarik bisa membaca di artikel berikut: [Index as a key is an anti-pattern¹⁴](#) dan [Understanding unique keys for array children in React.js¹⁵](#).

Untuk list yang tidak terlalu banyak dan nilainya sudah unik, itu pun bisa dipakai:

```

15.jsx_map_solution_2.html
1 let kopis = ["Espresso", "Cappuccino", "Moccacino"];
2
3 const myElement = (
4   <ul>
5     {kopis.map((kopi) => <li key={kopi}>{kopi}</li>)}
6   </ul>
7 );
8
9 ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```

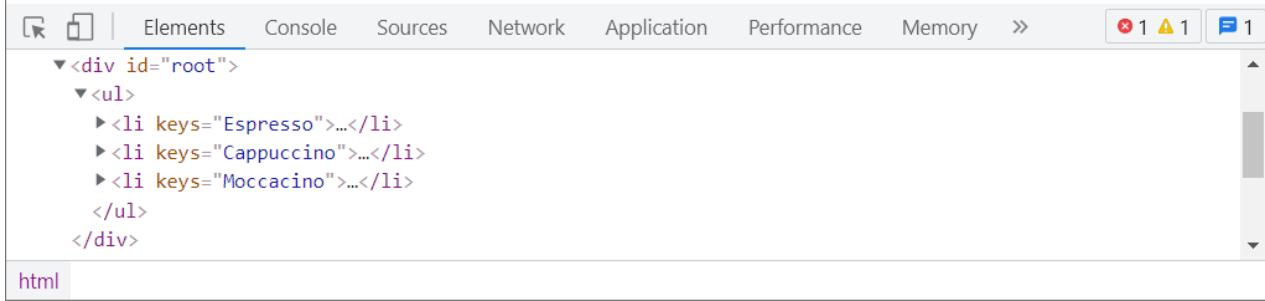
Sebagai nilai `key`, saya ambil dari isi variabel `kopi`. Ini bisa dipakai karena tidak ada nilai berulang di dalam array `kopis`.

Yang agak unik, ketika tab inspect element dibuka, atribut `key` tidak bisa ditemukan. Alasannya karena react hanya menggunakan atribut `key` untuk internal saja, sehingga tidak akan perlu ditampilkan ke dalam struktur DOM.

Jika kita ingin memeriksa nilai ini, tukar nama atribut `key` untuk sementara menjadi nama lain, misalnya `keys`, lalu lihat di web browser:

14. <https://robinpokorny.com/blog/index-as-a-key-is-an-anti-pattern>

15 <https://stackoverflow.com/questions/28329382/understanding-unique-keys-for-array-children-in-react-js>



Gambar: Melihat nilai property keys

Cara lain untuk melihat nilai key ini juga bisa dengan menginstall extension [React Developer Tools](#)¹⁶.

#Exercise

Kita sudah berhasil menampilkan list yang berasal dari array. Sekarang saatnya coba dengan data yang lebih kompleks. Untuk exercise kali ini saya ingin membuat tabel HTML yang datanya berasal dari array berikut:

```

1 const mahasiswa = [
2   {
3     nama: "Eka",
4     umur: 19,
5     jurusan: "Teknik Informatika"
6   },
7   {
8     nama: "Lisa",
9     umur: 18,
10    jurusan: "Sistem Informasi"
11  },
12  {
13    nama: "Rudi",
14    umur: 19,
15    jurusan: "Teknik Elektro"
16  }
17 ];

```

Array `mahasiswa` terdiri dari 3 element yang berbentuk object. Silahkan coba rancang kode JSX untuk menampilkan data-data ini ke dalam bentuk tabel HTML.

Berikut tampilan akhir di web browser:

16. <https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi>

Nama	Umur	Jurusan
Eka	19	Teknik Informatika
Lisa	18	Sistem Informasi
Rudi	19	Teknik Elektro

Gambar: Memproses array menjadi tabel menggunakan JSX

Tips: Sebagai pengingat, berikut contoh struktur kode HTML untuk membuat tabel:

```

1 <table>
2   <thead>
3     <tr>
4       <th>Nama</th><th>Umur</th><th>Jurusan</th>
5     </tr>
6   </thead>
7   <tbody>
8     <tr>
9       <td>...</td><td>...</td><td>...</td>
10    </tr>
11    <tr>
12      <td>...</td><td>...</td><td>...</td>
13    </tr>
14    ...
15  </tbody>
16 </table>
```

Silahkan coba sebentar untuk membuatnya.

Selamat juga anda berhasil! Tapi juga tidak masalah jika belum bisa karena kodanya memang sedikit kompleks. Pemahaman tentang cara kerja method `Array.map()` sangat penting di sini, termasuk juga cara mengakses property object.

Berikut kode yang saya gunakan:

16.jsx_map_exercise.html

```

1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>React Uncover</title>
```

```

7   <style>
8     table, th, td {
9       border: 1px solid black;
10    }
11    th, td {
12      padding: 0.4rem 0.6rem;
13    }
14  </style>
15 </head>
16 <body>
17
18  <div id="root"></div>
19
20  <script src="js/react.development.js"></script>
21  <script src="js/react-dom.development.js"></script>
22  <script src="js/babel.js"></script>
23  <script type="text/babel">
24
25  const mahasiswa = [
26    {
27      nama: "Eka",
28      umur: 19,
29      jurusan: "Teknik Informatika"
30    },
31    {
32      nama: "Lisa",
33      umur: 18,
34      jurusan: "Sistem Informasi"
35    },
36    {
37      nama: "Rudi",
38      umur: 19,
39      jurusan: "Teknik Elektro"
40    }
41  ];
42
43  const myElement = (
44    <table>
45      <thead>
46        <tr>
47          <th>Nama</th><th>Umur</th><th>Jurusan</th>
48        </tr>
49      </thead>
50      <tbody>
51        {mahasiswa.map(
52          (mahasiswa,i) =>
53            <tr key={i}>
54              <td>{mahasiswa.nama}</td>
55              <td>{mahasiswa.umur}</td>
56              <td>{mahasiswa.jurusan}</td>
57            </tr>
58          )}
59      </tbody>

```

```

60      </table>
61  );
62
63  ReactDOM.createRoot(document.getElementById('root')).render(myElement);
64
65  </script>
66 </body>
67 </html>

```

Agar tampilan tabel lebih menarik, saya menambah sedikit kode CSS di baris 7-14. Ini tidak wajib, sekedar merapikan tampilan saja.

Inti dari kode kita ada di baris 51 – 58, dimana saya menggunakan method `mahasiswa.map()` untuk memproses semua element array `mahasiswa`. Di dalam function *callback*, variabel `mahasiswa` merujuk ke object yang sedang di proses saat ini, sehingga kita bisa mengakses `mahasiswa.nama`, `mahasiswa.umur` dan `mahasiswa.jurusan`.

4.8. JavaScript Event

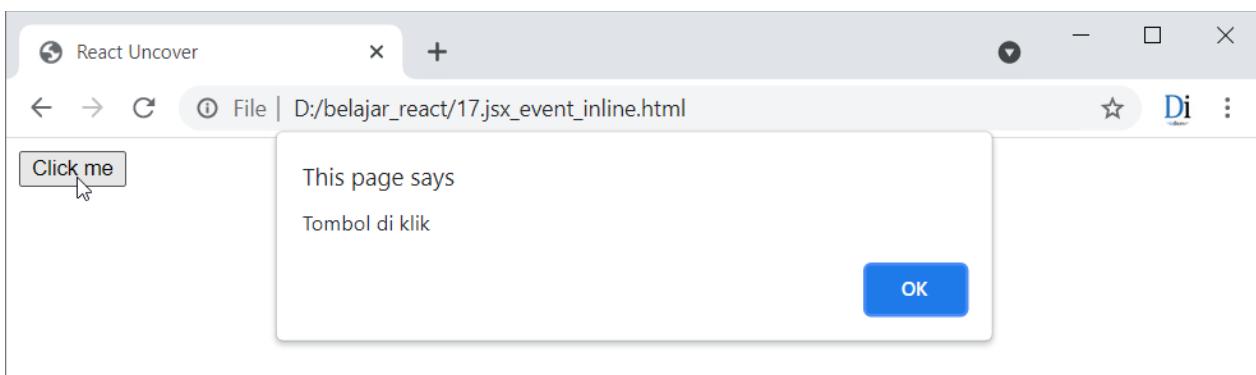
Kita juga bisa menambah event ke dalam JSX, berikut contoh penulisannya:

```

17.jsx_event_inline.html

1  const myElement = (
2    <button onClick={() => alert("Tombol di klik")}>
3      Click me
4    </button>
5  );
6
7  ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```



Gambar: Jendela alert tampil saat tombol di klik

Di baris 2 terdapat atribut `onClick` yang diisi dengan function. Penulisan seperti ini sangat mirip seperti *inline event* di JavaScript biasa.

Yang berbeda ada di penamaan atribut event. Jika di JavaScript ditulis dengan huruf kecil seperti `onclick`, maka di JSX harus ditulis dalam format camelCase, yakni `onClick`. Begitu juga

penamaan event lain seperti `onSubmit`, `onChange`, dst.

Agar lebih rapi, function untuk event juga bisa ditulis terpisah:

18.jsx_event_function.html

```
1 const onClickHandler = () => alert("Tombol di klik");
2
3 const myElement = (
4   <button onClick={onClickHandler}>
5     Click me
6   </button>
7 );
8
9 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Kali ini saya membuat fungsi `onClickHandler()` di baris 1 yang akan berjalan saat event `onClick` terjadi.

Perhatikan cara penulisan atribut `onClick` di baris 4. Kita cukup menulis nama fungsi tanpa tanda kurung, yakni `onClick={onClickHandler}`, dan bukan `onClick={onClickHandler()}`. Penulisan kedua akan langsung menjalankan fungsi `onClickHandler()` saat halaman di load, bukan saat event `onClick` terjadi.

React juga tidak menyarankan membuat event menggunakan `addEventListener()`, tapi cukup langsung tulis sebagai atribut saja.

Materi lebih detail tentang *event* di React akan kembali kita bahas dalam bab tersendiri.

Itulah perkenalan kita dengan **JSX** (JavaScript XML). React tidak mewajibkan JSX untuk membuat *react element*, tapi seperti yang terlihat penulisannya jauh lebih mudah dibandingkan `React.createElement()`. Sampai akhir bab nanti, JSX akan terus terpakai.

Berikutnya kita akan masuk ke materi tentang **React Component**.

5. React Component

Dalam perancangan aplikasi web, React memilih pendekatan berbeda dibandingkan JavaScript biasa (*native JavaScript* atau *vanilla JavaScript*).

Di JavaScript, umumnya kita diminta memisahkan kode HTML dengan JavaScript. Sedapat mungkin tidak ada kode JS yang tercampur dengan tag HTML. Tujuan dari pemisahan ini adalah agar kode menjadi lebih rapi dan mudah dikelola.

Akan tetapi konsep ini kadang tidak selalu memudahkan. Jika kodenya cukup panjang, perlu waktu untuk memahami kode JS ini untuk bagian HTML yang mana karena keduanya saling terpisah.

Di React, HTML dan JavaScript dibuat saling bergabung. Materi JSX yang baru saja kita pelajari merupakan implementasi dari konsep ini. Tapi bukankah itu akan membuat kodenya jadi berantakan? Betul, tapi React mengatur kode program menjadi komponen-komponen terpisah. Dalam bab ini kita akan bahas lebih jauh tentang **React Component**.

5.1. Pengertian React Component

Di React, **component** adalah bagian kecil dari kode program yang bertanggung jawab untuk menampilkan bagian tertentu. Daripada memecah kode program antara HTML dengan JavaScript, React memilih untuk memecah kode program berdasarkan tampilannya.

Agar lebih mudah ditulis (dan juga dibaca), istilah *component* akan saya terjemahkan sebagai **komponen** saja.

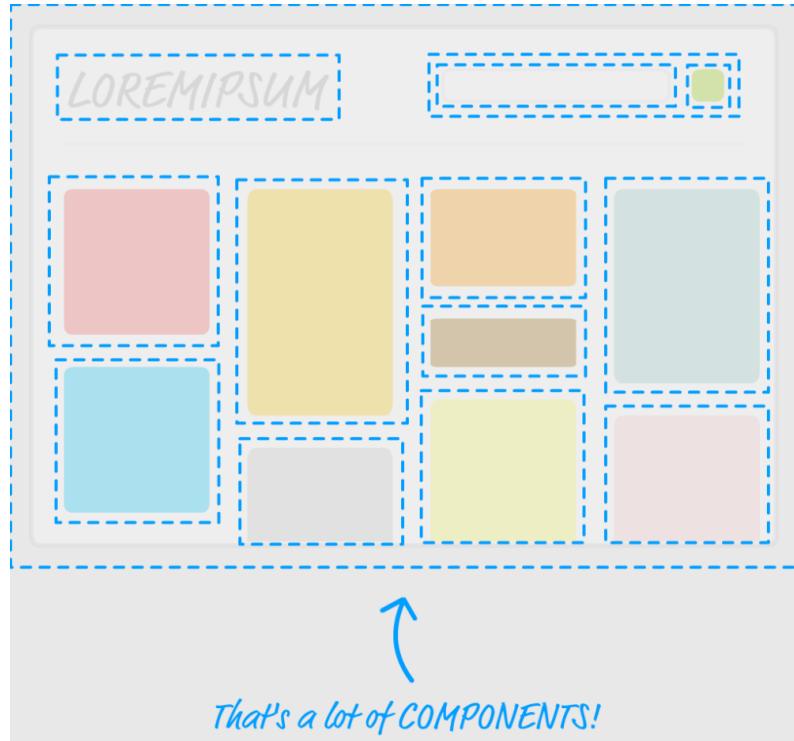
Implementasi sederhana dari komponen adalah bagian-bagian web seperti header, footer, form pencarian, dst. Setiap bagian ini bisa dibuat menjadi **React component**, yakni *header component*, *footer component*, dst.

Dalam setiap komponen bisa saja saling tercampur antara kode HTML, CSS, dan JavaScript, namun itu tidak masalah. Dengan membagi kode program berdasarkan tampilan, pengelolaan aplikasi menjadi lebih efektif. Ketika kita ingin menambah fitur baru untuk header, tinggal buka *header component* dan semua kode program sudah ada di situ.

Konsep ideal dari *React component* adalah menampilkan **satu bagian** web saja. Jika komponen itu mengelola banyak bagian sekaligus, disarankan untuk dipecah menjadi komponen -

komponen yang lebih kecil.

Untuk aplikasi besar, biasanya ada komponen yang bertugas hanya menampung komponen lain (sebagai container). Misalnya *header component* bisa bertindak sebagai container bagi *menu component*, *logo component*, dan *search component*.



Gambar: Ilustrasi React component (sumber gambar: kirupa.com)

Tugas **React component** sebenarnya mirip seperti function di dalam programming, yakni bagian kecil dari kode program yang memecahkan suatu masalah.

5.2. Class Component vs Functional Components

Saat ini terdapat 2 cara untuk membuat komponen di React, yakni berbentuk *class* (disebut sebagai **class component**), atau berbentuk *function* (disebut sebagai **functional component**).

Sejak awal React di rilis, *class component* merupakan satu-satunya cara untuk membuat komponen. Kemudian barulah *functional component* diperkenalkan pada React versi 0.14 (bisa juga disebut React versi 14).

Dalam banyak hal, *functional component* menawarkan konsep yang lebih praktis dan kode kita menjadi lebih ringkas. Akan tetapi ketika awal diperkenalkan, *functional component* masih punya kelemahan besar, yakni tidak mendukung pembuatan **state**. Karena itulah *functional component* sempat disebut sebagai **stateless component** (komponen yang tidak punya state).

Pada awal 2019, diperkenalkanlah fitur **hook** di React versi 16.8. Fitur ini membuat *functional*

component bisa memiliki state, dan sejak saat itu tidak lagi dikenal istilah *stateless component* karena baik class component maupun functional component sama-sama mendukung state.

State dan **Hook** adalah konsep inti React yang akan kita bahas dalam bab terpisah. Untuk sementara bisa dipahami kalau keduanya merupakan fitur penting di React dan terpaksa saya bahas untuk menjelaskan sejarah *class component* dan *functional components*.

Dengan adanya hook, functional component dianggap sebagai cara penulisan komponen yang lebih modern, sedangkan class component dianggap sebagai versi "jadul".

Jika anda bertanya di forum atau group programming terkait kode React yang ditulis dalam bentuk class component, sangat mungkin ada yang menyuruh untuk beralih ke functional component karena dianggap lebih baru.

Akan tetapi di dokumentasi React ditulis bahwa class component tetap di dukung dan tidak ada rencana untuk menghapusnya. Di katakan juga bahwa dalam kode program Facebook sendiri masih banyak kode React dalam bentuk class component dan tim Facebook tidak berencana mengubahnya.

Dua variasi cara pembuatan komponen ini mendatangkan masalah tersendiri bagi kita yang ingin belajar React. Meskipun functional component lebih disarankan, akan tetapi di luar sana masih banyak tutorial lama yang menggunakan class component. Termasuk dokumentasi React juga masih banyak ditulis dalam bentuk class component (belum ada versi functional).

Tidak mustahil juga suatu saat anda ditugaskan untuk meng-handle project React yang ditulis dalam bentuk class component.

Karena alasan-alasan inilah dalam buku **React Uncover** ini saya memutuskan untuk membahas keduanya sekaligus, yakni *class component* dan juga *functional component*.

Bahasan kita memang akan jadi lebih panjang, tapi saya berharap bisa memberikan konsep dasar React yang lengkap. Terlebih ada satu materi yang lebih mudah dibahas dalam bentuk class component, yakni terkait *React Lifecycle*.

Mulai dari bab ini dan beberapa bab ke depan, saya akan bahas penulisan dalam bentuk class component terlebih dahulu, lalu baru melihat versi functional component-nya.

5.3. Membuat Class Component

Kita akan masuk ke pembuatan **class component** terlebih dahulu. Berikut contohnya:

01.class_component.html

```
1  <!DOCTYPE html>
2  <html lang="id">
```

```
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>React Uncover</title>
7 </head>
8
9
10 <body>
11   <div id="root"></div>
12
13   <script src="js/react.development.js"></script>
14   <script src="js/react-dom.development.js"></script>
15   <script src="js/babel.js"></script>
16   <script type="text/babel">
17
18     class JudulReact extends React.Component {
19       render() {
20         return <h1>Sedang Belajar React</h1>;
21       }
22     }
23
24     ReactDOM.createRoot(document.getElementById('root')).render(<JudulReact />);
25
26   </script>
27 </body>
28
29 </html>
```

Sama seperti bahasan materi JSX, semua kode program harus ditulis ke dalam tag `<script type="text/babel">`

Untuk membuat class component, kita harus men-extends class bawaan React yang bernama `React.Component`. Dengan kata lain, semua komponen adalah turunan dari `React.Component`.

Di sini saya membuat class bernama `JudulReact`. Nama class boleh bebas, tapi harus ditulis dengan **PascalCase**, yakni huruf besar di setiap kata termasuk kata pertama.

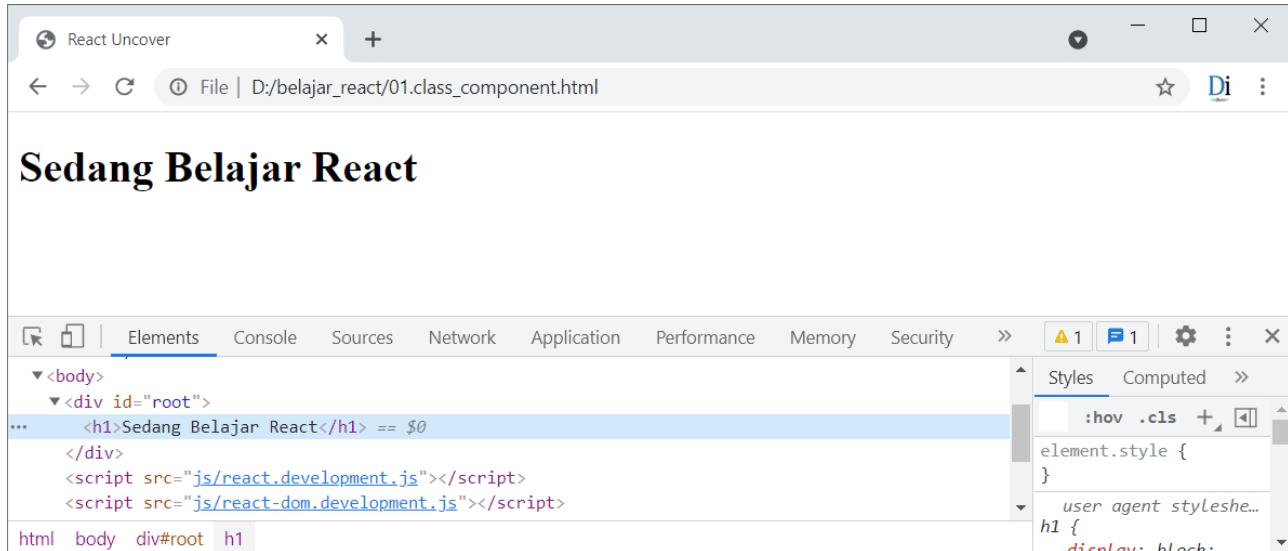
Sepanjang class component, nantinya bisa ditulis berbagai method lain, akan tetapi terdapat 1 method yang harus selalu ada, yakni `render()`. Sesuai namanya, method `render()` berfungsi untuk me-render atau menampilkan sesuatu.

Dalam contoh di atas, method `render()` saya buat untuk me-return kode JSX, yakni `<h1> Sedang Belajar React</h1>`. Setiap kali component `JudulReact` dipanggil, kode JSX ini akan di-render.

Akan tetapi hasil render sebuah component hanya sampai di virtual DOM React. Untuk bisa terlihat di web browser, kita tetap butuh bantuan method `ReactDOM.createRoot().render()` seperti yang ada di baris 24.

Perhatikan juga argument pertama dari method `ReactDOM.createRoot().render()`, itulah cara

memanggil React component, yakni dengan menulis nama component sebagai tag <JudulReact />. Berikut tampilannya di web browser:



Gambar: Tampilan component <JudulReact /> di web browser

Kode JSX <h1>Sedang Belajar React</h1> akan tampil ke dalam tag <div id="root"> sebagai hasil dari pemanggilan class <JudulReact />.

Multiple Component

Dalam sebuah aplikasi React, biasanya terdapat lebih dari satu komponen yang tampil bersamaan. Akan tetapi method ReactDOM.createRoot().render() hanya bisa menerima satu komponen saja. Salah satu solusi adalah dengan mengakses beberapa komponen dari sebuah variabel JSX:

02.class_component_multiple.html

```
1  class JudulReact extends React.Component {  
2      render() {  
3          return <h1>Sedang Belajar React</h1>;  
4      }  
5  }  
6  
7  class JudulJavaScript extends React.Component {  
8      render() {  
9          return <h1>Sedang Belajar JavaScript</h1>;  
10     }  
11 }  
12  
13 const MyElement = (  
14     <div>  
15         <JudulReact/>  
16         <hr/>  
17         <JudulJavaScript/>  
18     </div>
```

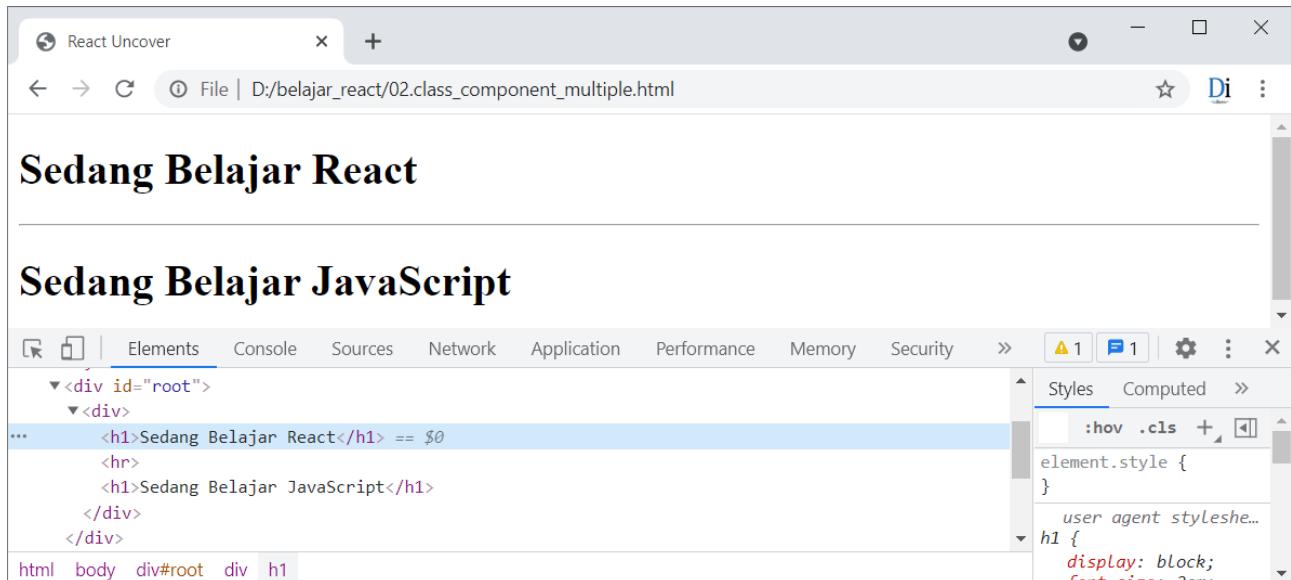
React Component

```
19 );
20
21 ReactDOM.createRoot(document.getElementById('root')).render(MyElement);
```

Di awal kode program terdapat 2 buah class component, yakni `JudulReact` di baris 1-5 dan `JudulJavaScript` di baris 7-11. Struktur kedua komponen sangat mirip, hanya beda pada bagian teks tag `<h1>` saja.

Di baris 13, konstanta `MyElement` saya isi dengan kode JSX yang mengakses `<JudulReact/>` dan `<JudulJavaScript/>`. Dapat kembali dilihat kalau React component bisa dianggap sebagai *custom tag* HTML, atau tag HTML yang kita definisikan sendiri.

Pada saat di proses oleh method `ReactDOM.createRoot().render()`, tag `<JudulReact/>` dan `<JudulJavaScript/>` ini akan berganti dengan struktur JSX yang di-return oleh kedua komponen. Berikut hasilnya:



Gambar: Mengakses component `<JudulReact/>` dan `<JudulJavaScript/>`

Nested Component

Selain mengakses komponen dari variabel biasa, kita juga bisa mengaksesnya dari method `render()` milik komponen lain. Penulisan seperti ini sangat sering dipakai dalam kode React:

03.class_component_nested.html

```
1 class JudulReact extends React.Component {
2   render() {
3     return <h1>Sedang Belajar React</h1>;
4   }
5 }
6
7 class JudulJavaScript extends React.Component {
8   render() {
9     return <h1>Sedang Belajar JavaScript</h1>;
```

```

10     }
11 }
12
13 class MyComponent extends React.Component {
14   render() {
15     return (
16       <div>
17         <JudulReact />
18         <hr />
19         <JudulJavaScript />
20       </div>
21     )
22   }
23 };
24
25 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);

```

Isi class component JudulReact dan JudulJavaScript masih sama seperti sebelumnya. Namun sekarang di baris 13 – 23 saya membuat komponen ketiga bernama MyComponent. Isinya berupa kode JSX yang mengakses kedua komponen tersebut.

Komponen `<MyComponent />` inilah yang menjadi nilai input untuk method `ReactDOM.createRoot().render()`. Hasil akhir dari kode ini akan sama persis seperti contoh kita sebelumnya.

Contoh ini juga memperlihatkan bentuk hubungan antar komponen React. Komponen MyComponent bertindak sebagai *parent* (induk) dari JudulReact dan JudulJavaScript. Begitu juga dengan komponen JudulReact dan JudulJavaScript yang saling menjadi *sibling* (saudara) dan sebagai *child* (anak) dari MyComponent.

5.4. Membuat Functional Component

Di awal bab sempat kita bahas bahwa mulai dari React 14 hadir cara pembuatan komponen menggunakan function sebagai alternatif dari class component. Berikut contoh penulisannya:

04.function_component.html

```

1  function JudulReact() {
2    return <h1>Sedang Belajar React</h1>;
3  }
4
5  function JudulJavaScript() {
6    return <h1>Sedang Belajar JavaScript</h1>;
7  }
8
9  function MyComponent() {
10   return (
11     <div>
12       <JudulReact />
13       <hr />
14       <JudulJavaScript />
15     </div>

```

React Component

```
16    )
17 }
18
19 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

Bisa terlihat kalau penulisan functional component lebih singkat daripada class component. Di sini kita tidak perlu meng-extends `React.Component`, tapi cukup buat fungsi JavaScript biasa yang me-return JSX, dan itu sudah menjadi sebuah component.

Agar lebih modern, penulisan function bisa dibuat dengan format *arrow notation*:

05.function_component_arrow.html

```
1 const JudulReact = () => <h1>Sedang Belajar React</h1>;
2 const JudulJavaScript = () => <h1>Sedang Belajar JavaScript</h1>;
3
4 const MyComponent = () => {
5   return (
6     <div>
7       <JudulReact />
8       <hr />
9       <JudulJavaScript />
10    </div>
11  )
12}
13
14 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

Dengan arrow notation, penulisan function menjadi lebih ringkas lagi. Component `JudulReact` dan `JudulJavaScript` masing-masing hanya butuh 1 baris saja.

5.5. Gabungan Class dan Functional Component

Meskipun jarang dilakukan, tapi kita bisa mengakses class component dari functional component dan sebaliknya:

06.class_and_function_component.html

```
1 class JudulReact extends React.Component {
2   render() {
3     return <h1>Sedang Belajar React</h1>;
4   }
5 }
6
7 const JudulJavaScript = () => <h1>Sedang Belajar JavaScript</h1>;
8
9 const MyComponent = () => {
10   return (
11     <div>
12       <JudulReact />
13       <hr />
```

```
14      <JudulJavaScript />
15    </div>
16  )
17 }
18
19 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

Kali ini komponen JudulReact saya buat menggunakan class, sedangkan JudulJavaScript dibuat menggunakan function. Keduanya bisa saling mengakses satu sama lain karena hanya beda cara penulisan saja.

Dalam praktek aslinya, setiap komponen disarankan ada dalam file terpisah, bukan digabung dalam satu file seperti contoh di atas. Namun agar lebih ringkas dan mudah dipraktekkan, saya akan tunda pemisahan file komponen sampai kita masuk ke materi tentang react create app di pertengahan buku nanti.

5.6. Menambah Property dan Method di Class Component

Class component pada dasarnya sama seperti class JavaScript biasa. Kita bisa menambah property (variabel) serta method (function) ke dalamnya. Property dan method ini diperlukan sebagai alat bantu ketika memproses data.

Berikut contoh penambahan variabel ke dalam class component:

```
07.class_component_variabel_1.html

1  class MyComponent extends React.Component {
2    render() {
3      let judulReact = "Belajar React";
4      return (
5        <React.Fragment>
6          <h1>{judulReact}</h1>
7          <hr />
8          <p>Stt... lagi serius {judulReact.toLocaleLowerCase()}</p>
9        </React.Fragment>
10     )
11   }
12 }
13
14 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

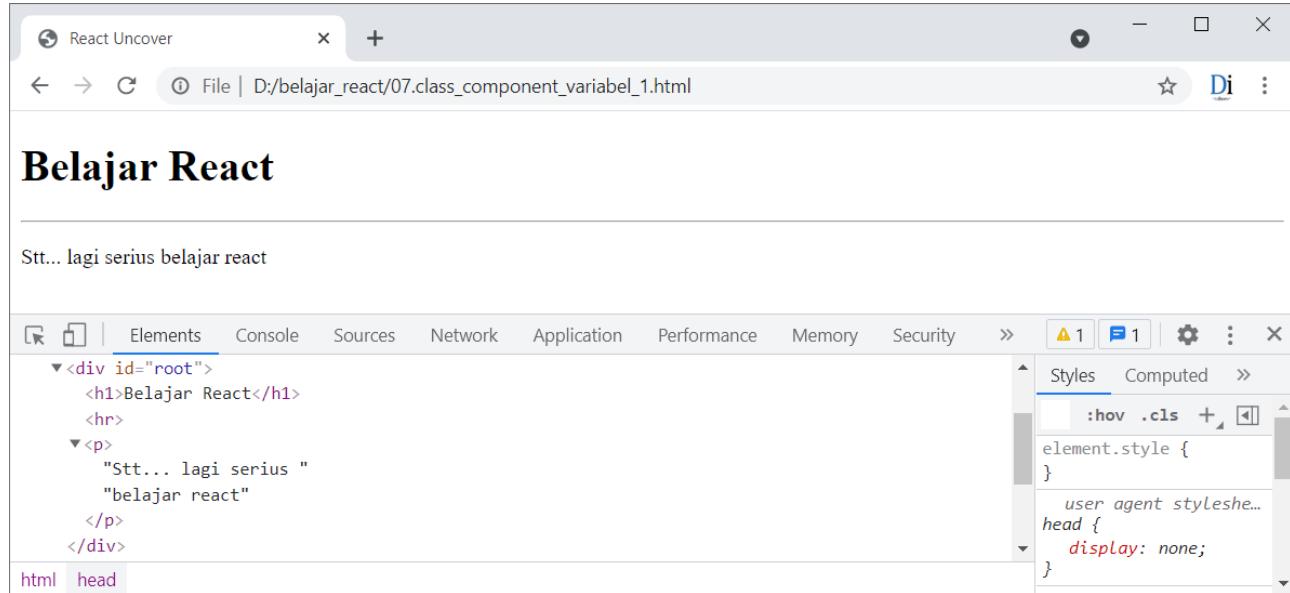
Kali ini class MyComponent sedikit lebih kompleks. Di baris 3 terdapat variabel judulReact yang berisi string "Belajar React". Ini bisa ditulis karena posisinya ada di dalam method render(), atau dengan kata lain variabel judulReact bertindak sebagai *internal variable* milik method render().

Variabel judulReact selanjutnya saya akses di dua tempat, yakni di baris 6 dan di baris 8. Di baris 8 ini terdapat tambahan method `toLocaleLowerCase()` bawaan JavaScript untuk

React Component

mengubah isi string menjadi lower case (huruf kecil semua).

Berikut hasil dari kode di atas:



Gambar: Penggunaan variabel di dalam class component

Lokasi penulisan variabel lain yang bisa kita tulis adalah di luar class:

08.class_component_variabel_2.html

```
1 let judulReact = "Belajar React";
2
3 class MyComponent extends React.Component {
4   render() {
5     return (
6       <React.Fragment>
7         <h1>{judulReact}</h1>
8         <hr />
9         <p>Stt... lagi serius {judulReact.toLocaleLowerCase()}</p>
10      </React.Fragment>
11    )
12  }
13 }
14
15 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

Kali ini variabel `judulReact` saya pindahkan ke baris 1. Ini membuat `judulReact` berperan sebagai *global variable* yang bisa diakses dari komponen apa saja (dalam file yang sama).

Jika kita ingin variabel itu hanya bisa diakses dari dalam satu class saja, maka tempatkan ke dalam class:

09.class_component_property.html

```
1 class MyComponent extends React.Component {
2   judulReact = "Belajar React";
```

```

3   render() {
4     return (
5       <React.Fragment>
6         <h1>{this.judulReact}</h1>
7         <hr />
8         <p>Stt... lagi serius {this.judulReact.toLocaleLowerCase()}</p>
9       </React.Fragment>
10    )
11  }
12 }
13
14 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);

```

Sekarang variabel `judulReact` pindah ke baris 2. Ini membuatnya berperan sebagai *property* dari class `MyComponent`. Di JavaScript, penulisan property tidak butuh keyword `let` atau `var`, cukup nama property saja.

Namun karena `judulReact` sudah menjadi property, kita harus menambah keyword `this` saat mengaksesnya di baris 6 dan 8.

Posisi penulisan property lain yang lebih sering di pakai adalah di dalam constructor:

```

10.class_component_constructor.html
1  class MyComponent extends React.Component {
2   constructor() {
3     super();
4     this.judulReact = "Belajar React";
5   }
6
7   render() {
8     return (
9       <React.Fragment>
10      <h1>{this.judulReact}</h1>
11      <hr />
12      <p>Stt... lagi serius {this.judulReact.toLocaleLowerCase()}</p>
13      </React.Fragment>
14    )
15  }
16 }
17
18 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);

```

Di baris 2-5 saya menambah sebuah constructor ke dalam class `MyComponent`. Ketika kita menambah constructor di class component, di baris awal **harus ada** perintah `super()`. Perintah ini diperlukan agar constructor milik `React.Component` (yang menjadi *parent class*) juga ikut dijalankan. Setelah itu barulah saya membuat property `this.judulReact`.

Itulah beberapa tempat sebagai lokasi pendeklarasikan variabel di dalam class component. Setiap posisi punya keunggulan masing-masing yang akan kita bahas secara bertahap sepanjang buku ini.

Selain property, kita juga bisa menambahkan method bantu ke dalam class component. Berikut contohnya:

```
11.class_component_method.html

1  class MyComponent extends React.Component {
2      constructor() {
3          super();
4          this.judulReact = "Belajar React";
5      }
6
7      judulHurufKecil() {
8          return this.judulReact.toLocaleLowerCase();
9      }
10
11     render() {
12         return (
13             <React.Fragment>
14                 <h1>{this.judulReact}</h1>
15                 <hr />
16                 <p>Stt... lagi serius {this.judulHurufKecil()}</p>
17             </React.Fragment>
18         )
19     }
20 }
21
22 ReactDOM.createRoot(document.getElementById('root')).render(<MyComponent />);
```

Di baris 7-9 saya mendeklarasikan method `judulHurufKecil()`. Method ini mengembalikan versi huruf kecil dari property `judulReact`. Inilah yang selanjutnya di panggil di baris 16. Teknik seperti ini cukup sering kita pakai jika butuh pemrosesan data yang panjang.

Materi tentang `property`, `method`, `this`, `constructor()` dan `super()` bukanlah bagian dari React, tapi konsep dasar **OOP JavaScript**. Jika anda mengalami kesulitan memahaminya, bisa coba cari tutorial atau buku yang membahas pemrograman object di JavaScript.

5.7. Menampilkan Data dengan Class Component

Setelah memahami konsep dasar class component, kita bisa masuk ke praktek yang lebih *real*, yakni menampilkan data.

Di React, umumnya data ini berasal dari database atau hasil pemanggilan API. Data tersebut kebanyakan berbentuk array atau object. React component bertugas untuk memproses dan menampilkannya ke dalam web browser.

Mari bahas dengan contoh kode berikut:

12.class_component_menampilkan_data.html

```

1  const mahasiswa =
2  {
3    id: "01",
4    nama: "Eka",
5    jurusan: "Teknik Informatika",
6    pasFoto: "people1.jpg"
7  };
8
9  class Mahasiswa extends React.Component {
10    render() {
11      return (
12        <figure>
13          <img src={"img/" + mahasiswa.pasFoto} alt={mahasiswa.nama} />
14          <figcaption>{mahasiswa.nama} ({mahasiswa.jurusan})</figcaption>
15        </figure>
16      )
17    }
18  }
19
20 ReactDOM.createRoot(document.getElementById('root')).render(<Mahasiswa />);

```

Di baris 1 – 7 terdapat konstanta `mahasiswa` yang berisi object dengan 4 property. Di bab 2 pernah kita bahas kalau mengakses property dari sebuah object cukup dengan menambah tanda titik, misalnya `mahasiswa.id` atau `mahasiswa.nama`.

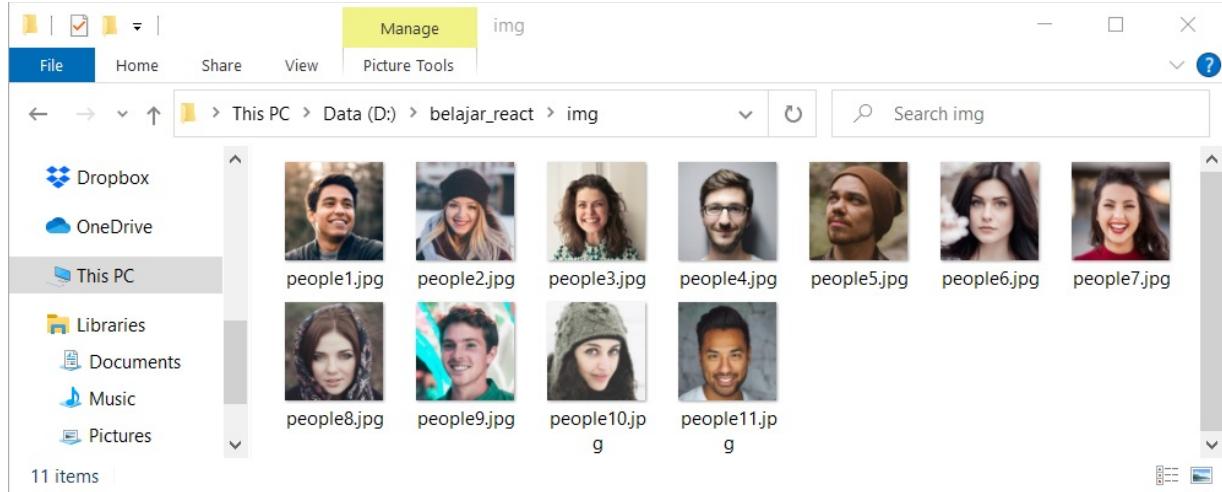
Kemudian di baris 9 – 18 saya membuat class component bernama `Mahasiswa`. Method `render()` dari class ini mengembalikan struktur JSX yang mengakses setiap property dari object `mahasiswa`.

Kita boleh bebas merancang struktur JSX. Dalam contoh ini saya membuat tag `<figure>` yang di dalamnya terdapat tag `` dan `<figcaption>`.

Jika diperhatikan, atribut `src` milik tag `` tertulis sebagai `"img/" + mahasiswa.pasFoto`. Ini akan di-render oleh React menjadi `"img/people1.jpg"`. Agar gambar ini bisa tampil, file `people1.jpg` harus tersedia di folder `img`.

Sebagai gambar dummy untuk kode ini (dan juga kode-kode kita selanjutnya), silahkan buat folder `img` di dalam `D:\belajar_react\`, lalu isi dengan gambar sembarang dengan nama `people1.jpg`, `people1.jpg`, dst hingga `people11.jpg`. Atau cara yang lebih praktis, bisa copy folder `img` dari file program `belajar_react.zip` yang ada di Google Drive.

React Component



Gambar: Sample gambar yang saya siapkan di folder D:\belajar_react\img

Setelah folder `img` tersedia, berikut hasil dari kode React kita:

A screenshot of a browser window titled 'React Uncover'. The main content area shows a portrait of a young man with the caption 'Eka (Teknik Informatika)'. Below the browser window is the browser's developer tools, specifically the Elements tab. The DOM tree shows a `<div id="root">` element containing a `<figure>` element with an `` child and a `<figcaption>` child. The `` tag has a `src="img/people1.jpg"` attribute and an `alt="Eka"` attribute. The `<figcaption>` contains the text "Eka" on the first line, followed by a line break, then " (" on the second line, "Teknik Informatika" on the third line, and ")" on the fourth line. To the right of the DOM tree is the Styles panel, which shows a CSS rule for the `body` element: `display: block; margin: 8px;`. The bottom tabs of the developer tools show 'html' and 'body'.

Gambar: Tampilan component Mahasiswa

Inilah cara menampilkan data object menggunakan React component. Silahkan dipelajari sebentar terkait bagaimana kode JSX yang ada di method `render()` di proses menjadi struktur HTML seperti tampilan di atas.

Lanjut dengan contoh lain, biasanya data yang kita proses tidak hanya satu saja, tapi banyak. Data ini umumnya tersusun dalam bentuk array seperti konstanta `mahasiswa`s berikut:

```
1 const mahasiswa = [  
2   {
```

```

3     id:"01",
4     nama: "Eka",
5     jurusan: "Teknik Informatika",
6     pasFoto: "people1.jpg"
7   },
8   {
9     id:"02",
10    nama: "Lisa",
11    jurusan: "Sistem Informasi",
12    pasFoto: "people2.jpg"
13  },
14  {
15    id:"03",
16    nama: "Rudi",
17    jurusan: "Teknik Elektro",
18    pasFoto: "people4.jpg"
19  }
20 ];

```

Tugas kita adalah, membuat React component untuk menampilkan semua data ini di web browser. Struktur JSX yang diperlukan hampir sama seperti contoh sebelumnya, hanya saja kali ini kita butuh perulangan untuk menampilkan data.

Berikut kode yang bisa dipakai:

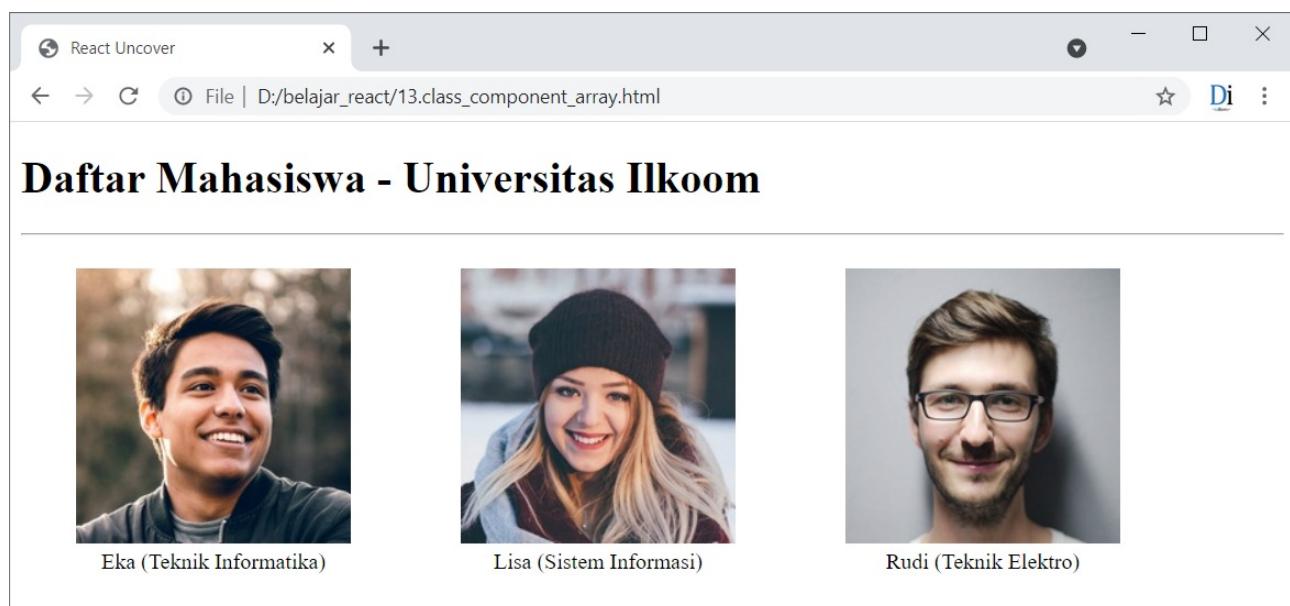
13.class_component_array.html

```

1 const mahasiswa = [
2   {
3     id:"01",
4     nama: "Eka",
5     jurusan: "Teknik Informatika",
6     pasFoto: "people1.jpg"
7   },
8   ...
9   ...
10 ];
11
12 class DaftarMahasiswa extends React.Component {
13   render() {
14     return (
15       <React.Fragment>
16         <h1>Daftar Mahasiswa - Universitas Ilkoom</h1>
17         <hr />
18         <section style={{display: "flex", textAlign: "center"}}>
19           {
20             mahasiswa.map( (mahasiswa) =>
21               <figure key={mahasiswa.id}>
22                 <img src={"img/" + mahasiswa.pasFoto} alt={mahasiswa.nama} />
23                 <figcaption>{mahasiswa.nama} ({mahasiswa.jurusan})</figcaption>
24               </figure>
25             )
26           }
27         </section>

```

```
28      </React.Fragment>
29    )
30  }
31 }
32
33 ReactDOM.createRoot(document.getElementById('root')).render(<DaftarMahasiswa />);
```



Gambar: Menampilkan banyak data mahasiswa

Untuk menyingkat kode program, saya tidak menulis utuh isi array `mahasiswas`. Data ini ada di kode sebelumnya yang berisi 3 object.

Method `render()` di komponen `DaftarMahasiswa` dibuka dengan tag `<React.Fragment>`. Ini sekedar untuk menghindari penambahan tag `<div>` sebagai container JSX. Selanjutnya di bagian awal saya membuat tag `<h1>` sebagai judul halaman web. Ini berfungsi untuk mempercantik tampilan saja.

Tag `<section>` di baris 18 berisi atribut `style` agar tampilan box mahasiswa berjejer dari kiri ke kanan. Ini didapat dari perpaduan kode CSS `display:"flex"` dan `textAlign:"center"`.

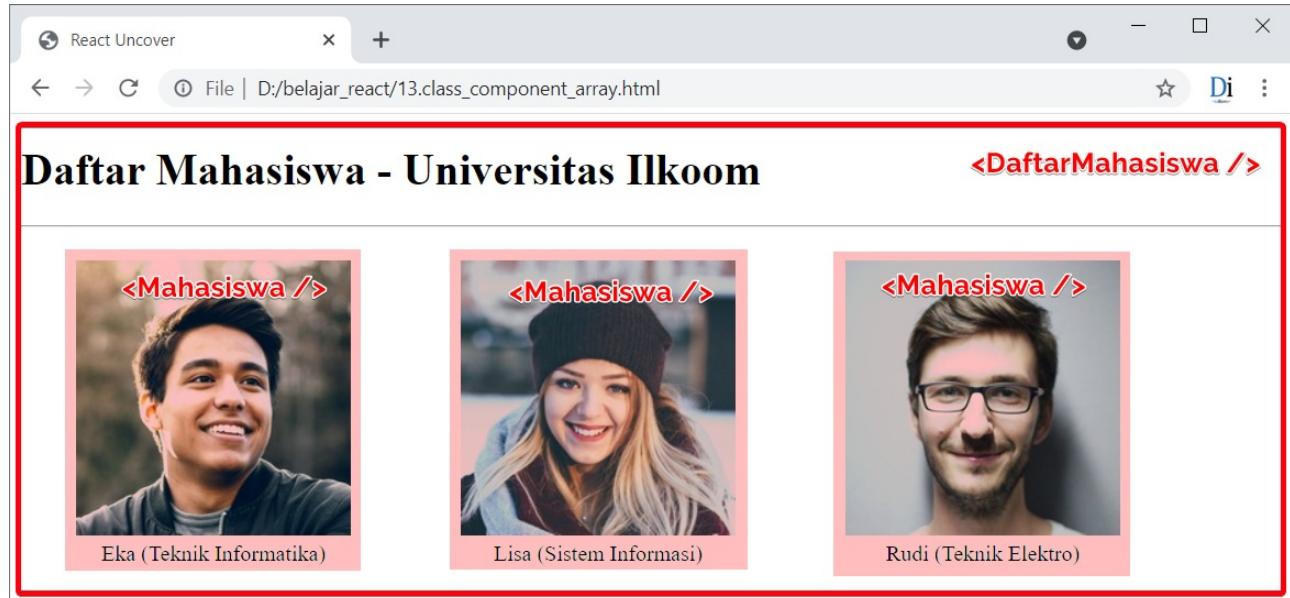
Kode utama kita ada di antara baris 20 – 25. Method `mahasiswas.map()` akan diproses sejumlah element array yang ada di konstanta `mahasiswas`. Dalam setiap perulangan, variabel `mahasiswa` diisi dengan object yang saat ini sedang diakses.

Struktur JSX yang saya pakai kurang lebih sama seperti contoh kita sebelumnya. Yang berbeda ada di tambahan atribut `key={mahasiswa.id}` ke dalam tag `<figure>`. Ini diperlukan agar React bisa membedakan setiap tag `<figure>`.

5.8. Memecah Class Component

Sampai di sini kita sudah berhasil menampilkan satu dan banyak data mahasiswa

menggunakan React component. Tantangan selanjutnya, saya ingin "mengeluarkan" kode JSX yang menampilkan data mahasiswa menjadi komponen terpisah. Ilustrasinya sebagai berikut:



Gambar: Membagi class component

Dalam contoh sebelum ini, saya hanya memakai 1 komponen saja, yakni `DaftarMahasiswa`. Untuk kode yang sederhana, ini tidak jadi masalah. Akan tetapi jika struktur JSX dari setiap mahasiswa menjadi lebih kompleks, disarankan kode tersebut di pecah menjadi komponen sendiri, misalnya komponen `Mahasiswa`.

Dengan demikian, komponen `DaftarMahasiswa` nantinya menjadi container yang memanggil komponen `Mahasiswa`.

Berikut contoh kode untuk pemisahan ini:

```
14.class_component_mahasiswa_nested.html

1  class Mahasiswa extends React.Component {
2      render() {
3          return (
4              <figure>
5                  
6                  <figcaption>Rudi Irawan (Teknik Informatika)</figcaption>
7              </figure>
8          )
9      }
10 }
11
12 class DaftarMahasiswa extends React.Component {
13     render() {
14         return (
15             <React.Fragment>
16                 <h1>Daftar Mahasiswa - Teknik Informatika</h1>
17                 <hr />
```

```

18      <Mahasiswa />
19    </React.Fragment>
20  )
21 }
22 }
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<DaftarMahasiswa />);

```

Agar lebih sederhana, di sini saya belum menggunakan data terpisah. Komponen `Mahasiswa` di baris 1 – 10 berisi data yang langsung ditulis ke dalam JSX. Komponen ini selanjutnya diakses sebagai tag `<Mahasiswa />` dari dalam komponen `DaftarMahasiswa` di baris 18.

Inilah yang dimaksud dengan "memecah komponen", yakni membagi kode JSX di sebuah komponen besar ke dalam komponen-komponen yang lebih kecil.

Sebagai latihan, bisakah anda modifikasi contoh kita sebelumnya (array `mahasiswas`) ke dalam bentuk komponen terpisah? Silahkan coba sebentar. Sedikit tips, perulangan `mahasiswas.map()` bisa ikut dibawa ke dalam komponen `Mahasiswa`.

Selamat jika berhasil! Berikut kode yang saya gunakan:

15.class_component_mahasiswas_nested.html

```

1 const mahasiswas = [
2   {
3     id: "01",
4     nama: "Eka",
5     jurusan: "Teknik Informatika",
6     pasFoto: "people1.jpg"
7   },
8   {
9     id: "02",
10    nama: "Lisa",
11    jurusan: "Sistem Informasi",
12    pasFoto: "people2.jpg"
13  },
14  {
15    id: "03",
16    nama: "Rudi",
17    jurusan: "Teknik Elektro",
18    pasFoto: "people4.jpg"
19  }
20];
21
22 class Mahasiswa extends React.Component {
23   render() {
24     return (
25       mahasiswas.map((mahasiswa) =>
26         <figure key={mahasiswa.id}>
27           <img src={"img/" + mahasiswa.pasFoto} alt={mahasiswa.nama} />
28           <figcaption>{mahasiswa.nama} ({mahasiswa.jurusan})</figcaption>
29         </figure>
30     )
31   }
32 }

```

```

31     )
32   }
33 }
34
35 class DaftarMahasiswa extends React.Component {
36   render() {
37     return (
38       <React.Fragment>
39         <h1>Daftar Mahasiswa - Teknik Informatika</h1>
40         <hr />
41         <section style={{ display: "flex", textAlign: "center" }}>
42           <Mahasiswa />
43         </section>
44       </React.Fragment>
45     )
46   }
47 }
48 ReactDOM.createRoot(document.getElementById('root')).render(<DaftarMahasiswa />);

```

Yang saya lakukan hanya men-cut isi method `mahasiswas.map()` untuk di pindah ke komponen `Mahasiswa`, kemudian mengganti kode yang di pindah tersebut dengan pemanggilan tag `<Mahasiswa />` di baris 42.

Perlu tidaknya memecah sebuah komponen, lebih ke alur logika kita saja. Jika rasanya komponen itu sudah terlalu besar atau melakukan banyak hal sekaligus, bisa dipertimbangkan untuk dipisah.

5.9. Menampilkan Data dengan Functional Component

Semua hal yang bisa kita lakukan dengan class component, juga bisa dibuat menggunakan functional component. Caranya, cukup ganti penulisan class menjadi function.

Berikut kode program untuk menampilkan array `mahasiswas` yang kali ini saya tulis menggunakan functional component:

16.function_component_mahasiswas.html

```

1 const mahasiswas = [
2   {
3     id:"01",
4     nama: "Eka",
5     jurusan: "Teknik Informatika",
6     pasFoto: "people1.jpg"
7   },
8   ...
9   ...
10 ];
11
12 const Mahasiswa = () => {
13   return (

```

```

14     mahasiswa.map( (mahasiswa) =>
15       <figure key={mahasiswa.id}>
16         <img src={"img/" + mahasiswa.pasFoto} alt={mahasiswa.nama} />
17         <figcaption>{mahasiswa.nama} ({mahasiswa.jurusan})</figcaption>
18       </figure>
19     )
20   )
21 }
22
23 const DaftarMahasiswa = () => {
24   return (
25     <React.Fragment>
26       <h1>Daftar Mahasiswa - Teknik Informatika</h1>
27       <hr />
28       <section style={{display: "flex", textAlign: "center"}}>
29         <Mahasiswa />
30       </section>
31     </React.Fragment>
32   )
33 }
34
35 ReactDOM.createRoot(document.getElementById('root')).render(<DaftarMahasiswa />);

```

Secara garis besar tidak banyak perbedaan. Kode JSX yang ada sama persis seperti contoh di class component. Yang perlu di modifikasi hanya di perubahan class menjadi function saja.

Di dalam kode JSX, penulisan React component sangat mirip seperti tag HTML biasa. Untuk membedakannya, jika tag tersebut diawali huruf besar maka itu adalah komponen, jika tidak maka akan dianggap sebagai tag HTML.

Dalam bab ini kita telah membahas **React Component** yang hampir bisa dipastikan selalu ada di setiap aplikasi React. Karena sejarahnya, komponen ini hadir dalam 2 "rasa", yakni **class component** dan **functional component**. Meskipun *functional component* dianggap lebih modern, pengetahuan seputar *class component* tetap sangat bermanfaat.

Ketika komponen saling berinteraksi, kadang perlu mengirim data untuk bisa diproses oleh komponen lain. Pengiriman data ini bisa dilakukan dengan **props**, yang akan kita bahas dalam bab selanjutnya.

6. React Props

Materi kali ini masih berhubungan erat dengan React component. Kita akan bahas apa itu **props** beserta fungsinya.

6.1. Pengertian Props

Props adalah sebuah fitur di React untuk mengirim data antar komponen. Biasanya data ini berasal dari *parent component* untuk dikirim ke *child component*. Props merupakan singkatan dari **property** (akhiran 's' ditambah untuk membuat versi plural, dimana *props* = banyak *prop*).

Data yang ingin kita kirim sebagai *props*, ditulis ke dalam tag JSX dari komponen tersebut. Penulisannya mirip seperti atribut HTML biasa.

Sebagai contoh, jika terdapat kode JSX <Belajar materi="React"/>, maka *materi="React"* ini akan sampai ke komponen *Belajar* sebagai *props*.

Terdapat sedikit perbedaan mengenai cara akses props antara *class component* dengan *functional component*, oleh karena itu akan kita bahas secara terpisah.

6.2. Props di Class Component

Sebelumnya sempat disinggung kalau nilai yang dikirim sebagai *props*, ditulis dalam bentuk atribut HTML seperti <Belajar materi="React"/>. Sesampainya di class component *Belajar*, nilai "React" akan ditampung sebagai *class property* dan bisa diakses menggunakan perintah *this.props.materi*.

Berikut contoh penggunaannya:

01.props_1.html

```
1  class Belajar extends React.Component {  
2      render() {  
3          return <h1>Belajar {this.props.materi} </h1>;  
4      }  
5  }  
6  
7  const myElement = <Belajar materi="React" />  
8  
9  ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

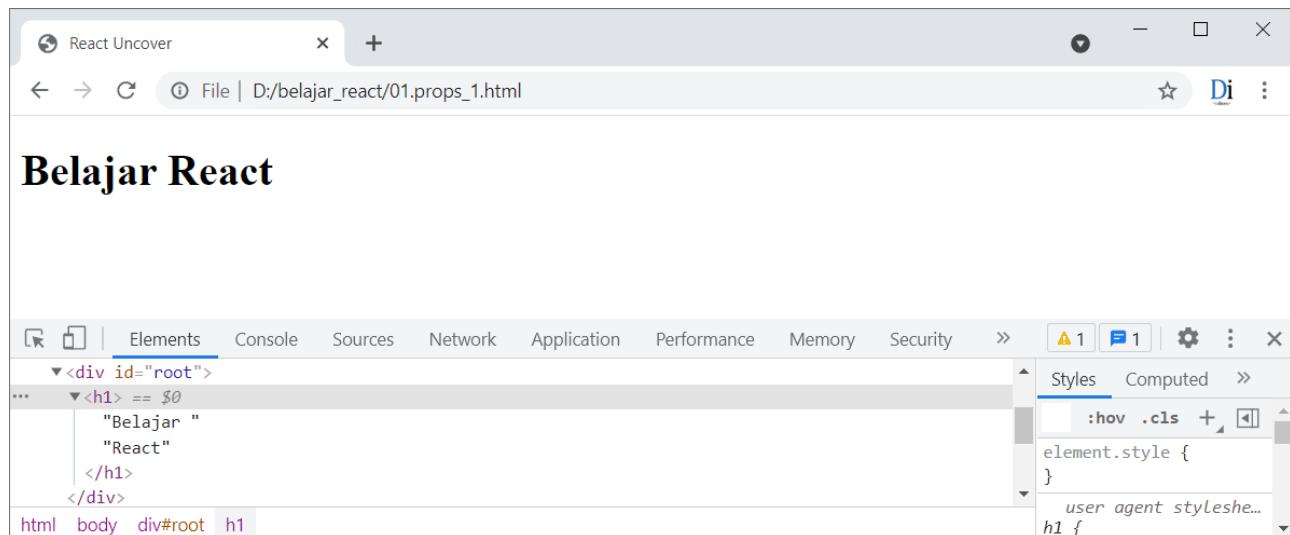
Agar mudah dipahami, kita mulai dari baris paling bawah terlebih dahulu.

Di baris 9, terlihat kalau method `ReactDOM.createRoot().render()` ingin menampilkan isi variabel / konstanta `myElement`. Maka selanjutnya kita cari apa isi variabel tersebut.

Isinya berupa kode JSX `<Belajar materi="React"/>` seperti yang ada di baris 7. Karena tag ini diawali dengan huruf besar, maka itu merujuk ke sebuah komponen, sedangkan atribut `materi="React"` akan dikirim sebagai `props` ke dalam komponen `Belajar`.

Pendefinisian komponen `Belajar` ada di baris 1 – 5. Method `render()` langsung mengembalikan kode JSX `<h1>Belajar {this.props.materi} </h1>`. Kode `this.props.materi` akan diganti oleh React menjadi string "React", karena string itulah yang dikirim dari atribut `materi`.

Berikut hasilnya di web browser:



Gambar: Menampilkan isi props "materi"

Kita bebas mengirim berapa pun jumlah `props` ke dalam sebuah komponen, dan itu semua bisa diakses dari `this.props.<nama_atribut>`:

02.props_2.html

```

1  class Belajar extends React.Component {
2      render() {
3          console.log(this.props);
4          return <h1>Belajar {this.props.materi} </h1>;
5      }
6  }
7
8  const myElement = <Belajar materi="React" id="01" className="judul" />
9
10 ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```

Di baris 8 saya menambah 2 lagi penulisan atribut ke dalam tag `<Belajar/>`, yakni `id="01"` dan `className = "judul"`. Sesampainya di komponen `Belajar`, semua atribut tersimpan ke dalam

property `this.props`.

Sebagai bukti, saya menambah perintah `console.log(this.props)` di dalam method `render()` dan berikut hasilnya:



Gambar: Hasil dari perintah `console.log(this.props)`

Terlihat bahwa semua atribut sudah ada di dalam object `this.props`. Jika kita ingin mengakses string 'judul', itu bisa didapat dari `this.props.className`.

Bagi yang sudah paham tentang perbedaan argument dan parameter function, maka atribut yang kita kirim saat penulisan tag JSX ini bisa diibaratkan sebagai *argument* (nilai input untuk function).

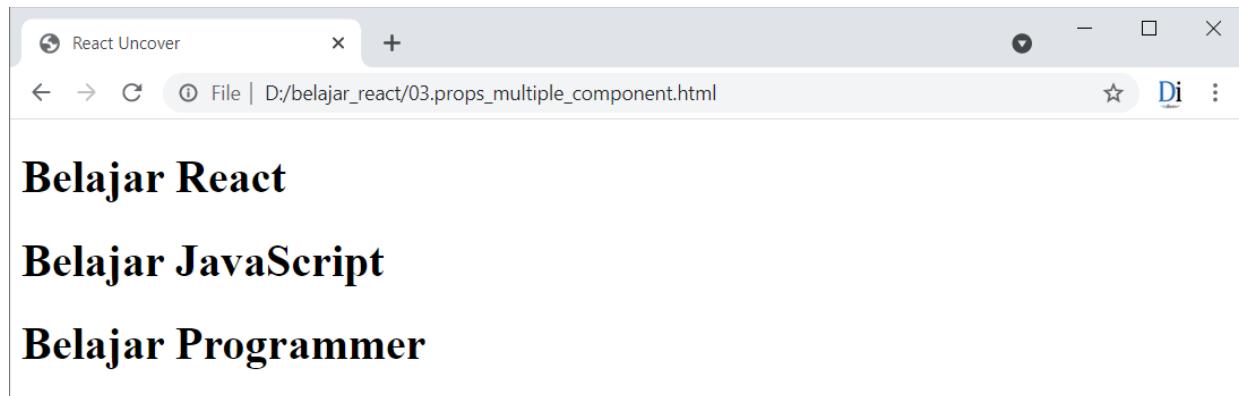
Lalu sesampaikan di komponen, atribut tadi akan disimpan ke dalam `this.props` yang mirip seperti *parameter* (variabel penampung di dalam function).

Konsep ini akan makin jelas saat kita masuk ke praktik `props` di functional component.

Setiap kali suatu komponen di akses, itu akan membuat semacam "scope" atau ruang lingkup terpisah. Kita bisa mengakses komponen yang sama berulang kali dan mengirim data `props` yang berbeda-beda. Berikut contoh praktik dari konsep ini:

03.props_multiple_component.html

```
1 class Belajar extends React.Component {
2     render() {
3         return <h1>Belajar {this.props.materi} </h1>;
4     }
5 }
6
7 const myElement = (
8     <div>
9         <Belajar materi="React"/>
10        <Belajar materi="JavaScript"/>
11        <Belajar materi="Programmer"/>
12    </div>
13 )
14
15 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



Gambar: Mengakses multiple React component

Kode untuk pendefinisian komponen Belajar masih sama seperti sebelumnya, hanya saja sekarang di dalam konstanta myElement saya mengakses 3 kali komponen Belajar dengan atribut materi yang berbeda-beda. Hasilnya, isi property `this.props.materi` akan tampil sesuai nilai yang dikirim saat komponen tersebut diakses.

Lanjut ke contoh yang lebih kompleks, silahkan pelajari sejenak maksud dari kode program di bawah ini. Agar lebih mudah dibaca, sebaiknya mulai dari kode paling bawah:

04.props_mahasiswa.html

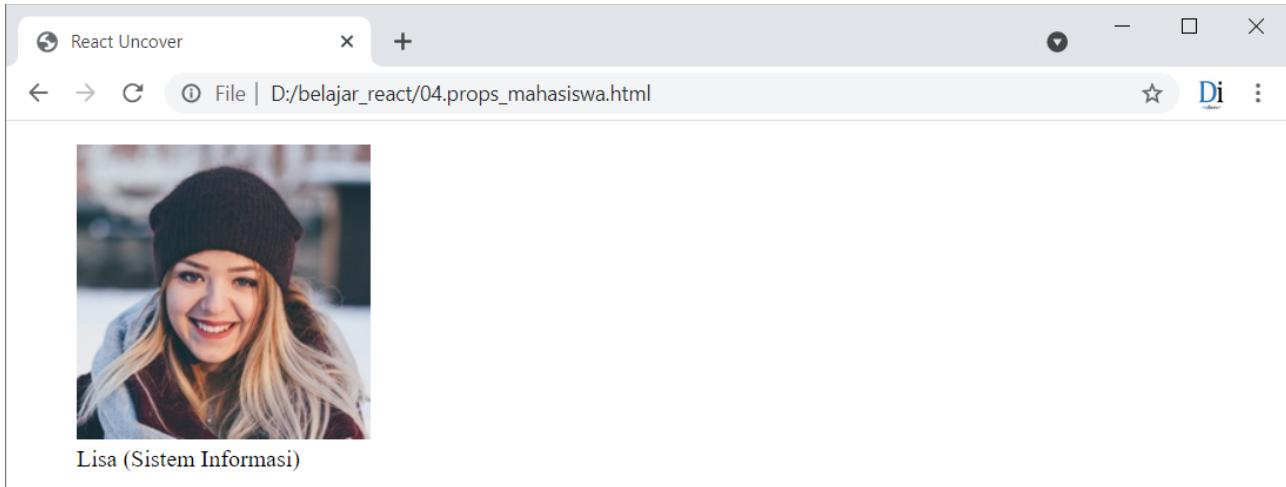
```

1  class Mahasiswa extends React.Component {
2      render() {
3          return (
4              <figure>
5                  <img src={"img/" + this.props.pasFoto} alt={this.props.nama} />
6                  <figcaption>{this.props.nama} ({this.props.jurusan})</figcaption>
7              </figure>
8          )
9      }
10 }
11
12 class DaftarMahasiswa extends React.Component {
13     render() {
14
15         const mahasiswa =
16             {
17                 nama: "Lisa",
18                 jurusan: "Sistem Informasi",
19                 pasFoto: "people2.jpg"
20             };
21
22         return (
23             <Mahasiswa
24                 nama={mahasiswa.nama}
25                 jurusan={mahasiswa.jurusan}
26                 pasFoto={mahasiswa.pasFoto}
27             />
28         )
29     }

```

React Props

```
30  }
31
32 ReactDOM.createRoot(document.getElementById('root')).render(<DaftarMahasiswa />);
```



Gambar: Menampilkan data mahasiswa menggunakan props

Di baris 32, method `ReactDOM.createRoot().render()` akan memproses komponen `<DaftarMahasiswa/>`, maka kita lanjut masuk ke kode yang mendefinisikan komponen ini.

Definisi komponen `DaftarMahasiswa` ada di baris 12 – 30. Di dalam method `render()`, terdapat konstanta `mahasiswa` yang berisi 1 object dengan 3 property: `nama`, `jurusan` dan `pasFoto`.

Setelah itu saya me-return kode JSX dari komponen `DaftarMahasiswa` di baris 23 – 27.

Perhatikan kode yang ada, isinya akan mengakses komponen `<Mahasiswa/>` dan mengirim 3 buah atribut, yakni `nama={mahasiswa.nama}`, `jurusan={mahasiswa.jurusan}` dan `pasFoto={mahasiswa.pasFoto}`.

Saatnya masuk ke definisi komponen `Mahasiswa` di baris 1 – 10. Sampai di sini kita sudah mengetahui kalau komponen `Mahasiswa` akan menerima 3 buah data `props`. Data tersebut bisa diakses dari `this.props.nama`, `this.props.jurusan` dan `this.props.pasFoto`, tinggal selanjutnya "meramu" data-data ini untuk menjadi sebuah struktur JSX.

Alur pengiriman data di atas akan sangat sering kita jumpai di React. Parent component `DaftarMahasiswa` mengirim data dalam bentuk `props` ke dalam child component `Mahasiswa`. Nantinya, komponen `Mahasiswa` bisa saja meneruskan `props` tadi ke child component lain, dst.

#Exercise

Di akhir bab **React Component**, terdapat kode program yang menampilkan 3 object dari array `mahasiswas`. Di situ, array `mahasiswas` saya definisikan sebagai *global variable* agar bisa diakses oleh komponen `DaftarMahasiswa` dan komponen `Mahasiswa`.

Berikut kode yang dimaksud:

```
1 const mahasiswas = [
```

```

2   {
3     id: "01",
4     nama: "Eka",
5     jurusan: "Teknik Informatika",
6     pasFoto: "people1.jpg"
7   },
8   {
9     id: "02",
10    nama: "Lisa",
11    jurusan: "Sistem Informasi",
12    pasFoto: "people2.jpg"
13 },
14 {
15   id: "03",
16   nama: "Rudi",
17   jurusan: "Teknik Elektro",
18   pasFoto: "people4.jpg"
19 }
20 ];
21
22 class Mahasiswa extends React.Component {
23   render() {
24     return (
25       mahasiswas.map((mahasiswa) =>
26         <figure key={mahasiswa.id}>
27           <img src={"img/" + mahasiswa.pasFoto} alt={mahasiswa.nama} />
28           <figcaption>{mahasiswa.nama} ({mahasiswa.jurusan})</figcaption>
29         </figure>
30       )
31     )
32   }
33 }
34
35 class DaftarMahasiswa extends React.Component {
36   render() {
37     return (
38       <React.Fragment>
39         <h1>Daftar Mahasiswa - Teknik Informatika</h1>
40         <hr />
41         <section style={{ display: "flex", textAlign: "center" }}>
42           <Mahasiswa />
43         </section>
44       </React.Fragment>
45     )
46   }
47 }
48
49 ReactDOM.createRoot(document.getElementById('root'))
50   .render(<DaftarMahasiswa/>);

```

Sebagai bahan latihan, saya ingin memindahkan array `mahasiswas` ke dalam komponen `DaftarMahasiswa`.

Efek dari perubahan ini, komponen `Mahasiswa` tidak bisa lagi mengakses array `mahasiswas`, serta perulangan `mahasiswas.map()` di baris 25 otomatis tidak bisa berjalan.

Karena itu, data setiap mahasiswa harus dikirim sebagai `props` dari komponen `DaftarMahasiswa` ke dalam komponen `Mahasiswa`. Perulangan `mahasiswas.map()` juga perlu ikut dipindah dari komponen `Mahasiswa` ke komponen `DaftarMahasiswa`.

Silahkan coba rancang sebentar alur kode program untuk masalah ini. Saya ingin hasil akhirnya tetap sama seperti tampilan awal.

Baik, berikut salah satu solusi dari latihan kita:

`05.props_mahasiswas_exercise.html`

```

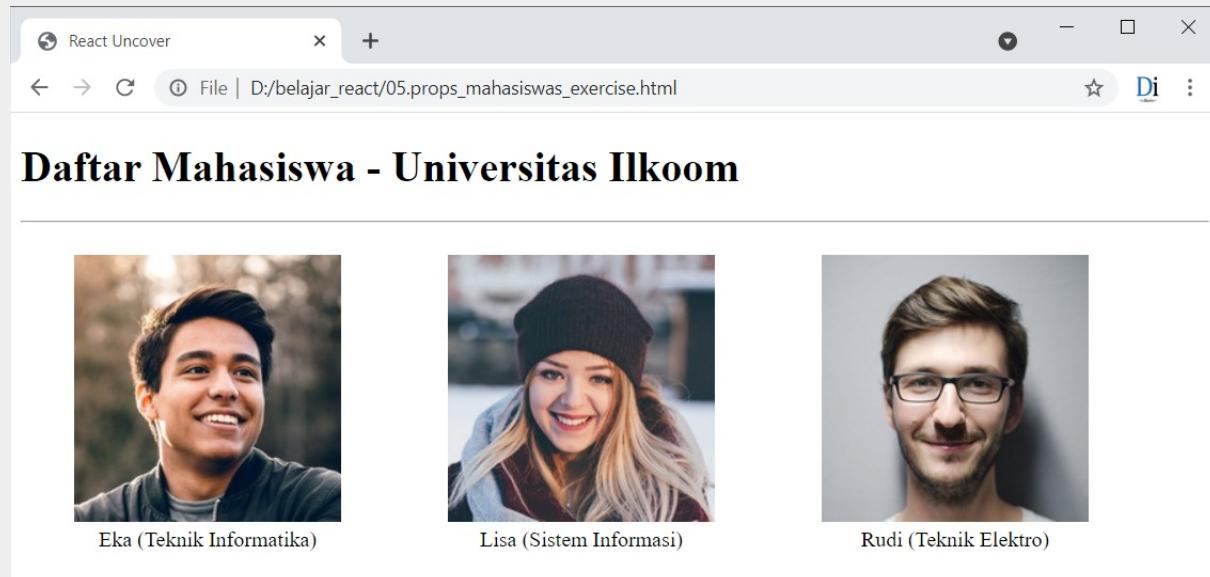
1 class Mahasiswa extends React.Component {
2   render() {
3     return (
4       <figure key={this.props.id}>
5         <img src={"img/" + this.props.pasFoto} alt={this.props.nama} />
6         <figcaption>{this.props.nama} ({this.props.jurusan})</figcaption>
7       </figure>
8     )
9   }
10 }
11
12 class DaftarMahasiswa extends React.Component {
13   render() {
14
15     const mahasiswas = [
16       {
17         id: "01",
18         nama: "Eka",
19         jurusan: "Teknik Informatika",
20         pasFoto: "people1.jpg"
21       },
22       {
23         id: "02",
24         nama: "Lisa",
25         jurusan: "Sistem Informasi",
26         pasFoto: "people2.jpg"
27       },
28       {
29         id: "03",
30         nama: "Rudi",
31         jurusan: "Teknik Elektro",
32         pasFoto: "people4.jpg"

```

```

33     }
34 ];
35
36 return (
37   <React.Fragment>
38     <h1>Daftar Mahasiswa - Universitas Ilkoom</h1>
39     <hr />
40     <section style={{ display: "flex", textAlign: "center" }}>
41       {
42         mahasiswa.map((mahasiswa) =>
43           <Mahasiswa
44             key={mahasiswa.id}
45             nama={mahasiswa.nama}
46             jurusan={mahasiswa.jurusan}
47             pasFoto={mahasiswa.pasFoto}
48           />
49         )
50       }
51     </section>
52   </React.Fragment>
53 )
54 }
55 }
56
57 ReactDOM.createRoot(document.getElementById('root'))
58   .render(<DaftarMahasiswa/>);

```



Gambar: Menampilkan array mahasiswa menggunakan props

Kunci utama dari kode ini ada di baris 42-49. Setiap kali method `mahasiswa.map()` melakukan perulangan, komponen `<Mahasiswa />` akan dijalankan dengan mengirim atribut `key`, `nama`, `jurusan` dan `pasFoto` yang berbeda-beda. Semua atribut diterima sebagai `props` di dalam komponen `Mahasiswa`.

Children Props

Nama *props* yang kita akses di dalam komponen akan selalu berpasangan dengan nama atribut saat komponen itu ditulis. Misalnya `this.props.nama` dipakai untuk mengakses nilai atribut *nama*, atau `this.props.jurusan` akan mengakses nilai atribut *jurusan*, dst.

Diluar itu, terdapat satu *props* khusus yang dipakai bukan untuk mengakses atribut, tapi mengakses "sesuatu" yang ada di antara tag pembuka dan tag penutup komponen.

Selama ini kita mengakses React component menggunakan *self closing tag* seperti `<Belajar/>` atau `<Mahasiswa/>`. Sebenarnya, komponen itu juga bisa ditulis secara berpasangan seperti `<Belajar></Belajar>`. Akan tetapi penulisan seperti ini lebih cocok jika terdapat teks antara tag pembuka dan tag penutup. Isi teks inilah yang nantinya bisa diakses dari **`this.props.children`**.

Berikut contoh prakteknya:

06.props_children.html

```

1  class Belajar extends React.Component {
2    render() {
3      return <h1 id={this.props.id}>Belajar {this.props.children}</h1>;
4    }
5  }
6
7 const myElement = <Belajar id="01">React</Belajar>
8
9 ReactDOM.createRoot(document.getElementById('root')).render(myElement);

```

Di baris 7, saya mengakses komponen *Belajar* dengan pasangan tag pembuka dan penutup. Di antara kedua tag ini terdapat teks "React". Sesampainya di komponen *Belajar*, teks "React" bisa diakses dari perintah `this.props.children`, sehingga hasil akhir kode di atas adalah:

`<h1 id="01">Belajar React</h1>`

Inilah fungsi dari `this.props.children`.

Pada prakteknya, `this.props.children` relatif jarang dipakai. Mengirim data dalam bentuk atribut biasa terasa lebih rapi, namun bisa saja ini berguna dalam beberapa kasus tertentu.

Mengakses Props di Constructor

Di pembahasan tentang class component pada bab sebelumnya, kita sempat menyinggung tentang *constructor*. Dalam konsep OOP, *constructor* berfungsi untuk menjalankan kode program saat proses instansiasi atau pada saat object pertama kali dibuat.

Salah satu hal yang biasa dilakukan di *constructor* adalah membuat property dan mengisinya dengan nilai awal (proses inisialisasi). Nilai awal ini bisa saja berasal dari *props*, namun ada sedikit masalah yang bisa terjadi:

React Props

07.props_constructor_problem.html

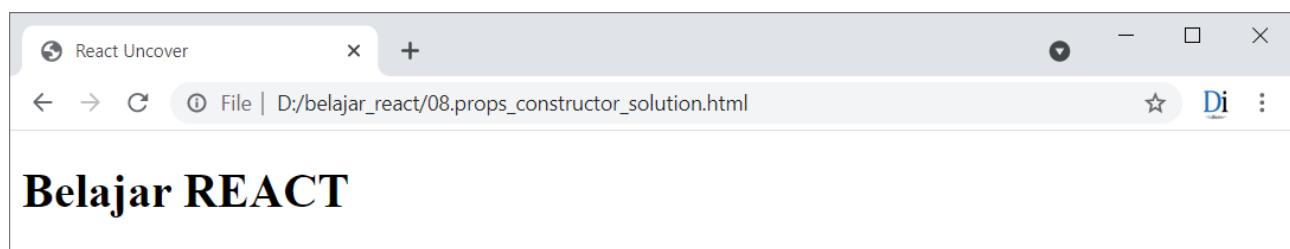
```
1 class Belajar extends React.Component {  
2     constructor() {  
3         super();  
4         this.judul = this.props.materi.toLocaleUpperCase(); // ini akan error  
5     }  
6     render() {  
7         return <h1>Belajar {this.judul} </h1>;  
8     }  
9 }  
10  
11 const myElement = <Belajar materi="React" />  
12  
13 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Dalam contoh ini saya membuat property `this.judul` di dalam constructor (baris 4), dan ingin mengisinya dengan versi huruf besar dari `this.props.materi`. Akan tetapi kode tersebut menjadi error karena ternyata `props` tidak terdefinisi di dalam constructor: `Uncaught TypeError: Cannot read properties of undefined (reading 'materi')`.

Pada situasi kita ingin mengakses `props` di dalam constructor, maka harus mengisi `props` sebagai argument method `super()`, yakni menjadi `super(props)`, selain itu `props` juga harus dilewatkan sebagai argument `constructor()`:

08.props_constructor_solution.html

```
1 class Belajar extends React.Component {  
2     constructor(props) {  
3         super(props);  
4         this.judul = this.props.materi.toLocaleUpperCase();  
5     }  
6     render() {  
7         return <h1>Belajar {this.judul} </h1>;  
8     }  
9 }  
10  
11 const myElement = <Belajar materi="React" />  
12  
13 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```



Gambar: Melewatkannya props ke dalam constructor

Perubahannya ada di baris 2 dan 3. Inilah yang harus dilakukan jika ingin mengakses `props` dari dalam constructor.

Meskipun perintah `super(props)` hanya diperlukan pada situasi tertentu (seperti dalam contoh kita), tapi banyak programmer React yang selalu menulis kode ini ketika membuat constructor, walaupun tidak sedang mengakses props di constructor.

Tidak ada yang salah dari kebiasaan ini, hanya saja itu tidak wajib. Melewatkannya props ke dalam `super()` baru diperlukan jika kita ingin mengakses props dari dalam constructor. Di luar itu seperti di method `render()`, props tetap bisa diakses meskipun hanya menulis perintah `super()` saja.

Jika butuh penjelasan yang lebih teknis, bisa kunjungi: [Why Do We Write `super\(props\)`?¹⁷](#)

6.3. Props di Functional Component

Di *functional component*, **props** berfungsi sama, yakni untuk mengirim data antar komponen. Namun karena tidak berbentuk class, kita tidak perlu menggunakan keyword `this` saat mengakses props. Sebagai gantinya, props perlu ditampung ke dalam parameter fungsi.

Penjelasan ini akan lebih mudah dengan contoh berikut:

```
09.function_props.html
1  const Belajar = (props) => <h1>Belajar {props.materi} </h1>;
2
3  const myElement = (
4    <div>
5      <Belajar materi="React" />
6      <Belajar materi="JavaScript" />
7      <Belajar materi="Programmer" />
8    </div>
9  )
10
11 ReactDOM.createRoot(document.getElementById('root')).render(myElement);
```

Hasil struktur HTML:

```
<div id="root">
  <div>
    <h1>Belajar React </h1>
    <h1>Belajar JavaScript </h1>
    <h1>Belajar Programmer </h1>
  </div>
</div>
```

Cara mengakses *props* ada di baris 1, yakni dengan perintah `{props.materi}`. Akan tetapi kita harus menginput keyword `props` sebagai parameter komponen.

Nama parameter untuk menampung *props* ini bisa saja diganti dengan nama lain, React secara

¹⁷. <https://overreacted.io/why-do-we-write-super-props>

otomatis akan mengisinya dengan data props. Baris 1 dari contoh di atas juga bisa ditulis sebagai berikut:

```
const Belajar = (data) => <h1>Belajar {data.materi} </h1>;
```

Akan tetapi mayoritas programmer React tetap memilih menggunakan parameter dengan nama "props" agar tidak bingung.

#Exercise

Sebagai latihan, bisakah anda mengubah exercise kita sebelumnya yang menampilkan data array `mahasiswa` dari class component menjadi functional component?

Pada dasarnya tidak butuh banyak perubahan, hanya di bagian awal komponen saja.

Berikut hasil perubahan kode tersebut:

10.function_mahasiswa_exercise.html

```
1 const Mahasiswa = (props) => {
2   return (
3     <figure key={props.id}>
4       <img src={"img/" + props.pasFoto} alt={props.nama} />
5       <figcaption>{props.nama} ({props.jurusan})</figcaption>
6     </figure>
7   )
8 }
9
10 const DaftarMahasiswa = () => {
11
12   const mahasiswa = [
13     {
14       id: "01",
15       nama: "Eka",
16       jurusan: "Teknik Informatika",
17       pasFoto: "people1.jpg"
18     },
19     {
20       id: "02",
21       nama: "Lisa",
22       jurusan: "Sistem Informasi",
23       pasFoto: "people2.jpg"
24     },
25     {
26       id: "03",
27       nama: "Rudi",
```

```

28     jurusan: "Teknik Elektro",
29     pasFoto: "people4.jpg"
30   }
31 ];
32
33 return (
34   <React.Fragment>
35     <h1>Daftar Mahasiswa - Universitas Ilkoom</h1>
36     <hr />
37     <section style={{ display: "flex", textAlign: "center" }}>
38       {
39         mahasiswa.map((mahasiswa) =>
40           <Mahasiswa
41             key={mahasiswa.id}
42             nama={mahasiswa.nama}
43             jurusan={mahasiswa.jurusan}
44             pasFoto={mahasiswa.pasFoto}
45           />
46         )
47       }
48     </section>
49   </React.Fragment>
50 )
51 }
52
53 ReactDOM.createRoot(document.getElementById('root'))
54   .render(<DaftarMahasiswa/>);

```

Kode ini saya rasa tidak butuh penjelasan lebih lanjut karena yang berubah hanya di bagian awal komponen saja. Kemudian kita juga perlu menghapus perintah `this` ketika mengakses props di dalam komponen `Mahasiswa`.

Dalam bab ini kita telah membahas fungsi dan cara mengakses **React props**. Penggunaan props di class component memang sedikit lebih rumit dibandingkan functional component, namun semoga anda bisa memahaminya.

Lanjut, kita akan masuk ke materi tentang **React Event**.

7. React Event

Kali ini kita akan bahas cara penulisan JavaScript event ke dalam React.

7.1. Pengertian Event dan Event Handler

Dalam JavaScript, **event** adalah suatu hal yang bisa dilakukan ke element HTML, seperti di click, di double click, di hover (meletakkan cursor mouse di atasnya), dll.

Untuk setiap event, kita bisa membuat sebuah **event handler**, yakni kode program yang dijalankan saat event itu terjadi. Misal ketika sebuah tombol diklik, tampilkan sebuah pesan. Atau ketika inputan form diubah, sembunyikan gambar, dst.

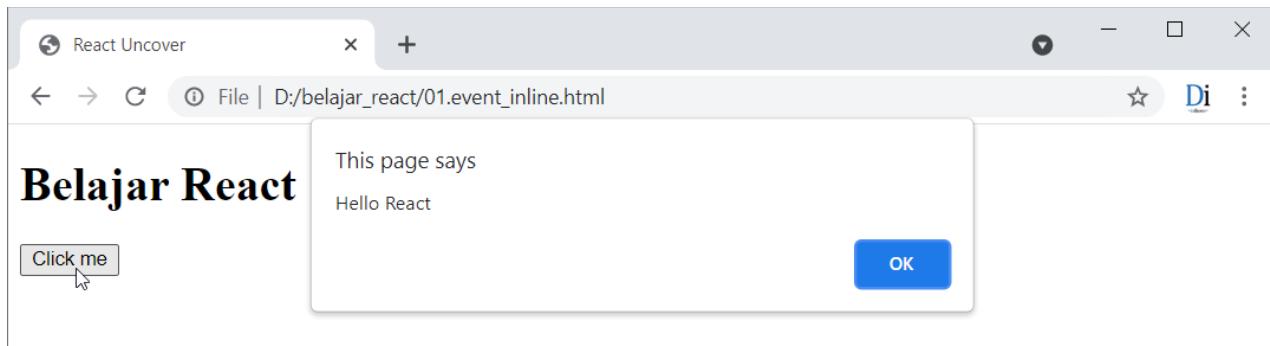
Event dan event handler inilah yang membuat efek interaktif ke sebuah halaman web. Di React, event menjadi fitur yang sangat penting karena dengan event-lah aplikasi web menjadi **reactive**.

Sama seperti di bab sebelumnya, kita akan bahas cara pembuatan event di class component terlebih dahulu. Setelah itu baru masuk ke functional component di bagian akhir.

7.2. Event Handler di Class Component

Sama di JavaScript biasa, event handler di React juga bisa ditulis dalam bentuk inline (langsung ke dalam atribut HTML). Berikut contoh penulisannya:

```
01.event_inline.html
1  class MyApp extends React.Component {
2      render() {
3          return (
4              <div>
5                  <h1>Belajar React</h1>
6                  <button onClick={() => alert("Hello React")}>Click me</button>
7              </div>
8          )
9      }
10 }
11
12 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Menjalankan inline event handler di React

Di baris 6 saya membuat tag `<button>` dengan sebuah atribut event `onClick`. Atribut ini berisi inline function `() => alert("Hello React")`. Maka ketika tombol di klik, akan tampil jendela alert dengan teks "Hello React".

Sedikit berbeda dengan JavaScript native, penulisan nama event di React menggunakan *camelCase*, yakni diawali huruf besar di setiap kata, kecuali kata pertama.

Jika di JavaScript ditulis sebagai `onclick` dan `onmouseenter`, maka di React ditulis sebagai `onClick`, dan `onMouseEnter`. Daftar lengkap dari penulisan event ini bisa ke [React SyntheticEvent¹⁸](#).

Selain menulis langsung secara inline, kode untuk event handler juga bisa dipisah menjadi sebuah method di dalam class component:

02.event_handler.html

```

1  class MyApp extends React.Component {
2
3      handleButtonClick() {
4          alert("Hello React");
5      }
6
7      render() {
8          return (
9              <div>
10                 <h1>Belajar React</h1>
11                 <button onClick={this.handleButtonClick}>Click me</button>
12             </div>
13         )
14     }
15
16 }
17
18 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

¹⁸. <https://reactjs.org/docs/events.html>

Sekarang di baris 11 atribut `onClick` berisi nilai `this.handleClick`. Ini merujuk ke sebuah method `handleButtonClick()` yang ada di baris 3-5. Artinya ketika event click terjadi, jalankan method `handleButtonClick()`.

Method `handleButtonClick()` bertindak sebagai event handler method, dan kita bebas memberikan nama apa saja. Sepanjang saya mempelajari React, terdapat 2 saran pemberian nama untuk event handler method, yakni salah satu dari:

- `handle<Sesuatu><Nama_event>`, contoh: `handleButtonClick()`, `handleFormSubmit()`, atau `handleMahasiswaDelete()`
- `<sesuatu><Nama_event>Handler`, contoh: `buttonClickHandler()`, `formSubmitHandler()`, atau `mahasiswaDeleteHandler()`

Cara penulisan ini tidak wajib, lebih ke saran agar mudah membedakan dengan method-method lain yang nantinya ada di dalam React component.

Event juga bisa ditempatkan terpisah ke setiap komponen. Silahkan pelajari sebentar kode berikut:

03.event_component.html

```

1  class Tombol extends React.Component {
2      handleClick() {
3          alert("Hello React");
4      }
5
6      render() {
7          return <button onClick={this.handleClick}>Click me</button>
8      }
9  }
10
11 class MyApp extends React.Component {
12     render() {
13         return (
14             <div>
15                 <h1>Belajar React</h1>
16                 <Tombol />
17             </div>
18         )
19     }
20 }
21
22 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Hasil dari kode ini masih sama seperti contoh kita sebelumnya. Namun sekarang saya membuat komponen **Tombol** terpisah yang akan diakses dari komponen **MyApp**.

7.3. Event Binding

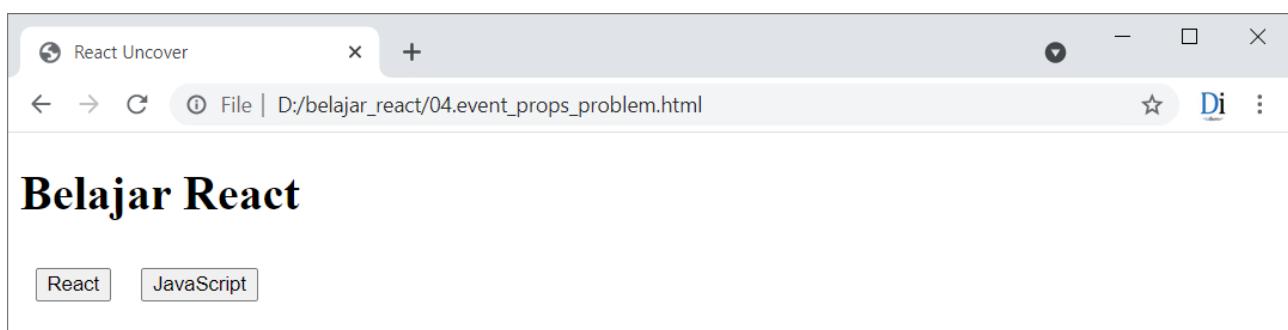
Di dalam method *event handler*, kita bisa menjalankan kode JavaScript apapun. Yang cukup sering dilakukan adalah mengirim data sebagai *props*, lalu diproses oleh *event handler* seperti contoh berikut:

04.event_props_problem.html

```

1  class Tombol extends React.Component {
2      handleButtonClick() {
3          alert(this.props.pesan);
4      }
5
6      render() {
7          return (
8              <button onClick={this.handleButtonClick} style={{ margin: "10px" }}>
9                  {this.props.children}
10             </button>
11         )
12     }
13 }
14
15 class MyApp extends React.Component {
16     render() {
17         return (
18             <div>
19                 <h1>Belajar React</h1>
20                 <Tombol pesan="Belajar React">React</Tombol>
21                 <Tombol pesan="Belajar JavaScript">JavaScript</Tombol>
22             </div>
23         )
24     }
25 }
26
27 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses props dari event handler

Tips untuk memahami kode yang cukup panjang seperti ini adalah mulai dari method `ReactDOM.createRoot().render()` untuk mengetahui apa komponen yang akan di render. Dalam contoh di atas, komponen tersebut adalah `<MyApp />`.

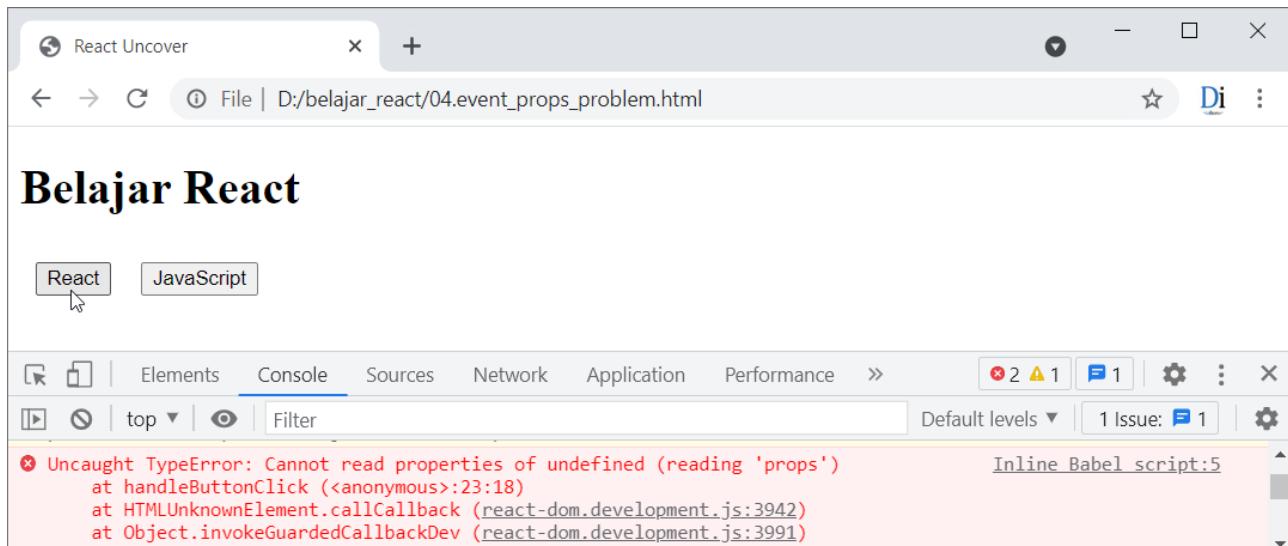
Di dalam struktur JSX komponen `MyApp`, terdapat satu tag `<h1>` dan dua buah tag `<Tombol/>`. Karena diawali dengan huruf besar, maka tag `<Tombol/>` ini tidak lain berbentuk *React component*. Selain itu terdapat atribut `pesan` yang ikut dikirim ke dalam komponen `Tombol`.

Pendefinisian komponen `Tombol` ada di baris 1 – 13. Untuk komponen yang memiliki beberapa method, sebaiknya baca dari method `render()` terlebih dahulu, lalu baru masuk ke method-method lain saat ada perintah yang mengaksesnya.

Di dalam method `render()` milik komponen `Tombol`, terdapat struktur JSX dengan tag `<button>`. Atribut `onClick` akan mengakses method `handleButtonClick()` yang berisi perintah `alert(this.props.pesan)`. Artinya pada saat `<button>` di klik, akan tampil jendela alert yang menampilkan teks dari `props.pesan`.

Atribut lain yang ada di tag `<button>` adalah `style` untuk membuat ruang margin sebesar 10px. Di antara tag pembuka `<button>` dan penutup `</button>` juga terdapat perintah `this.props.children`. Maka nilai ini akan berasal dari teks yang ada di antara tag pembuka dan penutup `<Tombol>`.

Mari tes dengan men-klik tombol "React":



Gambar: Error saat tombol "React" di klik

Ternyata tidak tampil apa-apa. Dan begitu tab console di buka, terlihat pesan Error: `Uncaught TypeError: Cannot read properties of undefined (reading 'props')`. Error ini merujuk ke baris 3 dengan alasan `this` tidak terdefinisi.

Masalah ini terjadi karena prinsip kerja JavaScript dimana jika kita memanggil method tanpa tambahan tanda kurung seperti di atribut `onClick={this.handleButtonClick}`, maka method ini harus di **binding** secara manual. Jika tidak, `this` tidak terdefinisi. Dan karena `this` tidak ada, maka kita juga tidak bisa mengakses `props` dari dalam *event handler*.

Penjelasan dari konsep **binding** ini agak rumit dan cukup panjang sehingga tidak akan saya bahas. Jika tertarik, bisa dibaca ke [Understanding JavaScript Bind \(\)](#).¹⁹

Terdapat beberapa solusi dari masalah ini. Pertama, kita bisa men-bind "this" pada saat menulis atribut event, yakni dengan mengubah baris 8 menjadi sebagai berikut:

05.event_bind_inline.html

```
...
<button onClick={this.handleClick.bind(this)} style={{ margin: "10px" }}>
...
...
```

Dengan tambahan ini, maka jendela alert sudah bisa tampil dan tidak ada error lagi.

Cara kedua adalah dengan mengubah nilai atribut event menjadi pemanggilan fungsi:

06.event_bind_inline_this.html

```
...
<button onClick={() => this.handleClick()} style={{ margin: "10px" }}>
...
...
```

Kali ini kita tidak perlu menggunakan method `bind()`, cukup menulis nilai atribut event dengan *arrow notation*.

Dalam kebanyakan kasus, kedua cara ini bisa dipakai. Akan tetapi ada sedikit kelemahan, yakni setiap kali komponen di-render, React akan membuat ulang method `handleButtonClick()`. Penjelasan dari efek ini akan lebih pas saat kita masuk ke materi tentang **state**, tapi intinya ada sedikit pengurangan performa untuk kode yang kompleks.

Solusi lain untuk menghindari efek performa ini adalah dengan men-bind "this" dari dalam constructor:

07.event_bind_constructor.html

```
1  class Tombol extends React.Component {
2      constructor() {
3          super();
4          this.handleClick = this.handleClick.bind(this);
5      }
6
7      handleClick() {
8          alert(this.props.pesan);
9      }
10
11     render() {
12         return (
13             <button onClick={this.handleClick} style={{margin:"10px"}}>
14                 {this.props.children}
15         
```

19. <https://www.smashingmagazine.com/2014/01/understanding-javascript-function-prototype-bind>

React Event

```
15      </button>
16    )
17  }
18 }
19 ...
20 ...
```

Dengan teknik ini, kita tetap menulis isi event `onClick` di baris 13 dengan cara biasa. Proses binding "this" akan dilakukan dari dalam constructor, tepatnya di baris 4.

Cara binding lewat constructor memang tidak mengalami pengurangan performa seperti dua solusi pertama, tapi kode kita sedikit lebih panjang. Alternatif solusi yang menurut saya lebih praktis adalah menggunakan teknik **public class fields syntax**. Caranya, tulis method `event handler` dengan format *arrow notation*:

08.event_bind_class_property.html

```
1  class Tombol extends React.Component {
2    handleButtonClick = () => {
3      alert(this.props.pesan);
4    }
5
6    render() {
7      return (
8        <button onClick={this.handleButtonClick} style={{margin:"10px"}}>
9          {this.props.children}
10         </button>
11      )
12    }
13 }
```

Sekilas tidak ada tampak perubahan, tapi sebelumnya method `handleButtonClick()` di baris 2-4 saya tulis dengan kode berikut:

```
handleButtonClick() {
  alert(this.props.pesan);
}
```

Sekarang dimodifikasi dalam bentuk arrow notation:

```
handleButtonClick = () => {
  alert(this.props.pesan);
}
```

Dengan sedikit perubahan ini, masalah proses binding "this" sudah teratasi.

Dalam dokumentasi React ditulis kalau *public class fields syntax* ini masih *experimental* (uji coba). Teknik ini sebenarnya bukan bawaan React, tapi milik JavaScript dan masih dalam tahap proposal untuk EcmaScript 2022.

Karena berstatus "proposal", masih ada kemungkinan perubahan syntax saat EcmaScript 2022 dirilis nanti. Namun karena kodanya cukup singkat, saya lebih memilih cara ini saja.

7.4. Event Object

Di dalam JavaScript, kita juga mengenal tentang **event object**, yakni object yang berisi informasi detail tentang event yang sedang terjadi.

Misalnya ketika sebuah tombol di klik, JavaScript otomatis mengirim *event object* ke dalam *event handler*. Dari *event object*, kita bisa mengakses berbagai informasi seperti posisi klik, waktu klik terjadi, hingga node tempat klik berlangsung.

Berikut potongan kode program yang menampilkan *event object* di JavaScript native:

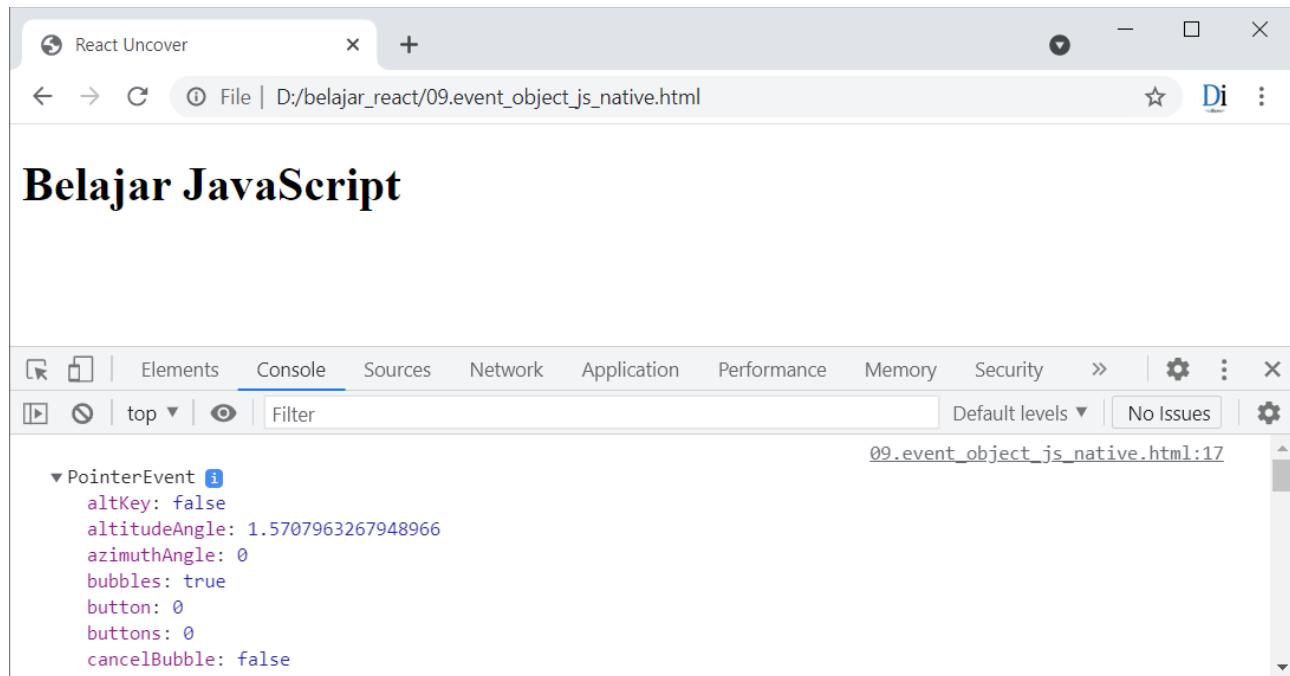
09.event_object_js_native.html

```
1 <body>
2
3   <h1 id="judul">Belajar JavaScript</h1>
4
5   <script>
6     var h1Node = document.getElementById("judul");
7     h1Node.addEventListener("click", function (event) {
8       console.log(event);
9     });
10  </script>
11
12 </body>
```

Di dalam kode HTML, terdapat tag `<h1>` dengan `id="judul"` (baris 3). Masuk ke kode JavaScript, variabel `h1Node` akan diisi *node object* hasil dari `document.getElementById("judul")`, ini tidak lain merujuk ke tag `<h1>` tadi.

Kemudian saya menambah event `click` beserta *event handler* ke `h1Node` (baris 7-9). Parameter `event` akan diisi oleh JavaScript dengan *event object* yang selanjutnya ditampilkan dari perintah `console.log(event)`.

Berikut hasil yang tampil saat tag `<h1>` di klik:



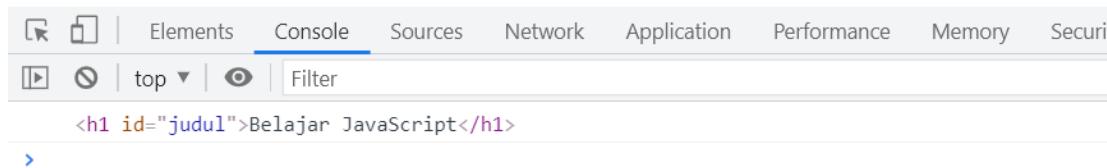
Gambar: Isi dari event object pada JavaScript native

Di dalam tab console, terlihat berbagai info miliki *event object*. Salah satu yang sering diakses adalah property **target** karena menyimpan *node object* tempat event sedang berlangsung:

10.event_object_target_js_native.html

```
1 <body>
2
3   <h1 id="judul">Belajar JavaScript</h1>
4
5   <script>
6     var h1Node = document.getElementById("judul");
7     h1Node.addEventListener("click", function (event) {
8       console.log(event.target);
9     });
10    </script>
11
12 </body>
```

Dengan perubahan di baris 8, sekarang ketika tag `<h1>` di klik akan tampil hasil berikut:



Gambar: Hasil perintah `console.log(event.target)`

Menggunakan *event object*, kita bisa membuat *event handler* yang dinamis, tidak terpaku untuk satu element saja.

Jika anda masih ragu tentang apa itu *event object*, bisa pelajari lagi di JavaScript dasar. Dalam buku **JavaScript Uncover** saya menyediakan satu bab khusus yang membahas **DOM Event**, termasuk di dalamnya tentang *event object*.

Kembali ke React, kita juga bisa "menangkap" *event object* di dalam event handler dengan cara yang sama seperti JavaScript native, yakni melalui parameter. Berikut contohnya:

11.event_object.html

```

1  class Tombol extends React.Component {
2
3    handleButtonClick = (event) => {
4      console.log(this);
5      console.log(event);           // event object
6      console.log(event.target);   // node object tempat event terjadi
7      console.log(event.target.innerHTML); // ambil isi innerHTML
8    }
9
10   render() {
11     return (
12       <button onClick={this.handleButtonClick} style={{ margin: "10px" }}>
13         {this.props.children}
14       </button>
15     )
16   }
17 }
18
19 class MyApp extends React.Component {
20   render() {
21     return (
22       <div>
23         <h1>Belajar React</h1>
24         <Tombol pesan="Belajar React">React</Tombol>
25         <Tombol pesan="Belajar JavaScript">JavaScript</Tombol>
26       </div>
27     )
28   }
29 }
30
31 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Kode untuk komponen `MyApp` dan method `render()` milik komponen `Tombol` masih sama seperti sebelumnya. Yang berubah ada di method `handleButtonClick()` antara baris 3-8.

Method `handleButtonClick()` sekarang menerima satu parameter, yakni: `event`. Parameter `event` otomatis diisi React dengan ***synthetic events object***.

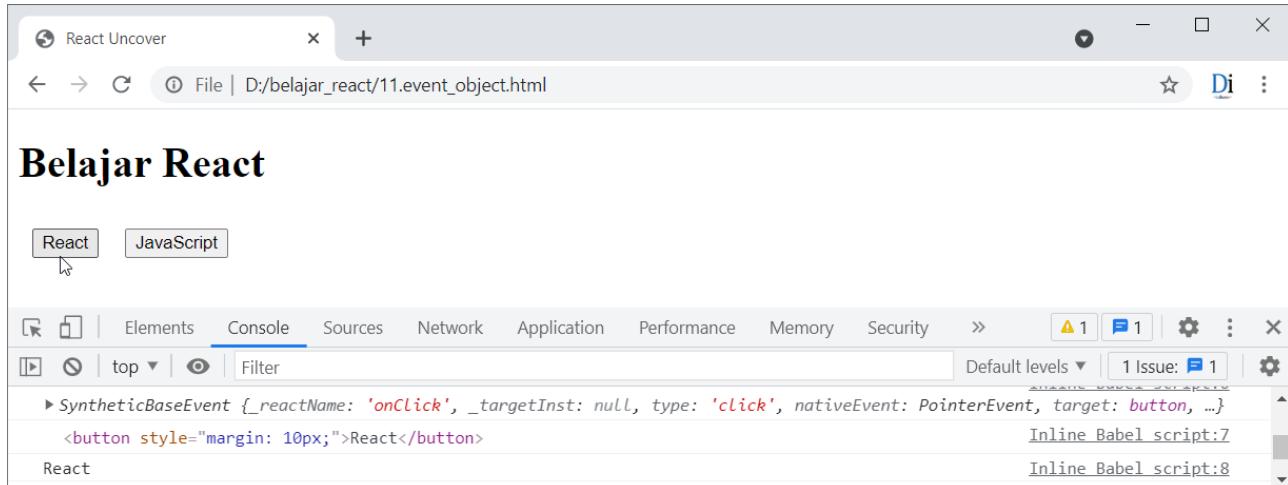
Disebut sebagai *synthetic* karena ini bukanlah *event object* yang sama seperti di JavaScript native, tapi di-generate sendiri oleh React untuk menghindari perbedaan implementasi antar web browser. Meskipun berbeda, mayoritas property *synthetic events object* tetap sama

seperti *event object* biasa.

Di dalam method `handleButtonClick()` saya memeriksa 3 nilai menggunakan `console.log()`:

- `event`: Berisi synthetic events object tempat event terjadi.
- `event.target`: Berisi node object tempat event terjadi.
- `event.target.innerHTML`: Berisi isi teks yang ada di element tempat event terjadi.

Silahkan jalankan kode di atas, lalu klik tombol "React" dan cek tab Console:



Gambar: Melihat hasil synthetic events object React

Hasil dari `console.log(event)` tampil sebagai `SyntheticBaseEvent`. Object ini punya berbagai property yang mayoritas sama seperti milik *event object* biasa.

Salah satu property yang sering dipakai adalah `event.target`. Property ini berisi *node object* tempat event berlangsung. Karena saya men-klik tombol React, maka `event.target` akan berisi *node object* dari tag `<button style="margin: 10px;">React</button>`.

Lebih spesifik lagi, `event.target.innerHTML` akan mengembalikan isi teks yang ada di antara tag pembuka dan penutup milik *object tempat event berlangsung*. Kembali, teks "React" tampil karena saya men-klik tombol React.

Sekarang silahkan klik tombol "JavaScript", maka isi dari *event object*, `event.target`, dan `event.target.innerHTML` juga akan berubah.

Materi tentang **event object** akan kembali kita bahas saat masuk ke **form processing**. Karena nantinya nilai inputan form bisa diakses dari `event.target.value`.

7.5. Menurunkan (Passing Down) Event Handler

Dalam kondisi tertentu, kita ingin event yang terjadi di *child component*, bisa menjalankan *event handler* milik *parent component*. Berikut contoh kode yang dimaksud:

12.event_passing_down.html

```

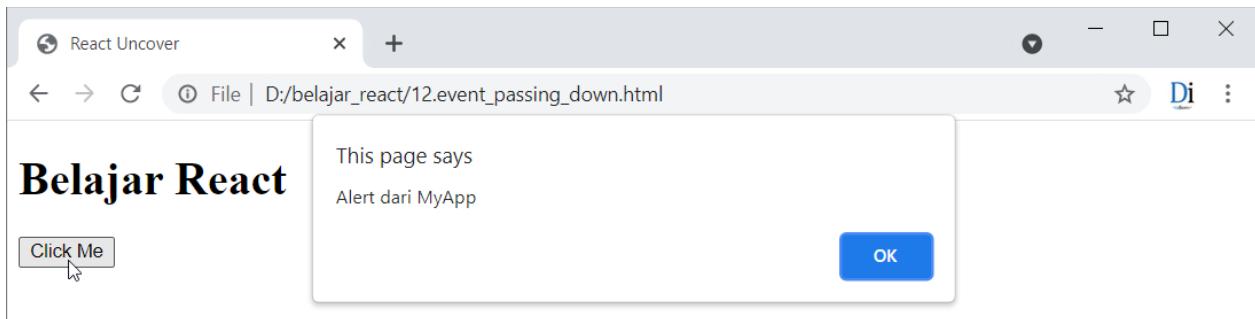
1  class Tombol extends React.Component {
2      render() {
3          return (
4              <button onClick={this.props.onTombolClick}>Click Me</button>
5          )
6      }
7  }
8
9  class MyApp extends React.Component {
10     handleClick = () => {
11         alert("Alert dari MyApp");
12     }
13
14    render() {
15        return (
16            <div>
17                <h1>Belajar React</h1>
18                <Tombol onClick={this.handleClick} />
19            </div>
20        )
21    }
22 }
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Dalam class `MyApp` terdapat method `handleClick()` di baris 10-12 yang berisi 1 perintah untuk menampilkan jendela `alert("Alert dari MyApp")`.

Kemudian di dalam method `render()`, saya mengakses komponen `Tombol` dengan tambahan atribut `onClick={this.handleClick}`. Perhatikan bahwa `props` yang di kirim adalah suatu method, bukan teks biasa. Nilai `this.handleClick` merujuk ke method `handleClick()` yang ada di komponen `MyApp`.

Sesampainya di dalam komponen `Tombol`, property `this.props.onClick` saya isi ke dalam event `onClick` di baris 4. Bisakah anda tebak apa yang terjadi saat tombol di klik? Betul, itu akan menjalankan method `handleClick()` milik komponen `MyApp`:



Gambar: Jendela alert "Alert dari MyApp" tampil saat tombol di klik

Inilah yang dimaksud dengan *passing down* atau menurunkan *event handler* dari parent ke

child component. Dengan cara ini, kita bisa menjalankan method milik parent component, meskipun event-nya ditulis dalam child component.

Pola penulisan seperti ini memang belum terlalu jelas fungsinya sekarang, tapi akan sangat berguna saat kita masuk ke materi **state** dalam bab setelah ini.

Dalam contoh di atas saya menulis atribut `onTombolClick={this.handleClick}`. Di sini, `onTombolClick` bukanlah event, tapi hanya nama atribut biasa. Kita bebas menggantinya dengan nama lain karena hanya berfungsi sebagai nama *props*.

Akan tetapi kebiasaan sebagian programmer React, jika yang dikirim sebagai *props* itu berisi event handler, maka awali dengan kata "on", seperti `onGambarClick`, atau `onItemClick`. Ini agar mudah membedakan dengan nama *props* biasa yang berisi teks.

7.6. Event Handler di Functional Component

Selesai dengan penulisan event handler di class component, sekarang kita masuk ke cara penulisan event handler di functional component. Untungnya, ini lebih mudah karena di functional component kita tidak mengalami masalah dengan proses binding "this" seperti di class component.

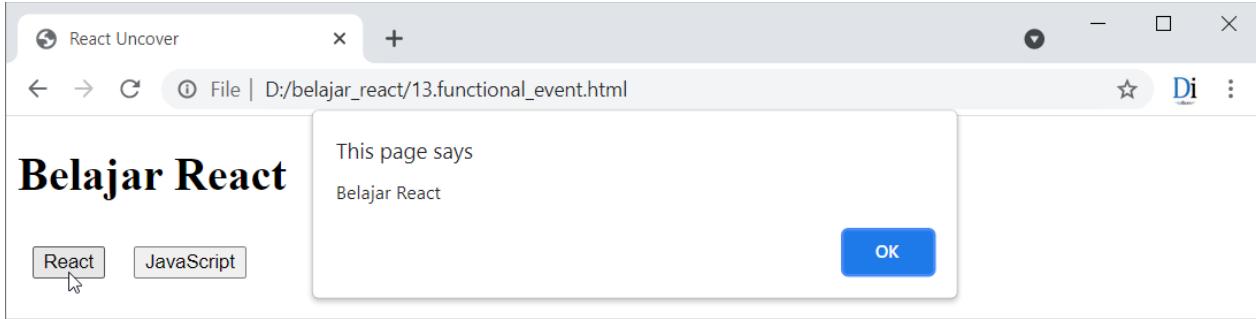
Berikut contoh pembuatan event dan event handler di functional component React:

13. functional_event.html

```

1  const Tombol = (props) => {
2
3      const handleButtonClick = () => {
4          alert(props.pesan);
5      }
6
7      return (
8          <button onClick={handleButtonClick} style={{ margin: "10px" }}>
9              {props.children}
10             </button>
11         )
12     }
13
14 const MyApp = () => {
15     return (
16         <div>
17             <h1>Belajar React</h1>
18             <Tombol pesan="Belajar React">React</Tombol>
19             <Tombol pesan="Belajar JavaScript">JavaScript</Tombol>
20         </div>
21     )
22   }
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Penulisan event dan event handler di functional component

Di dalam function component, *event handler* ditulis sebagai fungsi biasa. Dalam contoh di atas, fungsi `handleButtonClick()` di baris 3-5 bertindak sebagai event handler untuk komponen **Tombol**. Sisa kode lain mirip seperti contoh sebelumnya sehingga tidak akan saya bahas lagi.

Di function component kita juga tidak perlu berurusan dengan proses *binding* karena tidak butuh perintah "this" ketika mengakses `props`.

#Exercise

Event handler di functional component juga bisa menerima *event object*. Caranya juga sama seperti class component, yakni diinput sebagai parameter ke dalam event handler.

Hal yang sama juga berlaku untuk proses *passing down event handler* yang kurang lebih sama seperti di class component.

Sebagai latihan, bisakah anda mengkonversi contoh kode program di kedua materi tersebut ke versi functional component? Contoh kode tersebut ada di file `11.event_object.html` dan `12.event_passing_down.html` (ada di beberapa halaman sebelumnya).

Silahkan coba sebentar.

Berikut hasil konversi dari kedua kode tersebut:

`14.function_event_object.html`

```

1 const Tombol = (props) => {
2
3   const handleButtonClick = (event) => {
4     console.log(event);
5     console.log(event.target);
6     console.log(event.target.innerHTML);
7   }
8
9   return (

```

```

10      <button onClick={handleButtonClick} style={{ margin: "10px" }}>
11          {props.children}
12      </button>
13  )
14 }
15
16 const MyApp = () => {
17   return (
18     <div>
19       <h1>Belajar React</h1>
20       <Tombol pesan="Belajar React">React</Tombol>
21       <Tombol pesan="Belajar JavaScript">JavaScript</Tombol>
22     </div>
23   )
24 }
25
26 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

15.function_event_passing_down.html

```

1 const Tombol = (props) => {
2   return (
3     <button onClick={props.onTombolClick}>Click Me</button>
4   )
5 }
6
7 const MyApp = () => {
8   const handleClick = () => {
9     alert("Alert dari MyApp");
10  }
11
12  return (
13    <div>
14      <h1>Belajar React</h1>
15      <Tombol onTombolClick={handleClick} />
16    </div>
17  )
18 }
19
20 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Tidak banyak perubahan dari versi class component. Yang perlu dilakukan hanya mengubah awalan komponen dari class menjadi function, lalu menghapus perintah "this" saat mengakses props.

Di dalam JavaScript (dan juga React), event-lah yang membuat halaman web menjadi interaktif. Pemahaman seputar cara penggunaan event di React sangat penting karena hampir selalu dipakai di setiap aplikasi.

Penulisan event di class component memang sedikit rumit karena ada masalah di proses binding "this". Ini harus selalu diingat karena sering terlupakan dan bertanya-tanya kenapa props tidak bisa diakses dari dalam event handler. Sedangkan jika menggunakan functional component, tidak perlu berurusan dengan proses binding ini.

Dalam bab berikutnya, kita akan membahas fitur penting lain, yakni **React State**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

8. React State

Sejak awal buku ini kita sudah menulis puluhan kode React, tapi halaman yang dihasilkan masih statis. Ini karena untuk membuat efek interaktif di React, butuh **event** (sudah dibahas dalam bab sebelumnya), dan juga **state**.

State memiliki peranan penting dalam sebuah aplikasi React. Dalam bab ini kita akan bahas apa itu state dan cara penggunaannya.

8.1. Pengertian State

State adalah tempat menyimpanan data di dalam komponen. Dari pengertian ini terlihat kalau state berfungsi sama seperti variabel atau property biasa, dan itu memang benar. Akan tetapi ada satu fitur pembeda, jika nilai state berubah, komponen itu akan di-render ulang. Kita akan bahas pengertian ini secara bertahap.

Pertanyaan awal, kenapa harus menyimpan data di dalam state? Apa tidak bisa menggunakan property biasa saja? Untuk menjawab pertanyaan ini, kita akan bahas dengan kasus berikut:

01.normal_alert.html

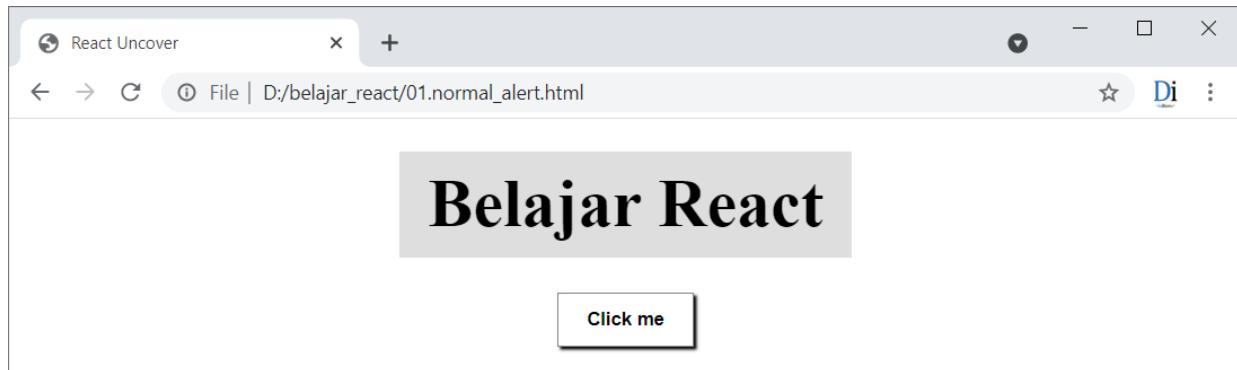
```
1  class MyApp extends React.Component {  
2      constructor(props) {  
3          super(props);  
4          this.judul = "Belajar React";  
5      }  
6  
7      handleButtonClick = () => {  
8          alert("Hello React");  
9      }  
10  
11     render() {  
12         return (  
13             <div>  
14                 <h1>{this.judul}</h1>  
15                 <button onClick={this.handleButtonClick}>Click me</button>  
16             </div>  
17         )  
18     }  
19 }  
20  
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Seperti biasa, kita mulai pembacaan kode program dari bawah terlebih dahulu. Method `ReactDOM.createRoot().render()` akan menampilkan komponen `<MyApp />`, maka lanjut masuk ke method `render()` dari komponen ini.

Isinya berupa container tag `<div>` di baris 13, lalu diikuti tag `<h1>{this.judul}</h1>`. Ini berarti isi teks tag `<h1>` akan diambil dari property `this.judul` yang di deklarasikan dalam constructor (baris 4). Maka nantinya akan diproses React menjadi `<h1>Belajar React</h1>`.

Kembali ke method `render()`, setelah itu terdapat tag `<button>` dengan tambahan atribut `onClick={this.handleButtonClick}`. Ini adalah *event click* yang mengakses *event handler* `handleButtonClick()`. Isi dari method `handleButtonClick()` sekedar menampilkan jendela `alert("Hello React")`.

Tidak ada sesuatu yang baru, semuanya sudah kita pelajari. Dan berikut tampilan halaman tersebut di web browser:



Gambar: Menampilkan judul "Belajar React" dan sebuah tombol "Click me"

Hasilnya terlihat lebih rapi karena saya menambah sedikit kode CSS di bagian `<head>`. Ini tidak berpengaruh ke kode React, hanya sekedar mempercantik saja.

Berikut kode CSS untuk tampilan di atas, bisa ditulis di dalam tag `<head>`:

```

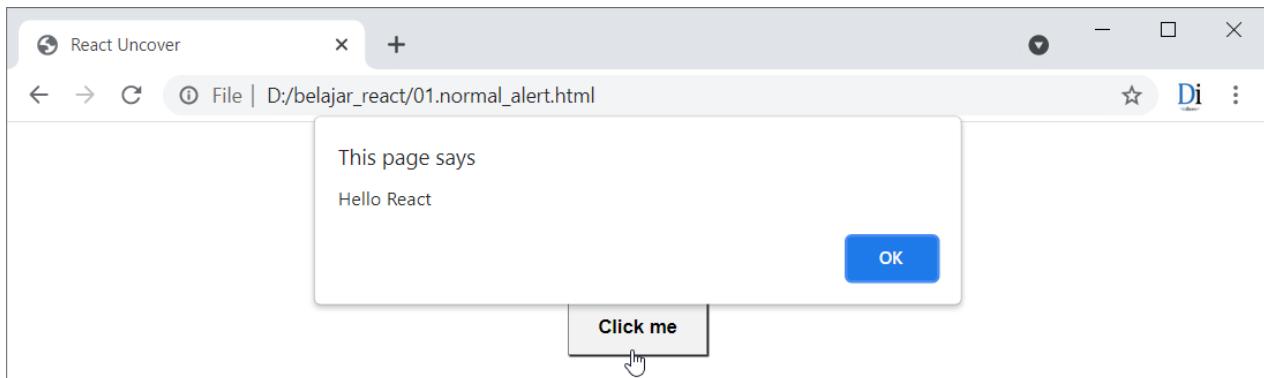
1 <style>
2   #root {
3     display: flex;
4     justify-content: center;
5   }
6
7   div {
8     display: flex;
9     justify-content: center;
10    flex-direction: column;
11    align-items: center;
12  }
13
14  h1 {
15    background-color: gainsboro;
16    font-size: 3em;
  
```

```

17     text-align: center;
18     padding: 10px 20px;
19     margin: 15px 0;
20   }
21
22   button {
23     background-color: white;
24     cursor: pointer;
25     padding: 10px 20px;
26     font-weight: bold;
27     border: 1px solid gray;
28     box-shadow: 2px 2px 2px black;
29     margin: 10px;
30   }
31
32   button:hover {
33     box-shadow: 1px 1px 1px black;
34     background-color: #f1f1f1;
35   }
36 </style>

```

Saat tombol "Click me" di klik, akan tampil jendela alert berikut:



Gambar: Jendela alert tampil saat tombol "Click me" di -klik

Sip, program kita berjalan sesuai dengan keinginan.

Selanjutnya, saya ingin ketika tombol "Click me" di klik, judul tag `<h1>` berubah dari "Belajar React" menjadi "Belajar JavaScript". Sekilas ini tampak mudah, tinggal mengganti isi `event handler handleButtonClick()` dari `alert("Hello React")` menjadi `this.judul="Belajar JavaScript"`. Berikut kode lengkapnya:

```

02.try_change_data.html

1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.judul = "Belajar React";
5    }
6
7    handleButtonClick = () => {

```

```

8     this.judul = "Belajar JavaScript";
9 }
10
11 render() {
12   return (
13     <div>
14       <h1>{this.judul}</h1>
15       <button onClick={this.handleButtonClick}>Click me</button>
16     </div>
17   )
18 }
19 }
20
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Perhatikan kembali isi tag `<h1>` di dalam method `render()`, yakni `<h1>{this.judul}</h1>`. Pada saat halaman tampil pertama kali, `this.judul` akan berisi teks "Belajar React" sebagaimana yang tertulis di baris 4.

Ketika event click terjadi, method `handleButtonClick()` akan dijalankan, maka isi `this.judul` akan berganti dari "Belajar React" menjadi "Belajar JavaScript". Akibat perubahan ini, harusnya teks di tag `<h1>` juga ikut berubah.

Namun berapa kali pun tombol "Click me" di klik, judul "Belajar React" tetap tidak berubah:



Gambar: Judul "Belajar React" tidak berubah menjadi "Belajar JavaScript"

Untuk memastikan isi property `this.judul` sudah berubah saat tombol di klik, saya akan modifikasi method `handleButtonClick()` menjadi sebagai berikut:

03.try_change_data_log.html

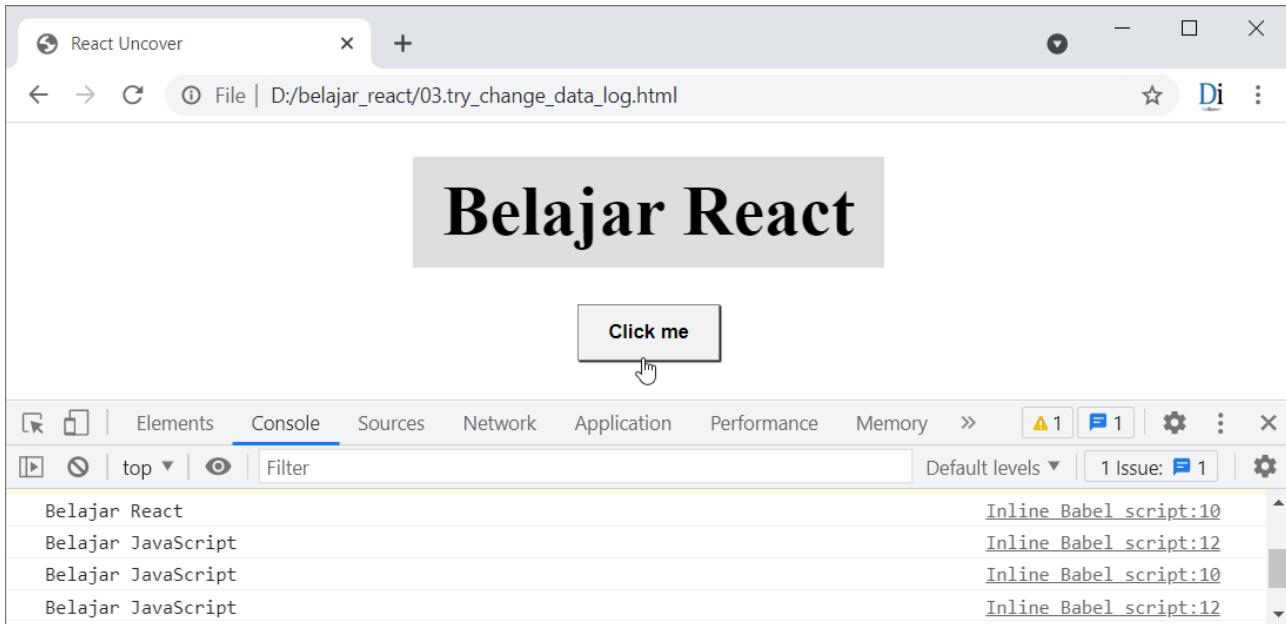
```

1 ...
2   handleButtonClick = () => {
3     console.log(this.judul);
4     this.judul = "Belajar JavaScript";
5     console.log(this.judul);
6   }
7 ...

```

Sekarang begitu tombol di klik, di dalam tab Console akan tampil isi property `this.judul`

sebelum dan sesudah perubahan. Berikut hasilnya:



Gambar: Memeriksa isi property this.judul

Ketika dijalankan pertama kali, tab Console belum berisi teks apapun.

Pada saat tombol "Click me" di klik, di baris 1 akan tampil "Belajar React". Teks ini berasal dari constructor yang menjalankan perintah `this.judul = "Belajar React"`. Lalu baris 2 juga tampil teks "Belajar JavaScript", ini membuktikan kalau isi property `this.judul` sudah berubah.

Jika kita klik tombol "Click me" sekali lagi, di baris 3 dan 4 sama-sama tampil teks "Belajar JavaScript". Ini terjadi karena constructor tidak lagi berjalan, sehingga property `this.judul` akan tetap berisi "Belajar JavaScript".

Berdasarkan percobaan ini bisa diambil kesimpulan bahwa *perubahan nilai property tidak ikut mengubah tampilannya di web browser*. Kita butuh suatu cara agar tampilan komponen di web browser juga ikut berubah, dan itulah fitur utama dari **state**.

Kembali ke pengertiannya, **state** adalah tempat menyimpan data di dalam komponen, dan perubahan isi state akan me-render ulang tampilan komponen.

Sekarang mari kita modifikasi property `this.judul` menjadi sebuah **state**:

04.react_state.html

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { judul: "Belajar React" };
5      }
6
7      handleButtonClick = () => {

```

```

8     this.setState({ judul: "Belajar JavaScript" });
9 }
10
11 render() {
12     return (
13         <div>
14             <h1>{this.state.judul}</h1>
15             <button onClick={this.handleButtonClick}>Click me</button>
16         </div>
17     )
18 }
19 }
20
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Perubahan dari contoh sebelumnya ada di baris 4, 8 dan 14.

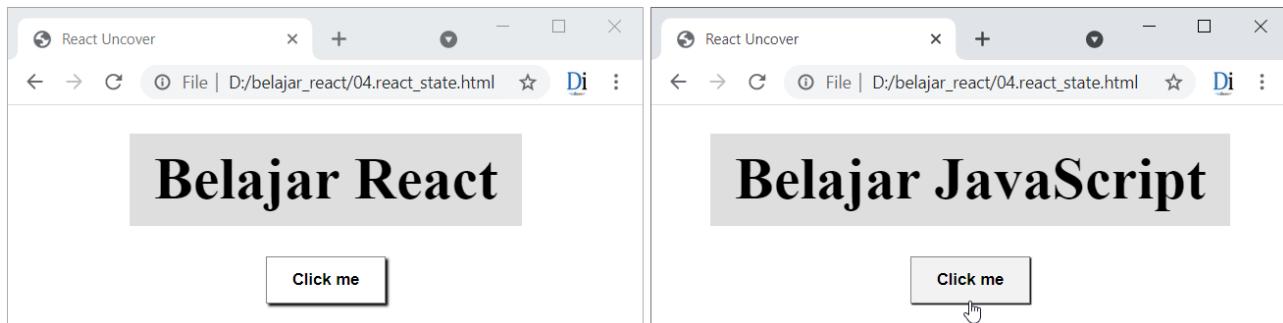
Di baris 4, itu adalah proses pembuatan state di *class component*. Caranya, input suatu object ke dalam property **this.state**. Object yang diinput harus ditulis dengan format `{nama_state: nilai_awal}`. Dalam contoh di atas, bisa disebut kalau saya sedang membuat state `judul` dan mengisinya dengan nilai awal `"Belajar React"`.

Nama state boleh bebas, begitu juga dengan nilainya. Nilai awal state bisa bertipe *string*, *number*, *boolean*, *array*, bahkan *object*. Jika kita tidak ingin memberikan nilai awal, bisa diisi dengan *null*.

Setelah di definisikan, nilai state bisa diakses dengan format `this.state.<nama_state>`. Karena sudah membuat state `judul`, maka perintah untuk mengaksesnya adalah `this.state.judul` seperti di baris 8. Pada saat proses render, React akan mengganti perintah ini dengan `"Belajar React"` sesuai nilai awal state `judul`.

Sekarang kita masuk ke proses mengubah nilai state. Untuk keperluan ini, React menyediakan method khusus, yakni `this.setState()` seperti di baris 8. State yang ingin diubah ditulis sebagai argument dalam bentuk pasangan nama state dan nilainya. Format ini kurang lebih sama seperti pembuatan state awal.

Mari jalankan kode di atas dan klik tombol "Click me":



Gambar: Judul "Belajar React" akan berubah menjadi "Belajar JavaScript" saat tombol di klik

Hasilnya, isi tag `<h1>` akan berubah dari "Belajar React" menjadi "Belajar JavaScript". Inilah efek

dari penggunaan state.

8.2. Prinsip Kerja State

Cara kerja state harus kita pahami karena sangat berpengaruh ke prinsip cara kerja React secara keseluruhan.

Di dalam JavaScript, agar tampilan element bisa berubah, struktur DOM dari element tersebut harus dibuat ulang. Inilah yang menjadi masalah sewaktu kita coba menukar isi property `this.judul` dari "Belajar React" menjadi "Belajar JavaScript". Di sisi JavaScript, isi property `this.judul` memang sudah berubah, akan tetapi element HTML yang menampilkannya juga harus di muat ulang.

React mengatasi masalah ini dengan konsep yang cukup praktis: jika nilai state berubah, render ulang komponen agar perubahan nilainya bisa terlihat di web browser.

Saya akan modifikasi kode kita sebelumnya dengan menambah satu perintah `console.log()` ke dalam method `render()`:

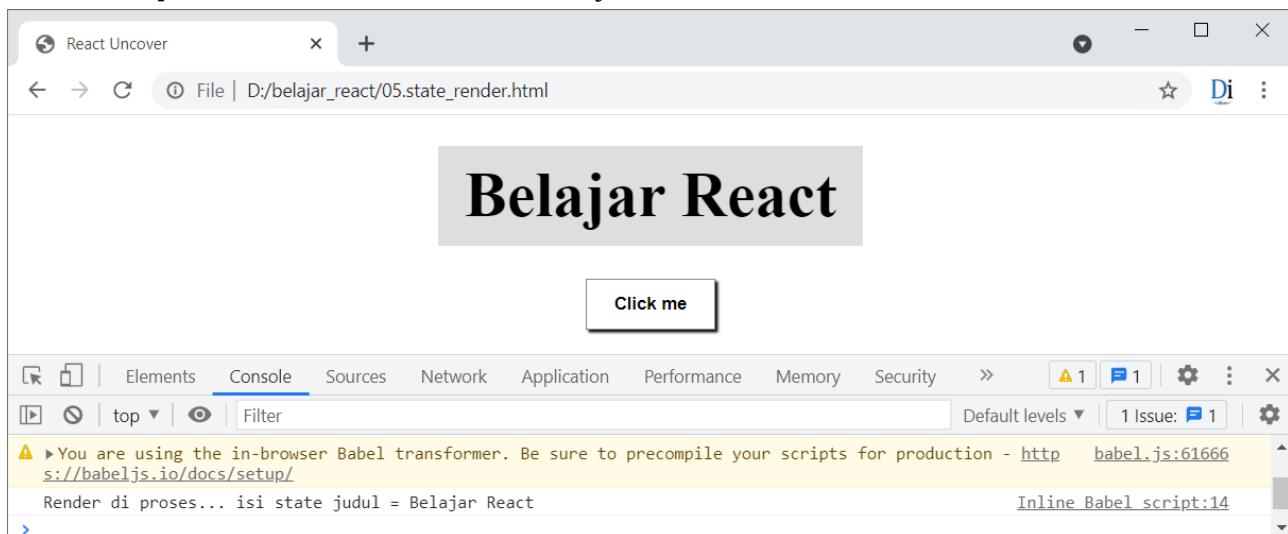
05.state_render.html

```

1 ...
2   render() {
3     console.log(`Render di proses... isi state judul = ${this.state.judul}`);
4     return (
5       <div>
6         <h1>{this.state.judul}</h1>
7       ...

```

Tambahan perintah ada di baris 3. Silahkan jalankan file ini dan cek isi tab Console:



Gambar: Perintah `console.log()` tampil saat proses render awal

Saat halaman di akses pertama kali, akan tampil teks "Render di proses... isi state judul = Belajar

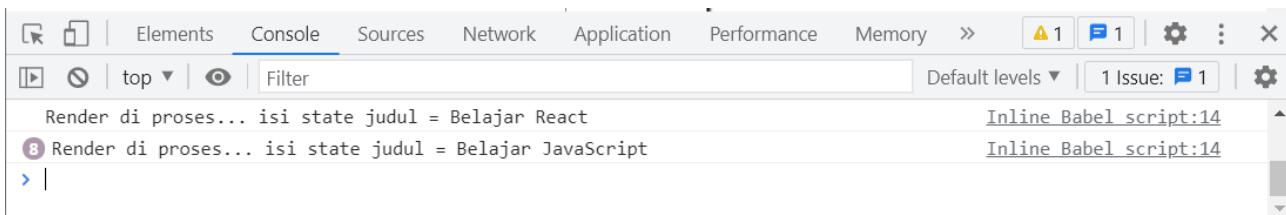
React". Teks ini berasal dari proses render awal komponen `MyApp`.

Sekarang coba klik tombol "Click me". Perhatikan begitu tombol di klik, akan tampil teks "Render di proses... isi state judul = Belajar JavaScript".

Kenapa teks ini bisa tampil? Padahal secara konsep seharusnya hanya kode yang ditulis ke dalam event handler-lah yang berjalan. Sedangkan perintah `console.log()` saya tulis ke dalam method `render()`, bukan di method `handleButtonClick()`.

Inilah bukti kalau method `render()` akan di jalankan ulang oleh React.

Silahkan klik tombol "Click me" beberapa kali, teks hasil perintah `console.log()` akan terus tampil. Artinya, setiap kali tombol di klik, method `render()` akan terus dijalankan ulang.



Gambar: Hasil tab Console

Jika terdapat teks yang sama, tab Console akan menggabung teks tersebut untuk menghemat tempat. Angka 8 di sisi kiri baris kedua menandakan ada 8 buah teks yang sama (saya sudah men-klik tombol "Click me" sebanyak 8 kali).

Tapi bukankah setelah klik ke-2, state `judul` akan terus di isi nilai yang sama, yakni "Belajar JavaScript", kenapa tetap di render ulang? Hal ini terjadi karena React menganggap nilai yang diinput adalah nilai baru (meskipun isi teksnya tetap sama).

Agar tidak terjadi proses render ulang seperti ini, kita bisa tambah sebuah kondisi `if` ke dalam method `handleButtonClick()`. Triknya, cek apakah state `judul` berisi teks selain "Belajar JavaScript". Jika `true`, ubah nilai state `judul`, jika `false`, maka tidak perlu jalankan perintah apapun:

06.state_render_if.html

```

1 ...
2   handleButtonClick = () => {
3     if (this.state.judul !== "Belajar JavaScript") {
4       this.setState({ judul: "Belajar JavaScript" });
5     }
6   }
7 ...

```

Dengan perubahan ini, ketika tombol "Click me" di klik untuk kedua, ketiga dan seterusnya, tidak terjadi proses re-render karena `this.setState()` tidak dijalankan.

Teknik optimasi seperti ini hanya bisa di pakai pada situasi tertentu saja. Karena bisa jadi kita malah ingin setiap kali tombol di klik, komponen App harus di render ulang.

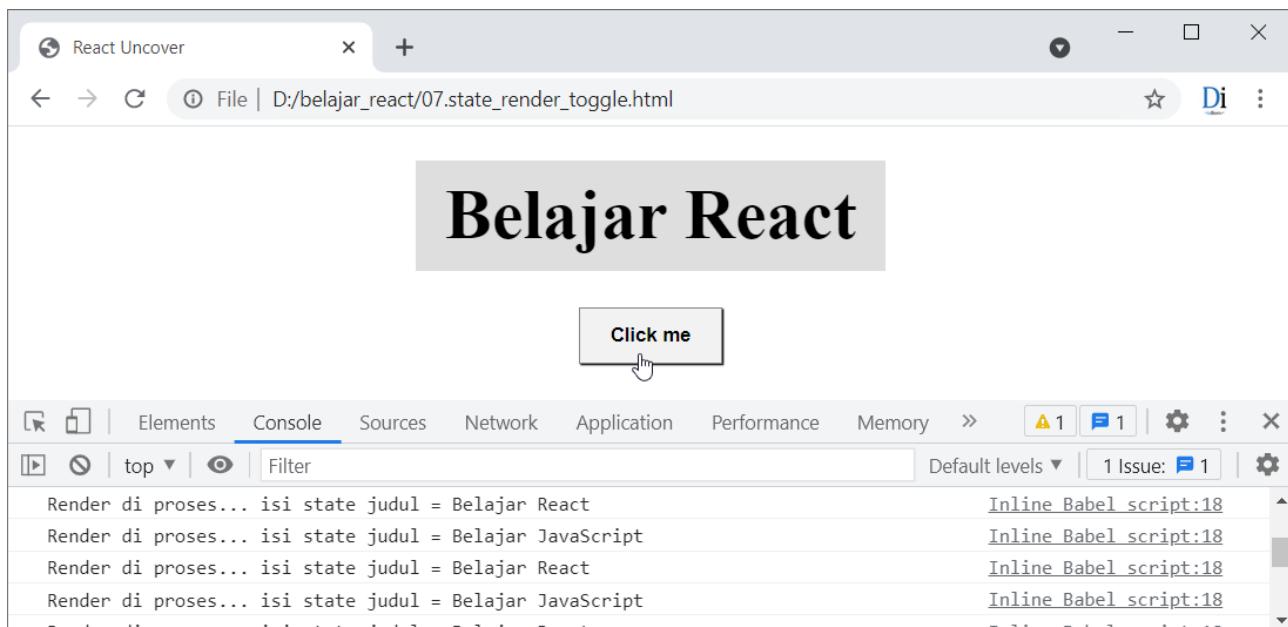
Misalnya, saya ingin tombol "Click me" menjadi toggle yang ketika di klik akan mengubah judul "Belajar React" menjadi "Belajar JavaScript" dan sebaliknya. Untuk keperluan ini, method `render()` harus selalu berjalan agar hasilnya bisa tampil di web browser:

07.state_render_toggle.html

```

1  ...
2  handleButtonClick = () => {
3      if (this.state.judul !== "Belajar JavaScript") {
4          this.setState({ judul: "Belajar JavaScript" });
5      } else {
6          this.setState({ judul: "Belajar React" });
7      }
8  }

```



Gambar: Tombol "Click me" menjadi toggle untuk menukar judul

Sekarang setiap kali tombol "Click me" di klik, judul akan terus berganti-ganti dari "Belajar React" menjadi "Belajar JavaScript" dan sebaliknya. Teks di tab Console juga akan terus berganti sebagai bukti kalau method `render()` terus berjalan.

Itulah prinsip kerja dari **React state**. Setiap kali nilainya berubah, method `render()` dari komponen tersebut akan di jalankan ulang.

8.3. Virtual DOM React

Dalam beberapa kesempatan saya pernah menyenggung kalau React menggunakan **virtual DOM** sebagai perantara antara kode program dengan DOM asli yang ada di web browser.

Virtual DOM ini digunakan React agar proses update element HTML berjalan lebih efisien. Di JavaScript native, meskipun perubahan data di element HTML masih sama seperti sebelumnya, element tersebut tetap dibuat ulang.

Untuk melihat bukti dari konsep ini, saya sudah siapkan kode yang mirip dengan praktik kita sebelumnya, tapi kali ini menggunakan JavaScript native (tanpa React):

08.change_value_js.html

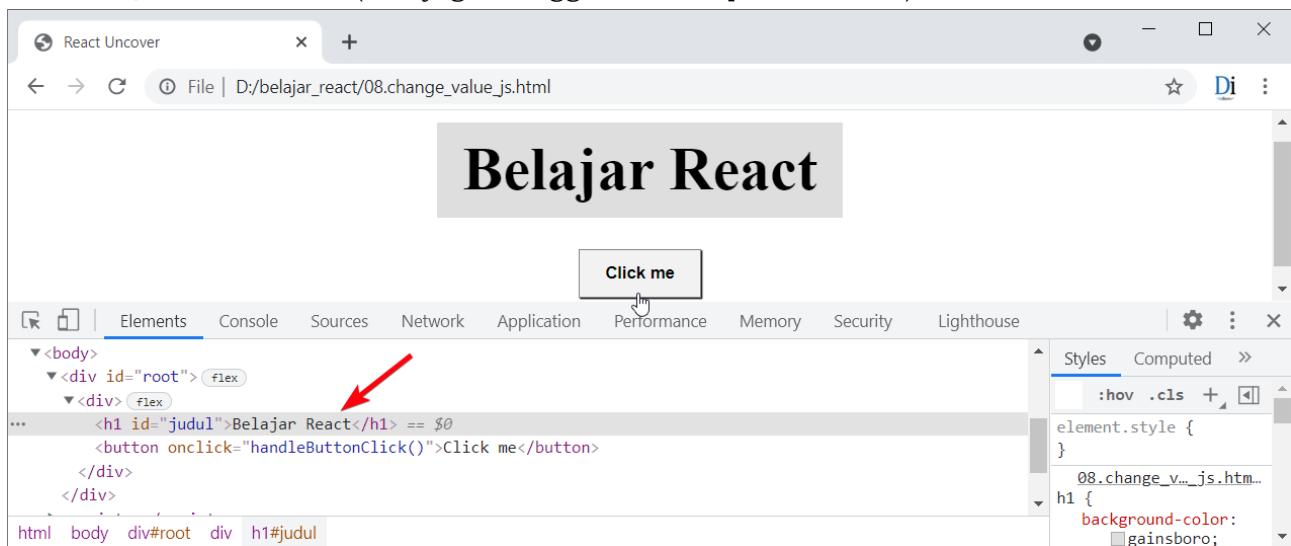
```

1 <body>
2   <div id="root">
3     <div>
4       <h1 id="judul">Belajar React</h1>
5       <button onclick="handleButtonClick()">Click me</button>
6     </div>
7   </div>
8
9   <script>
10    let h1Node = document.getElementById("judul");
11    const handleButtonClick = () => {
12      h1Node.innerHTML = "Belajar JavaScript";
13    };
14  </script>
15 </body>
```

Bagi yang pernah belajar JavaScript native, kemungkinan besar sudah bisa memahami maksud kode ini.

Event `onclick` di dalam tag `<button>` akan menjalankan fungsi `handleButtonClick()`. Pada saat tombol ini diklik, nilai atribut `innerHTML` milik `h1Node` akan diganti menjadi "Belajar JavaScript". Variabel `h1Node` sendiri merujuk ke tag `<h1 id="judul">`.

Silahkan buka web browser, masuk ke tab **Elements** di developer tools, lalu cari posisi tag `<h1>Belajar React</h1>` (bisa juga menggunakan inspect element):

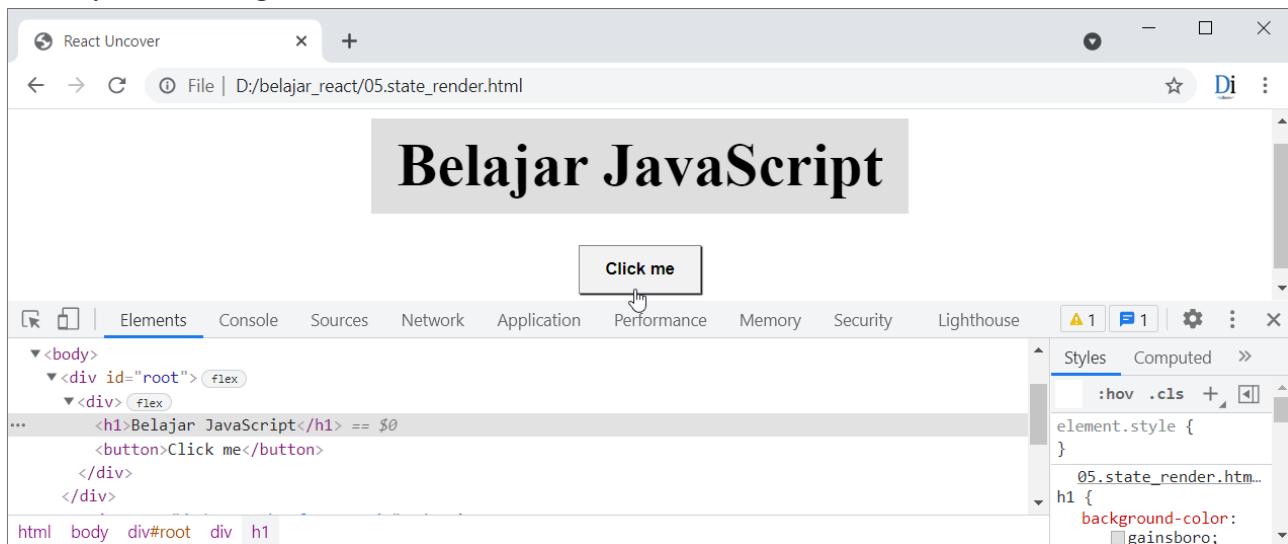


Gambar: Cari tag `<h1>Belajar React</h1>` di dalam tab Elements

Perhatikan bagian tag `<h1>` ini, lalu klik sekali tombol "Click me". Tag tersebut akan berkedip dan berubah menjadi `<h1>Belajar JavaScript</h1>`. Efek berkedip terjadi sebagai tanda ada perubahan element karena web browser mengganti isi teks lama menjadi teks baru.

Sekarang klik kembali tombol "Click me", ternyata tag `<h1>Belajar JavaScript</h1>` akan kembali berkedip meskipun isinya sama seperti sebelumnya. Setiap kali tombol di klik, element `<h1>` akan terus dibuat ulang. Perubahan struktur DOM seperti ini kurang efisien, karena toh isi teks tag `<h1>` masih tetap sama.

Sekarang silahkan buka kembali file `05.state_render.html`. Ini merupakan file yang kita pakai saat membahas proses re-render state dengan tambahan `console.log()`. Lakukan hal yang sama, yakni cek tag `<h1>Belajar React</h1>` di tab **Elements**, lalu klik tombol "Click me".



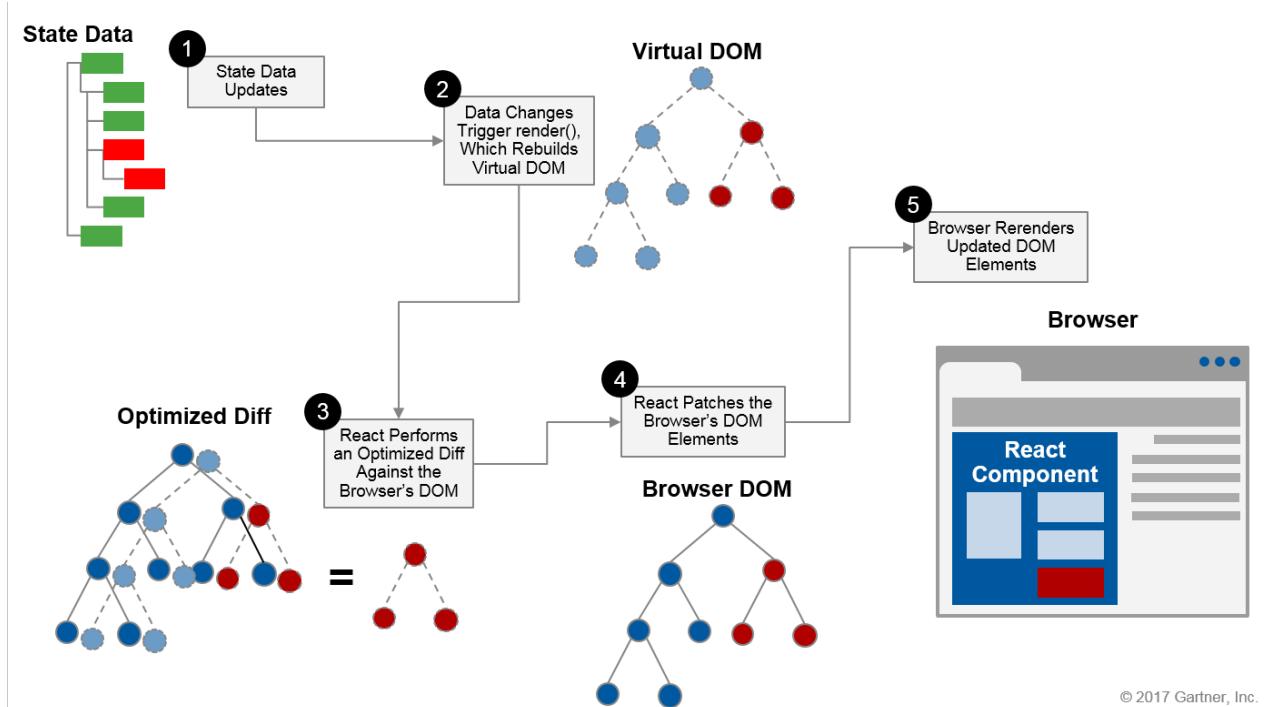
Gambar: Cari tag `<h1>Belajar React</h1>` di dalam tab Elements

Saat klik pertama, tag `<h1>` akan berkedip dan berubah menjadi `<h1>Belajar JavaScript </h1>`, ini sama di versi JavaScript native.

Akan tetapi begitu tombol di klik untuk kedua, ketiga, dst, tag `<h1>` tidak lagi berkedip. Di sini React melihat bahwa meskipun isi state berubah, tapi teksnya tetap sama dan tidak perlu mengubah struktur DOM.

Secara internal, proses re-render milik React tetap berjalan. Sebagai bukti, silahkan buka tab Console dan akan terlihat teks "Render di proses..." terus bertambah setiap kali tombol di klik.

Virtual DOM milik React memiliki peran penting dalam proses ini. Gambar berikut menjelaskan alur kerja dari virtual DOM:



Gambar: Ilustrasi proses update element di dalam React (sumber gambar: gartner.com)

Virtual DOM bisa diibaratkan sebagai "cloningan" dari struktur DOM asli yang ada di web browser.

Pada saat terjadi perubahan nilai state (1), React akan menjalankan method `render()` milik komponen tersebut dan mengupdate struktur virtual DOM (2). Virtual DOM yang baru saja di update akan dibandingkan dengan virtual DOM sebelum proses update untuk mencari apakah ada perbedaan atau tidak. Langkah ini dikenal dengan istilah "*diffing*" (3).

Jika terdapat perbedaan, maka perbedaan itulah (dan hanya itu saja) yang di update ke struktur DOM asli (4). Terakhir, DOM akan di tampilkan ke web browser (5).

Inti dari proses ini ada di "*diffing*". Jika pada saat membandingkan virtual DOM ternyata tidak ada perubahan, maka proses selesai sampai di situ, struktur DOM asli tidak perlu di update lagi. Karena itulah dalam contoh versi React, tag `<h1>` tidak berkedip.

Penggunaan virtual DOM akan makin terasa saat kita punya data berulang seperti tag ``. Jika ada 100 list dan ingin mengubah nilai pertama saja, maka seluruh list tidak perlu dibuat ulang, cukup list yang berubah saja. Syaratnya, setiap list harus memiliki atribut `key` agar proses "*diffing*" bisa berjalan dengan efisien.

8.4. State di Class Component

Pada saat membahas pengertian state, kita sudah melihat cara membuat state dalam class component, yakni dengan cara mengisi object ke dalam property `this.state`. Untuk mengupdate nilai state, gunakan fungsi `this.setState()`.

Posisi ideal pembuatan state adalah di constructor. Meskipun jarang dilakukan (dan juga bukan *best practice*), pembuatan state bisa ditulis di luar constructor. Berikut contohnya:

09.state_declaration.html

```

1  class MyApp extends React.Component {
2
3      handleButtonClick = () => {
4          this.state = { judul: "Belajar React" };
5          document.getElementById("judul").innerHTML = this.state.judul;
6      }
7
8      render() {
9          return (
10             <div>
11                 <h1 id="judul"></h1> // {this.state.judul} tidak bisa diakses di sini
12                 <button onClick={this.handleButtonClick}>Click me</button>
13             </div>
14         )
15     }
16 }
17
18 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Di sini, pembuatan state ada di dalam method `handleButtonClick()` di baris 4. Kode ini tidak akan error dan bisa berjalan. Namun karena state `judul` baru ada setelah tombol "Click me" di klik, maka kita tidak bisa mengaksesnya di baris 11. Sebab, saat proses render pertama kali property `this.state.judul` belum terdefinisi.

Untuk mengakalinya, pengisian judul tag `<h1>` terpaksa saya lakukan dengan mengakses DOM secara langsung lewat property `innerHTML` seperti di baris 5.

Umumnya, proses update state dilakukan dari dalam method *event handler* seperti `handleButtonClick()`. Namun karena event handler bisa ditulis secara inline ke dalam atribut, maka perintah `setState()` juga bisa dilakukan dari sana:

10.state_in_event_inline.html

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { judul: "Belajar React" };
5      }
6
7      render() {
8          return (
9              <div>
10                 <h1>{this.state.judul}</h1>
11                 <button onClick={() => this.setState({ judul: "Belajar JavaScript" })}>
12                     Click me
13                 </button>
14             </div>

```

```

15      )
16    }
17  }
18
19 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Sekarang proses update judul tag `<h1>` langsung saya tulis sebagai inline function di baris 11. Penulisan seperti ini hanya cocok untuk aplikasi yang sederhana saja.

Satu hal yang harus dihindari adalah menjalankan fungsi `this.setState()` secara langsung di dalam method `render()`. Ini bisa membuat *infinity loop* karena re-render akan berjalan terus-menerus. Terkecuali jika `this.setState()` diakses dari dalam event seperti contoh di atas.

Mengakses State Setelah di Update

Ketika mempelajari state lebih lanjut dan membuat beberapa contoh kode program, saya mengalami beberapa kendala yang butuh waktu berjam-jam untuk mencari apa yang salah. Diantaranya terkait pengaksesan nilai state setelah di update.

Ketika nilai state diubah menggunakan perintah `this.setState()`, nilai state **tidak berubah di dalam method** sebelum terjadi proses re-render. Kita akan bahas dengan contoh kode berikut:

```

11.state_access_problem.html

1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { judul: "Belajar React" };
5    }
6
7    handleButtonClick = () => {
8      console.log(`Isi state judul, sebelum di set = ${this.state.judul}`);
9      this.setState({ judul: "Belajar JavaScript" });
10     console.log(`Isi state judul, setelah di set = ${this.state.judul}`);
11   }
12
13  render() {
14    console.log(`Render di proses... isi state judul = ${this.state.judul}`);
15    return (
16      <div>
17        <h1>{this.state.judul}</h1>
18        <button onClick={this.handleButtonClick}>Click me</button>
19      </div>
20    )
21  }
22}
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

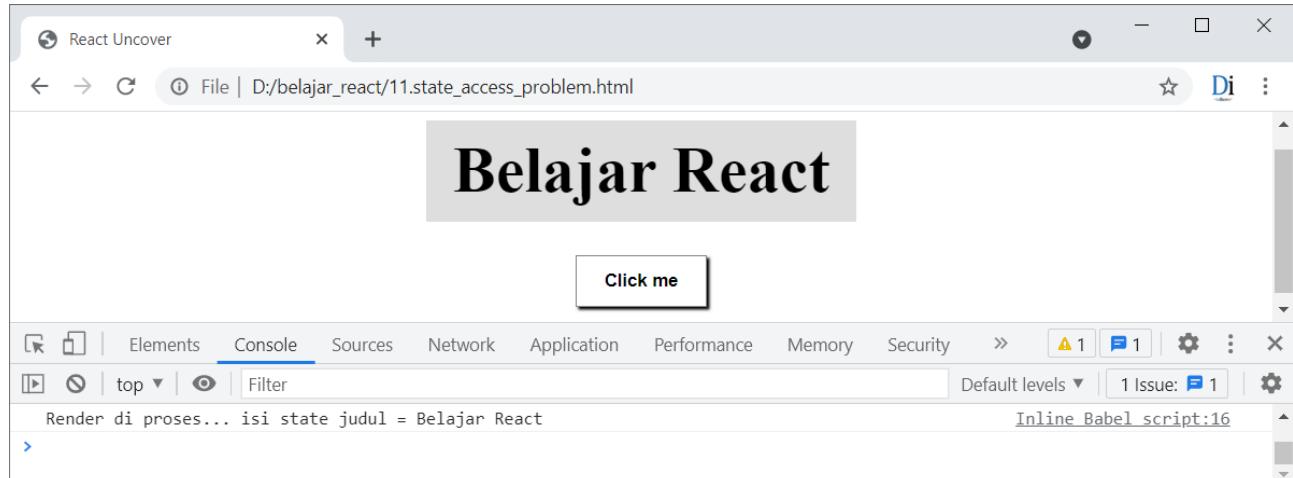
```

Kode ini masih sama seperti contoh kita sebelumnya. Tombol "Click me" memiliki event click yang akan mengubah judul tag `<h1>`. Tambahannya ada di baris 8, 10 dan 14 berupa perintah

`console.log()` yang saya pakai untuk melihat apa isi state judul di 3 posisi berikut:

- Perintah `console.log()` di baris 8 akan berjalan setelah event `click` terjadi, namun sebelum perintah `this.setState()` di proses.
- Perintah `console.log()` di baris 10 akan berjalan setelah perintah `this.setState()` di proses dan masih di dalam method `handleButtonClick()`.
- Perintah `console.log()` di baris 14 akan berjalan saat proses render.

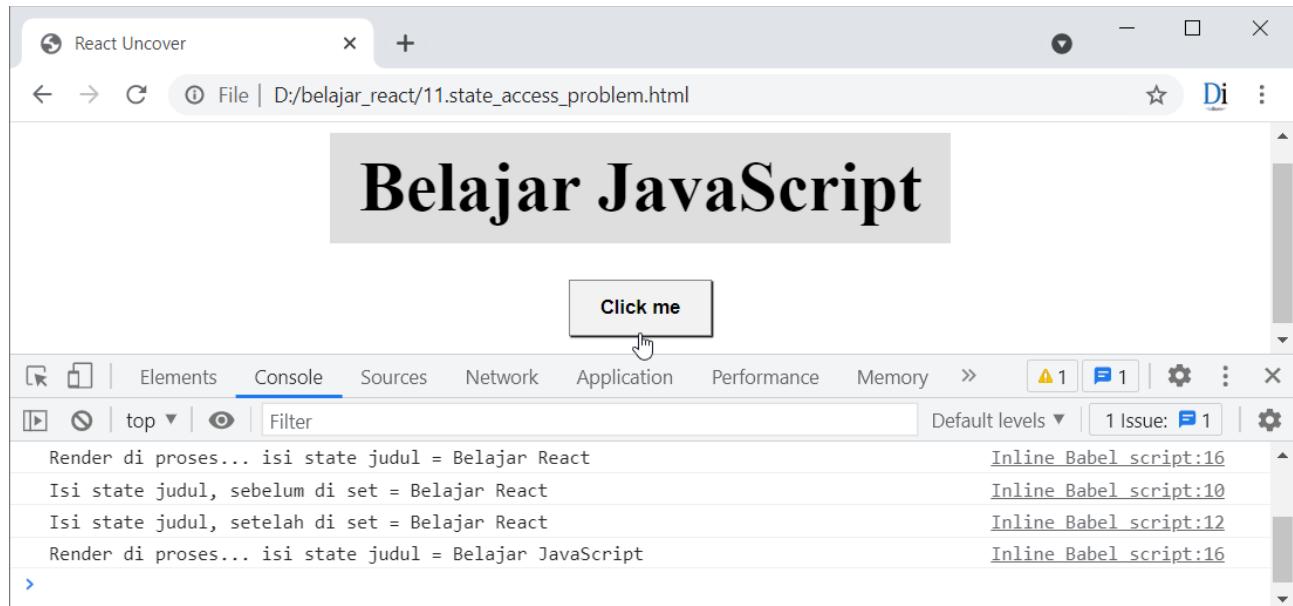
Jalankan kode di atas, lalu langsung lihat tab Console:



Gambar: Isi state judul saat proses render pertama

Teks "Render di proses... isi state judul = Belajar React" berasal dari perintah `console.log()` di baris 14. State `judul` berisi teks "Belajar React" sebagai hasil proses inisialisasi di dalam constructor. Tidak ada masalah.

Sekarang klik tombol "Click me" sekali:



Gambar: Isi state judul setelah tombol "Click me" di klik

Hasilnya, tampil 3 baris tambahan.

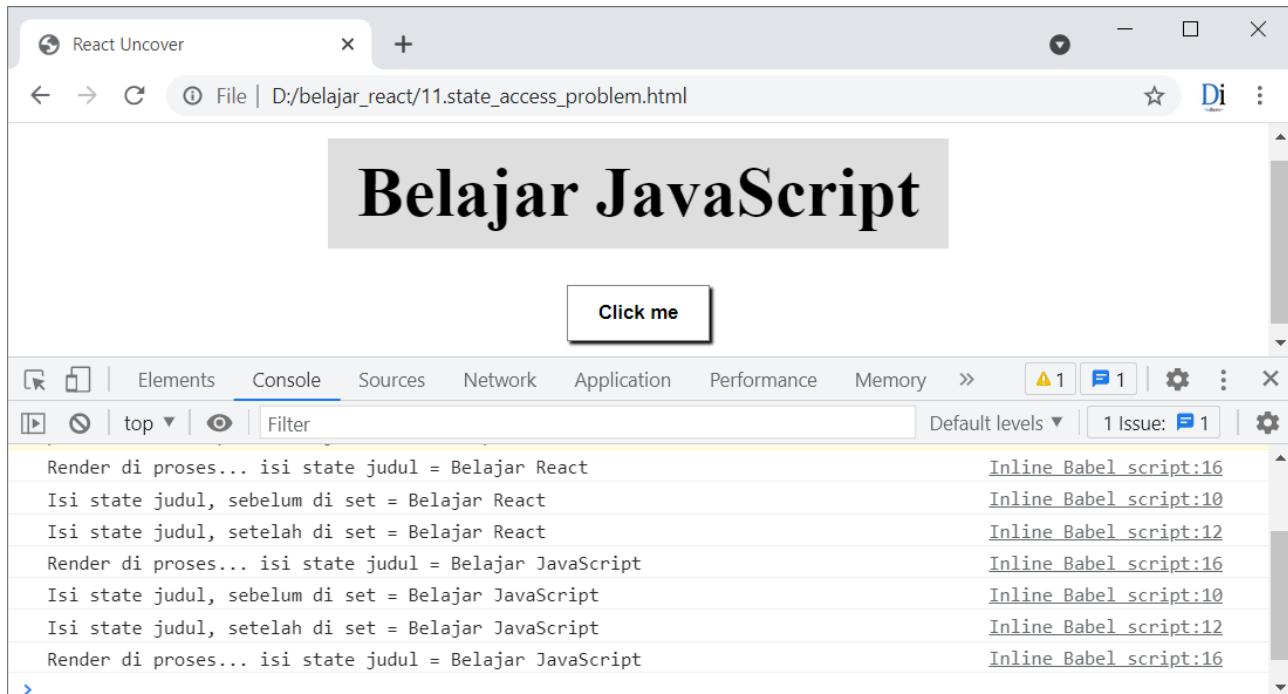
Baris kedua berasal dari perintah `console.log()` di dalam method `handleButtonClick()`, posisinya tepat sebelum fungsi `this.setState()` berjalan. Terlihat bahwa state `judul` masih berisi string "Belajar React". Ini juga sesuai dengan konsep dasar kita karena nilai state `judul` memang belum diubah.

Baris ketiga juga berasal dari dalam method `handleButtonClick()`, tapi dijalankan setelah fungsi `this.setState()`. Hasilnya? Ternyata state `judul` tetap berisi "Belajar React", padahal baru saja kita ubah!

Di sinilah saya sempat bingung apa yang salah dari kode ini. Apakah perintah `this.setState()` tidak berefek?

Hasil `console.log()` di baris ke-4 mengkonfirmasi bahwa nilai state `judul` sebenarnya sudah berubah. Di situ tampil teks "*Render di proses... isi state judul = Belajar JavaScript*", yang berarti nilai state `judul` sudah bukan "Belajar React" lagi, tapi sudah berganti menjadi "Belajar JavaScript".

Sekarang klik sekali lagi tombol "Click me":



Gambar: Isi state judul sudah berubah

Tiga baris terakhir memperlihatkan isi state `judul` sudah berubah menjadi "Belajar JavaScript".

Dari percobaan ini bisa diambil kesimpulan bahwa jika kita mengakses nilai state di dalam method yang menjalankan perintah `this.setState()`, hasil yang didapat masih berasal dari nilai state sebelumnya dan **belum berubah**. Setelah 1 kali render, barulah nilai state ikut berubah.

Akan tetapi jika state diakses di luar method yang menjalankan `this.setState()`, maka itu sudah mendapat nilai baru, tidak perlu menunggu proses re-render terlebih dahulu. Inilah yang menjelaskan kalau di method `render()`, nilai `this.state.judul` sudah berisi "Belajar JavaScript".

Jadi bagaimana cara mengatasi masalah ini?

Salah satu solusi adalah dengan langsung mengakses nilai yang akan kita input saja, tidak perlu lewat state. Method `handleButtonClick()` bisa diubah sebagai berikut:

12.state_access_solution.html

```

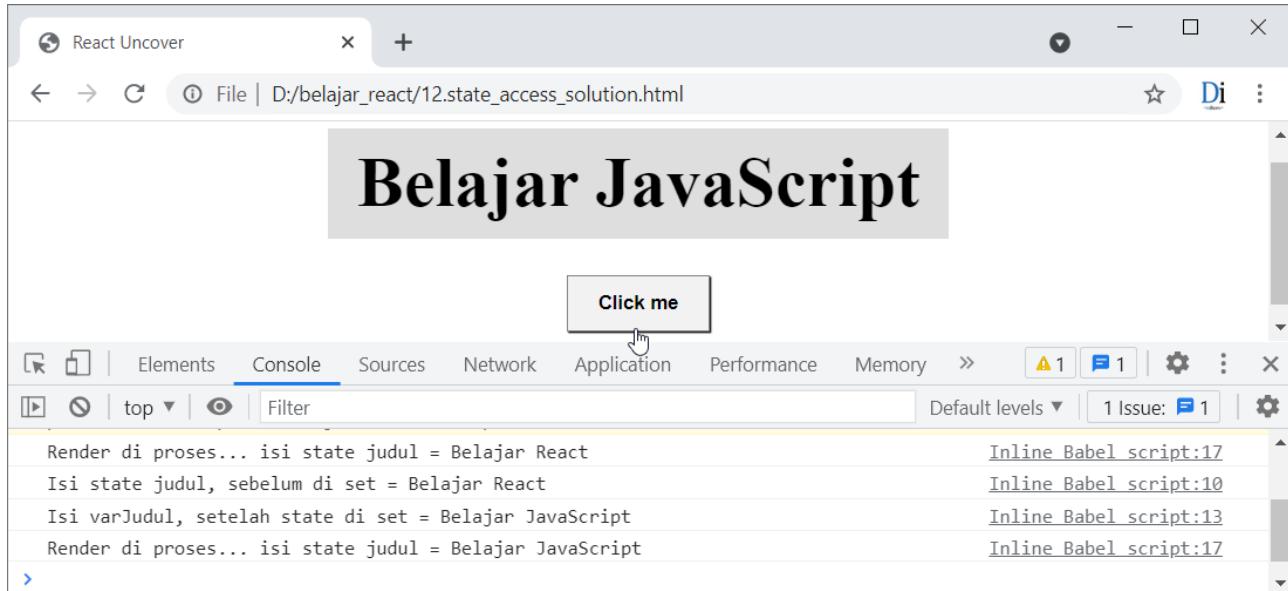
1 ...
2   handleButtonClick = () => {
3     console.log(`Isi state judul, sebelum di set = ${this.state.judul}`);
4     let varJudul = "Belajar JavaScript";
5     this.setState({ judul: varJudul });
6     console.log(`Isi varJudul, setelah state di set = ${varJudul}`);
7   }
8 ...

```

Di baris 4 saya membuat variabel `varJudul` dan mengisinya dengan string "Belajar JavaScript". Inilah nilai yang akan di input ke dalam method `this.setState()`.

Jika ingin mengakses nilai state judul dari dalam method `handleButtonClick()`, cukup akses variabel `varJudul` ini, tidak lewat `this.state.judul` (baris 6). Sehingga akan langsung dapat nilai terbaru.

Dengan perubahan ini, ketika tombol "Click me" di klik, perintah `console.log()` yang ditulis setelah fungsi `this.setState()` sudah langsung memberikan nilai "Judul JavaScript", tidak perlu menunggu proses re-render terlebih dahulu.



Gambar: Hasil `console.log()` di baris 3 sudah menampilkan nilai state terbaru

Mengubah nilai state sebenarnya juga bisa lewat cara manual, yakni dengan perintah `this.state = {<nama_state>:<nilai>}`. Ini sama seperti mengisi property biasa.

Akan tetapi jika ditulis seperti itu, proses re-render tidak akan terjadi. Oleh karena itu selalu gunakan method `this.setState({<nama_state>:<nilai>})` untuk mengupdate nilai state.

Membuat Beberapa State

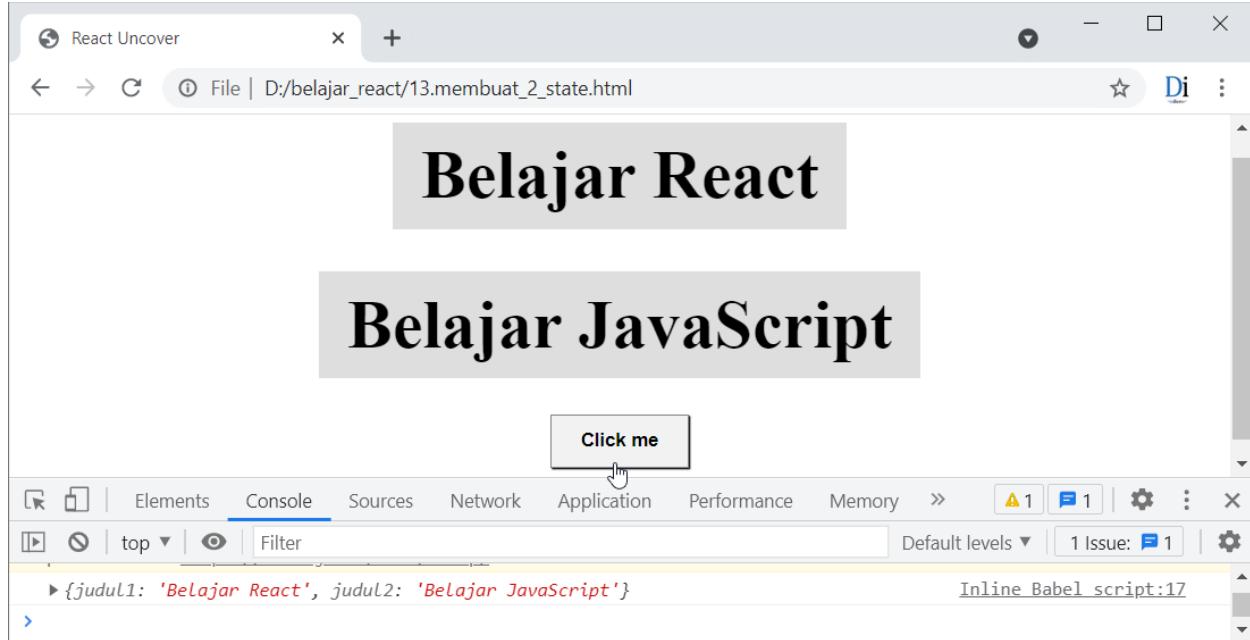
Kita juga bisa membuat beberapa state sekaligus. Caranya, tambah property baru ke dalam object yang menjadi nilai `this.state` dengan format berikut:

```
this.state = {
  nama_state_1: nilai_state_1,
  nama_state_2: nilai_state_2,
  nama_state_3: nilai_state_3,
  ... : ...,
};
```

Berikut contoh prakteknya:

```
13.membuat_2_state.html

1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        judul1: "Belajar React",
6        judul2: "Belajar JavaScript",
7      };
8    }
9
10   handleButtonClick = () => {
11     this.setState({ judul1: "Belajar React State" });
12   }
13
14   render() {
15     console.log(this.state);
16     return (
17       <div>
18         <h1>{this.state.judul1}</h1>
19         <h1>{this.state.judul2}</h1>
20         <button onClick={this.handleButtonClick}>Click me</button>
21       </div>
22     )
23   }
24 }
25
26 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

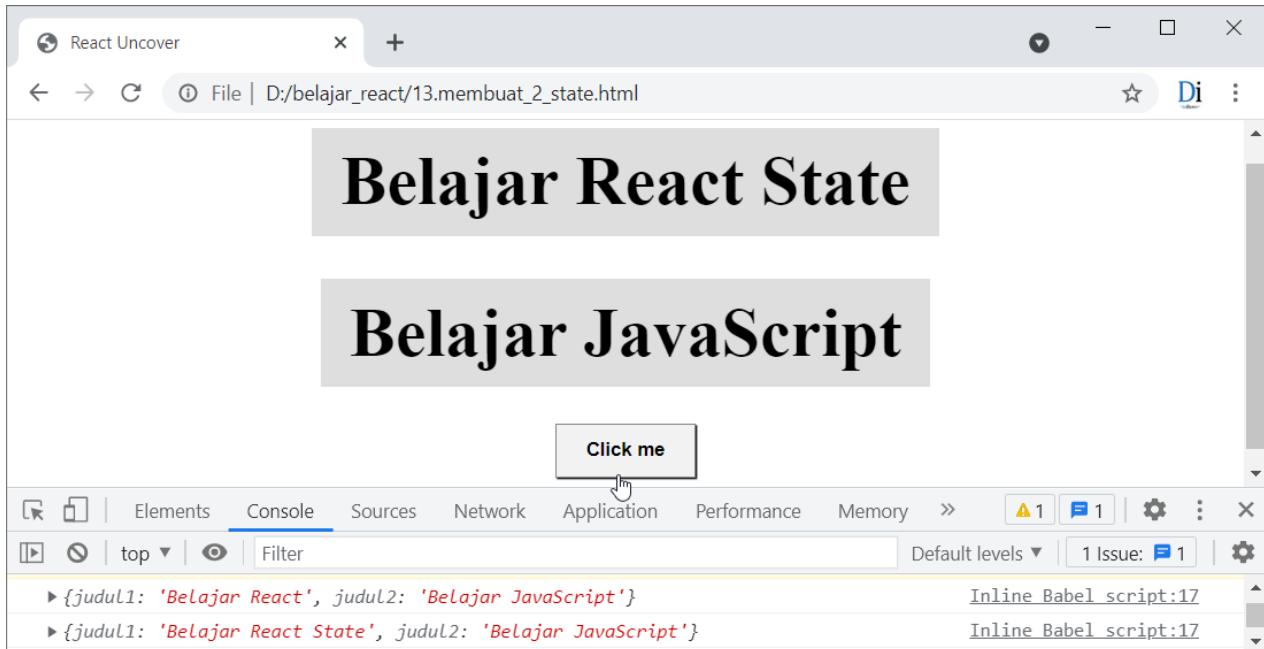


Gambar: Tampilan 2 judul yang berasal dari 2 state

Di baris 4-6 saya membuat 2 buah state bernama judul1 dan judul2. Ke dalam judul1 di isi nilai awal "Belajar React", dan untuk state judul2 di isi "Belajar JavaScript". Kedua state ditampilkan ke dalam tag <h1> di baris 18 dan 19.

Untuk tambahan informasi, perintah `console.log(this.state)` di baris 15 akan menampilkan isi state setiap kali method `render()` berjalan.

Pada saat tombol "Click me" di klik, method `handleButtonClick()` di baris 10-12 akan mengubah nilai state judul1 menjadi "Belajar React State" menggunakan perintah `this.setState({ judul1: "Belajar React State" })`. Berikut hasilnya:



Gambar: Judul pertama sudah berubah menjadi "Belajar React State"

Di class component, ketika kita mengupdate nilai state dengan perintah `this.setState()`, state yang namanya tidak ditulis tidak ikut berubah (nilainya tetap). Dalam contoh di atas saya hanya mengubah isi state `judul1`, maka nilai state `judul2` tidak diganggu dan tetap berisi string "Belajar React".

Jika kita juga ingin mengubah nilai `judul1` dan `judul2` sekaligus, tinggal tulis kedua state dalam method `this.setState()` seperti contoh berikut:

14.mengupdate_2_state.html

```

1 ...
2     handleButtonClick = () => {
3         this.setState({
4             judul1: "Belajar React State",
5             judul2: "Belajar Programming"
6         });
7     }
8 ...

```

Dengan kode ini, kedua judul akan berubah saat tombol "Click me" di klik.

Mengisi Object Sebagai Nilai State

Untuk aplikasi yang kompleks, kadang kita ingin mengisi object sebagai nilai state. Tujuannya agar setiap state tersusun atas kelompok-kelompok tertentu. Berikut salah satu praktek dari konsep ini:

15.state_object_value.html

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = {
5              judul: {
6                  satu: "Belajar React",
7                  dua: "Belajar JavaScript",
8              }
9          };
10     }
11
12     render() {
13         console.log(this.state);
14         return (
15             <div>
16                 <h1>{this.state.judul.satu}</h1>
17                 <h1>{this.state.judul.dua}</h1>
18             </div>
19         )
20     }
21 }
22
23 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Perhatikan cara pembuatan state di baris 4-9. State yang dibuat hanya 1, yakni `judul`. Akan tetapi nilai state berbentuk object dengan 2 buah property: `satu` dan `dua`.

Untuk menampilkan isi state, sambung menggunakan tanda titik sebagaimana cara mengakses property object, yaitu dengan perintah `{this.state.judul.satu}` dan `{this.state.judul.dua}` seperti di baris 16-17.

Dengan mengisi nilai object, kita bisa membuat struktur state yang lebih kompleks. Akan tetapi ada masalah dengan proses update state. Berikut contoh kasusnya:

```
16.state_object_value_problem.html

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = {
5              judul: {
6                  satu: "Belajar React",
7                  dua: "Belajar JavaScript",
8              }
9          };
10     }
11
12     handleButtonClick = () => {
13         this.setState({ judul: { satu: "Belajar React State" } });
14     }
15
16     render() {
17         console.log(this.state);
18         return (
19             <div>
20                 <h1>{this.state.judul.satu}</h1>
21                 <h1>{this.state.judul.dua}</h1>
22                 <button onClick={this.handleButtonClick}>Click me</button>
23             </div>
24         )
25     }
26 }
27
28 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Kali ini saya kembali menambah tombol "Click me" yang akan menjalankan method `handleButtonClick()` ketika di klik. Di dalam *event handler*, isi state diganti dengan perintah di baris 13. Berikut hasilnya:



Gambar: Tampilan setelah tombol "Click me" di klik

Ternyata perintah `this.setState({ judul: { satu: "Belajar React State" } })` akan menghapus nilai state `this.state.judul.dua`. Ini terjadi karena method `this.setState()` hanya mempertahankan nilai state untuk "layer" pertama saja. Jika nilai state sudah berbentuk object, seluruh object akan dibuat ulang. Efeknya, nilai state yang tidak ditulis akan terhapus.

Terdapat beberapa solusi yang bisa kita pakai. Pertama, bisa dengan menulis semua nilai state:

17.state_object_value_solution_1.html

```

1 ...
2   handleButtonClick = () => {
3     this.setState({
4       judul: {
5         satu: "Belajar React State",
6         dua: "Belajar JavaScript",
7       }
8     });
9   }
10 ...
  
```

Meskipun state yang ingin diubah hanya `judul.satu` saja, tapi state `judul.dua` tetap harus ditulis agar nilainya tidak hilang. Ini menjadi cara paling sederhana namun kurang praktis jika kita memiliki object dengan property yang cukup banyak.

Solusi kedua, buat variabel bantu:

18.state_object_value_solution_2.html

```

1 ...
2   handleButtonClick = () => {
3     let judulObj = this.state.judul;
4     judulObj.satu = "Belajar React State";
5     this.setState(judulObj);
6   }
7 ...
  
```

Di baris 3 saya membuat variabel `judulObj` dan mengisinya dengan `this.state.judul`. Ini

berarti semua nilai `state.judul` sudah ter-copy. Kemudian di baris 4 property `judulObj.satu` di isi dengan string "Belajar React State". Perintah ini akan menimpa nilai yang ada sebelumnya.

Terakhir, variabel `judulObj` diinput ke dalam method `this.setState()`. Hasilnya, nilai state `this.state.judul.satu` sudah berubah, namun tidak mengganggu isi `this.state.judul.dua`.

Solusi ketiga, manfaatkan *spread operator* "...":

19.state_object_value_solution_3.html

```

1 ...
2   handleButtonClick = () => {
3     this.setState({
4       judul: { ...this.state.judul, satu: "Belajar React State" }
5     });
6   }
7 ...

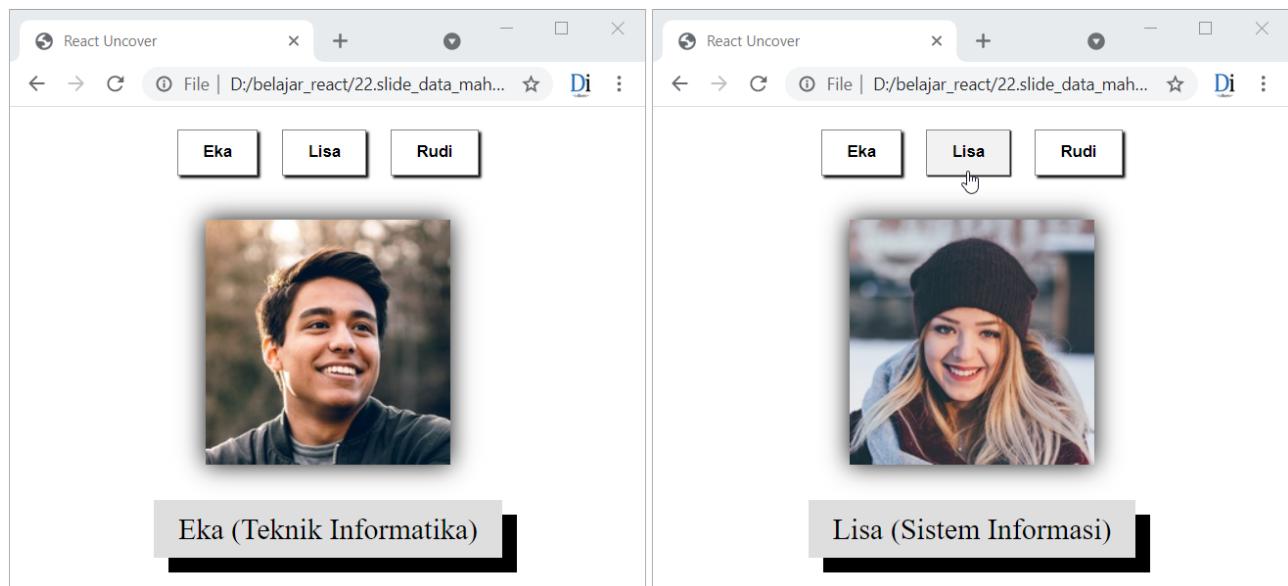
```

Perintah `...this.state.judul` akan "menyebarluaskan" semua isi `state.judul` yang sudah ada. Dengan demikian kita tidak perlu menulis semua property satu per satu. Lalu untuk state yang ingin di update, tinggal tulis dan isi dengan nilai baru.

Membuat nilai state sebagai object cukup sering dipakai pada pemrosesan form. Lebih lanjut tentang ini akan kita bahas dalam bab terpisah.

8.5. Mini Project: Slide Data Mahasiswa

Agar lebih memahami konsep penggunaan state di React, kita masuk ke sebuah mini project sederhana. Project ini akan membuat *slide data mahasiswa* dengan tampilan berikut:



Gambar: Tampilan akhir mini project "slide data mahasiswa"

Di bagian atas terdapat 3 tombol nama mahasiswa, yang ketika di klik akan menampilkan foto serta identitas dari mahasiswa tersebut. Kita akan bahas secara bertahap.

Pertama, silahkan pelajari sebentar maksud dari kode program berikut:

20.slide_data_mahasiswa_1.html

```

1  const mahasiswa = [
2    {
3      id: "01",
4      nama: "Eka",
5      jurusan: "Teknik Informatika",
6      pasFoto: "people1.jpg"
7    },
8    {
9      id: "02",
10     nama: "Lisa",
11     jurusan: "Sistem Informasi",
12     pasFoto: "people2.jpg"
13   },
14   {
15     id: "03",
16     nama: "Rudi",
17     jurusan: "Teknik Elektro",
18     pasFoto: "people4.jpg"
19   }
20 ];
21
22 class MyApp extends React.Component {
23   constructor() {
24     super()
25     this.state = {
26       nama: mahasiswa[0].nama,
27       jurusan: mahasiswa[0].jurusan,
28       pasFoto: mahasiswa[0].pasFoto,
29     };
30   }
31
32   handleMahasiswa0 = () => {
33     this.setState({
34       nama: mahasiswa[0].nama,
35       jurusan: mahasiswa[0].jurusan,
36       pasFoto: mahasiswa[0].pasFoto
37     });
38   }
39
40   handleMahasiswa1 = () => {
41     this.setState({
42       nama: mahasiswa[1].nama,
43       jurusan: mahasiswa[1].jurusan,
44       pasFoto: mahasiswa[1].pasFoto
45     });
46   }
47 
```

```

48     handleMahasiswa2 = () => {
49       this.setState({
50         nama: mahasiswas[2].nama,
51         jurusan: mahasiswas[2].jurusan,
52         pasFoto: mahasiswas[2].pasFoto
53       });
54     }
55
56   render() {
57     console.log(this.state);
58     return (
59       <div>
60         <button onClick={this.handleMahasiswa0}>{mahasiswas[0].nama}</button>
61         <button onClick={this.handleMahasiswa1}>{mahasiswas[1].nama}</button>
62         <button onClick={this.handleMahasiswa2}>{mahasiswas[2].nama}</button>
63       </div>
64     )
65   }
66 }
67
68 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Di baris 1-20 terdapat 3 data mahasiswa yang disimpan ke dalam konstanta `mahasiswas`. Data ini tersusun dalam bentuk array dari object yang sudah beberapa kali kita pakai.

Lompat ke method `render()` dari komponen `MyApp`, di baris 57 terdapat perintah `console.log(this.state)`, fungsinya sekedar mengetahui apa isi state pada setiap render. Informasi ini sangat bermanfaat untuk proses *debugging*. Jika hasil kode program tidak sesuai, tinggal buka tab Console untuk melihat apa isi state saat ini.

Masuk ke struktur JSX, terdapat 3 buah tag `<button>` yang masing-masingnya memiliki atribut `onClick`. Setiap tombol mendapat *event handler* yang berbeda-beda. Isi teks button diambil langsung dari array `mahasiswas`, sehingga akan diproses menjadi kode berikut:

```

<button>Eka</button>
<button>Lisa</button>
<button>Rudi</button>

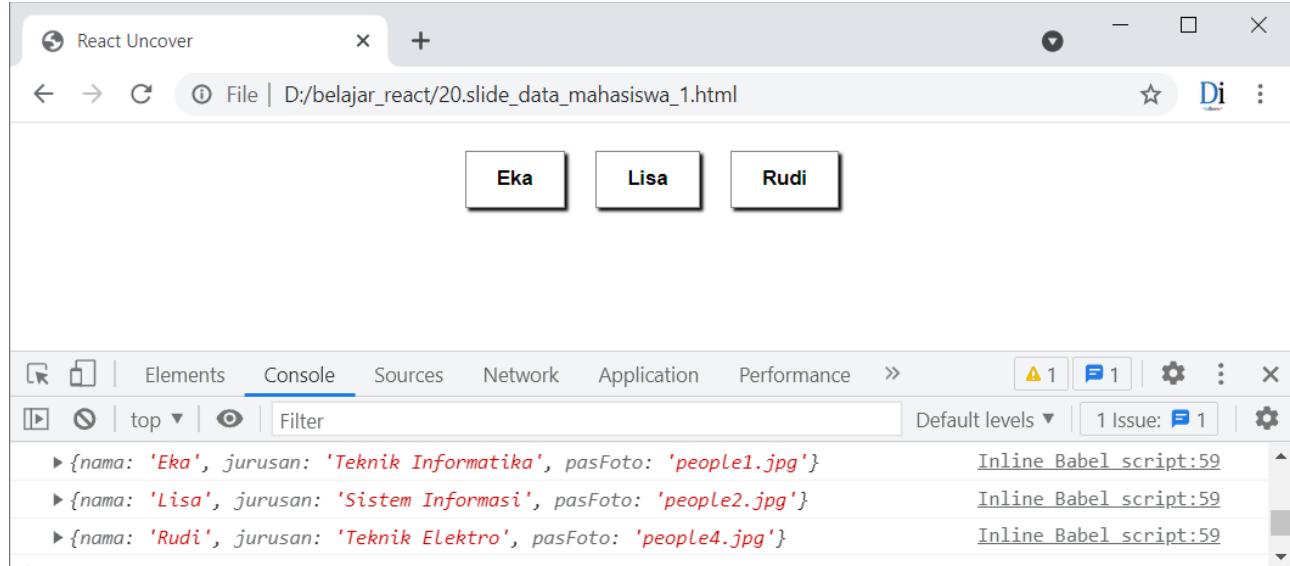
```

Lanjut ke constructor komponen `MyApp` di baris 23-30, saya membuat 3 state, yakni `nama`, `jurusan` dan `pasFoto`. Untuk `id` tidak di perlukan saat ini, tapi boleh saja ditambah. Nilai awal state diambil dari index 0 array `mahasiswas` (data pertama).

Tiga method lain di dalam `MyApp` berfungsi sebagai *event handler* tombol. Isinya akan mengganti nilai state berdasarkan urutan mahasiswa. Jika method `handleMahasiswa0()` di jalankan, maka ganti semua isi state dengan data `mahasiswas` di index 0. Jika `handleMahasiswa1()` berjalan, ganti semua isi state dengan data `mahasiswas` di index 1, dst.

Jalankan file di atas, lalu klik tombol secara bergantian dan cek apakah isi state di tab console juga ikut berubah:

React State



Gambar: Klik setiap tombol untuk memeriksa nilai state

Untuk membuat tampilan di atas, saya menambah sedikit kode CSS agar tombol jadi lebih rapi:

```
1 #root {
2   display: flex;
3   justify-content: center;
4 }
5
6 button {
7   background-color: white;
8   cursor: pointer;
9   padding: 10px 20px;
10  font-weight: bold;
11  border: 1px solid gray;
12  box-shadow: 2px 2px 2px black;
13  margin: 10px;
14 }
15
16 button:hover {
17  box-shadow: 1px 1px 1px black;
18  background-color: #f1f1f1;
19 }
```

Silahkan tambah kode ini ke dalam tag <style> di bagian <head>.

Data mahasiswa sudah berhasil diakses. Selanjutnya tinggal menginput data tadi ke dalam struktur JSX. Karena kita butuh menampilkan gambar, saya sudah siapkan folder **img** yang berisi beberapa gambar avatar. Gambar ini tersedia juga di file **belajar_react.zip** atau bebas jika ingin diganti dengan gambar lain.

Kita hanya perlu memodifikasi isi method `render()`:

21.slide_data_mahasiswa_2.html

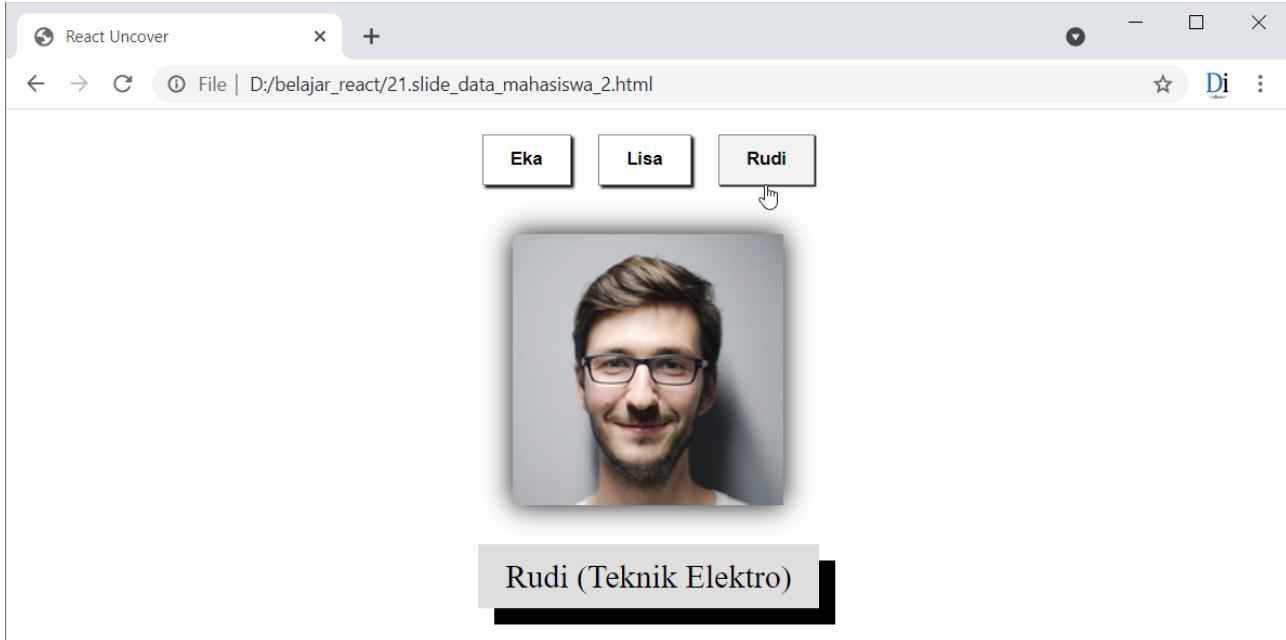
```
1 render() {
```

React State

```
2   return (
3     <div>
4       <button onClick={this.handleMahasiswa0}>{mahasiswa[0].nama}</button>
5       <button onClick={this.handleMahasiswa1}>{mahasiswa[1].nama}</button>
6       <button onClick={this.handleMahasiswa2}>{mahasiswa[2].nama}</button>
7
8       <figure>
9         <img src={"img/" + this.state.pasFoto} alt={this.state.nama} />
10        <figcaption>{this.state.nama} ({this.state.jurusan})</figcaption>
11      </figure>
12
13    </div>
14  )
15 }
```

Kembali, agar tampilannya lebih cantik, perlu sedikit bantuan kode CSS berikut:

```
1  #root {
2    display: flex;
3    justify-content: center;
4  }
5
6  div {
7    text-align: center;
8  }
9
10 button {
11   background-color: white;
12   cursor: pointer;
13   padding: 10px 20px;
14   font-weight: bold;
15   border: 1px solid gray;
16   box-shadow: 2px 2px 2px black;
17   margin: 10px;
18 }
19
20 button:hover {
21   box-shadow: 1px 1px 1px black;
22   background-color: #f1f1f1;
23 }
24
25 img {
26   box-shadow: 0px 0px 20px 0px black;
27   margin: 10px;
28 }
29
30 figcaption {
31   text-align: center;
32   background-color: gainsboro;
33   font-size: 1.5em;
34   padding: 10px 20px;
35   margin: 15px 0;
36   box-shadow: 12px 12px 0px black;
37 }
```



Gambar: State mahasiswa sukses ditampilkan ke web browser

Siip... aplikasi kita sudah selesai. Pada saat tombol di klik, isi state akan berubah dan menampilkan gambar serta info dari setiap mahasiswa.

Namun masih ada sedikit kekurangan. Prinsip **DRY** (Don't Repeat Yourself) belum kita terapkan di sini. Kode untuk ketiga *event handler* sangat mirip satu sama lain. Ini seharusnya bisa di gabung menjadi satu method yang dipakai bersama. Kode program juga masih sangat statis. Jika ingin menambah 1 mahasiswa baru, harus ikut menambah satu tag `<button>` dan *event handler* secara manual.

Sebagai solusi, tag `<button>` bisa kita generate menggunakan perulangan `mahasiswa.map()` agar jumlah tombol tampil sesuai jumlah array mahasiswa. Event handler juga bisa disatukan dengan memanfaatkan *event object*.

Berikut modifikasi yang diperlukan untuk komponen `MyApp`:

`22.slide_data_mahasiswa_3.html`

```

1  class MyApp extends React.Component {
2      constructor() {
3          super()
4          this.state = {
5              nama: mahasiswa[0].nama,
6              jurusan: mahasiswa[0].jurusan,
7              pasFoto: mahasiswa[0].pasFoto,
8          };
9      }
10
11     handleMahasiswaClick= (event) => {
12         let index = parseInt(event.target.getAttribute('index'));
13         this.setState({

```

```

14     nama: mahasiswa[index].nama,
15     jurusan: mahasiswa[index].jurusan,
16     pasFoto: mahasiswa[index].pasFoto
17   });
18 }
19
20 render() {
21   return (
22     <div>
23     {
24       mahasiswa.map((mahasiswa, i) =>
25         <button
26           key={mahasiswa.id}
27           onClick={this.handleMahasiswaClick}
28           index={i}>
29             {mahasiswa.nama}</button>
30       )
31     }
32
33     <figure>
34       <img src={"img/" + this.state.pasFoto} alt={this.state.nama} />
35       <figcaption>{this.state.nama} ({this.state.jurusan})</figcaption>
36     </figure>
37
38   </div>
39 )
40 }
41 }

```

Kode kali ini memang sedikit lebih kompleks dibanding sebelumnya.

Di dalam method `render()`, terdapat perulangan `mahasiswa.map()` yang akan men-generate tag `<button>`. Karena kita membuat perulangan, atribut `key` menjadi tambahan wajib yang nilainya bisa diambil dari `mahasiswa.id`.

Event `onClick` sekarang cukup mengakses `this.handleMahasiswaClick()` untuk semua button. Selain itu terdapat atribut `index={i}` sebagai atribut bantu yang akan saya pakai di dalam method `this.handleMahasiswaClick()`.

Masuk ke definisi method `handleMahasiswaClick()`, parameter event akan di isi React dengan `event object`. Event object kemudian saya pakai untuk mengambil nilai atribut `index` dari tag `<button>` yang saat ini sedang di klik (baris 12). Fungsi `parseInt()` diperlukan agar nilai hasil `getAttribute('index')` dikonversi dari string menjadi integer. Hasilnya lalu disimpan ke dalam variabel `index`.

Variabel `index` inilah yang menjadi penentu nilai index yang mengisi state di baris 13-17. Sebagai contoh, jika tombol yang di klik memiliki atribut `index="2"`, maka perintah `this.setState()` akan menjadi:

```

this.setState({
  nama: mahasiswa[2].nama,

```

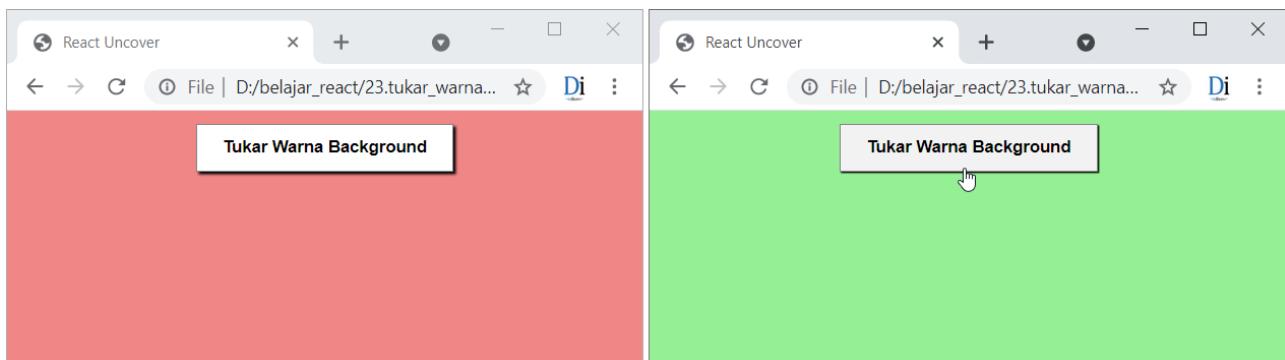
```
jurusan: mahasiswa[2].jurusan,  
pasFoto: mahasiswa[2].pasFoto  
});
```

Dengan semua kode ini, jika ditambah object mahasiswa baru di dalam array mahasiswa, tag <button> otomatis di generate dan langsung bisa berjalan.

Membuat kode dinamis seperti ini memang butuh sedikit "jam terbang". Untuk awal-awal, tidak masalah jika terdapat kode yang berulang (seperti contoh kita sebelumnya). Setelah aplikasinya jadi dan fiturnya berjalan semua, barulah lakukan *refactoring*, yakni cari apakah ada bagian kode program yang bisa dibuat lebih efisien.

8.6. Mini Project: Tukar Warna Background

Lanjut ke mini project kedua, saya ingin membuat tombol yang ketika di klik akan menukar warna background dari merah menjadi hijau dan sebaliknya (*toggle*). Berikut tampilan akhir dari project ini:



Gambar: Tampilan akhir mini project tukar warna background

Konsepnya adalah, ketika tombol di klik, tukar kode CSS milik tag <div> yang sudah di-set untuk memenuhi 100% tinggi dan lebar web browser.

Penukaran kode CSS ini bisa dilakukan dengan berbagai cara. Salah satunya, siapkan 2 buah class CSS di dalam tag <style>, lalu set property `background-color` dengan warna berbeda. Jadikan salah satu nama class ini sebagai nilai awal state (yang disimpan cukup string dari nama class saja). Ketika tombol di klik, tukar nilai state menjadi class lain.

Silahkan anda coba sebentar untuk membuat kode programnya.

Baik, berikut salah satu solusi yang bisa dipakai:

23.tukar_warna_bg_1.html

```
1  <!DOCTYPE html>  
2  <html lang="id">  
3
```

React State

```
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>React Uncover</title>
8    <style>
9      body {
10        margin: 0;
11        text-align: center;
12      }
13
14      button {
15        background-color: white;
16        cursor: pointer;
17        padding: 10px 20px;
18        font-weight: bold;
19        border: 1px solid gray;
20        box-shadow: 2px 2px 2px black;
21        margin: 10px;
22      }
23
24      button:hover {
25        box-shadow: 1px 1px 1px black;
26        background-color: #f1f1f1;
27      }
28
29      div {
30        height: 100vh;
31        width: 100vw;
32      }
33
34      .hijau {
35        background-color: lightgreen;
36      }
37
38      .merah {
39        background-color: lightcoral;
40      }
41    </style>
42  </head>
43
44  <body>
45    <div id="root"></div>
46
47    <script src="js/react.development.js"></script>
48    <script src="js/react-dom.development.js"></script>
49    <script src="js/babel.js"></script>
50    <script type="text/babel">
51
52      class MyApp extends React.Component {
53        constructor() {
54          super()
55          this.state = { divClass: "merah" };
56        }
57
58        handleButtonClick = () => {
```

```

59         if (this.state.divClass === "merah") {
60             this.setState({ divClass: "hijau" })
61         }
62     else {
63         this.setState({ divClass: "merah" })
64     }
65 }
66
67 render() {
68     return (
69         <div className={this.state.divClass}>
70             <button onClick={this.handleButtonClick}>
71                 Tukar Warna Background
72             </button>
73         </div>
74     )
75 }
76 }
77
78 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
79
80 </script>
81 </body>
82
83 </html>

```

Karena kode ini berhubungan langsung dengan CSS, maka saya tampilkan satu file HTML lengkap.

Di dalam tag `<style>` terdapat beberapa selector untuk mengatur tag `body` dan `button` (baris 9-27). Selector `div` di baris 29 berisi kode untuk membuat tag tersebut membesar memenuhi seluruh tinggi dan lebar web browser. Class yang dipakai untuk proses penukaran warna ada di baris 34-40, yakni class `.merah` dan `.hijau`.

Masuk ke kode React, di dalam method `render()` terdapat struktur JSX dengan tag `<div className={this.state.divClass}>` sebagai container luar. Di sini kita memakai state `divClass` sebagai nilai atribut `className`,

Lalu seperti biasa, tag `<button>` memiliki event `click` yang akan mengakses event handler `this.handleButtonClick()`.

Di dalam constructor, state `divClass` diisi dengan nilai awal string "merah". Maka ketika halaman tampil pertama kali, atribut `className` di dalam tag `<div>` akan di proses sebagai `class="merah"`.

Lanjut ke method `handleButtonClick()`, terdapat pemeriksaan kondisi untuk membuat fitur toggle, yakni jika state `divClass` berisi string "merah", ganti nilainya menjadi "hijau". Jika tidak, ganti menjadi "merah". Dengan demikian setiap kali tombol di klik, atribut tag `<div>` akan berganti-ganti dari `class="merah"` menjadi `class="hijau"`, dan sebaliknya.

Menukar isi class CSS menggunakan React cukup sering dilakukan, terutama saat membuat

efek-efek visual yang menarik.

Sebagai alternatif penulisan, pemeriksaan kondisi di dalam `handleButtonClick()` juga bisa dibuat dengan *conditional operator* "`? :`" sebagai berikut:

24.tukar_warna_bg_2.html

```

1 ...
2 handleButtonClick = () => {
3   this.state.divClass === "merah" ?
4     this.setState({ divClass: "hijau" }) : this.setState({ divClass: "merah" });
5 }
6 ...

```

Logika yang dipakai masih sama seperti sebelumnya. Jika kondisi `this.state.divClass === "merah"` menghasilkan nilai `true`, isi state `divClass` dengan `"hijau"`, jika `false`, isi state `divClass` dengan `"merah"`.

Atau juga bisa menggunakan variabel bantu seperti berikut:

25.tukar_warna_bg_3.html

```

1 ...
2 handleButtonClick = () => {
3   let color = (this.state.divClass === "merah") ? "hijau" : "merah";
4   this.setState({ divClass: color });
5 }
6 ...

```

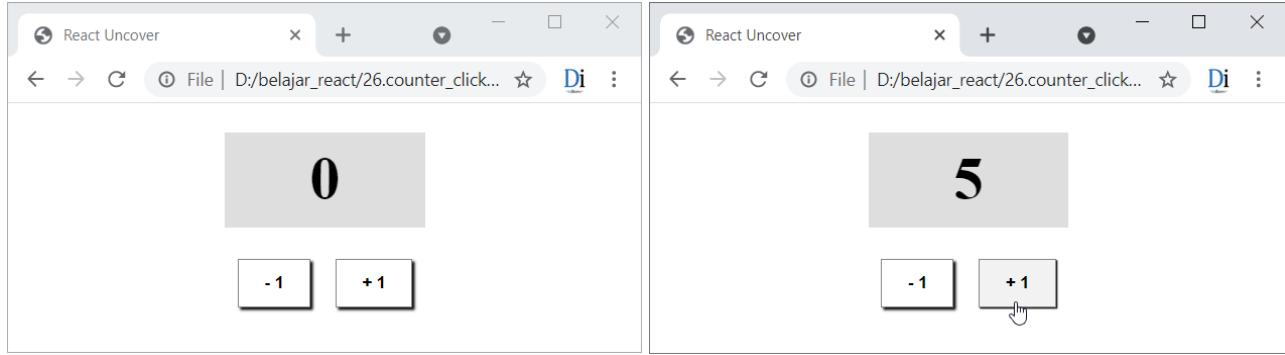
Variabel `color` akan berisi string `"hijau"` atau merah tergantung hasil dari `this.state.divClass === "merah"`, lalu variabel ini diinput ke dalam state `divClass`.

Tidak ada benar dan salah dari ketiga cara ini, lebih ke kesukaan saja. Kodenya memang lebih singkat jika ditulis lewat *conditional operator*, tetapi sedikit sulit dipahami dibandingkan kondisi `if` biasa.

8.7. Mini Project: Counter Click

Belum cukup dengan 2 mini project, kita masuk ke mini project ketiga yang sebenarnya lebih cocok sebagai exercise. Di sini saya ingin membuat angka counter yang akan di update setiap kali tombol di klik. Berikut tampilan akhirnya:

React State



Gambar: Tampilan mini project "counter click"

Jika di klik tombol "-1", angka counter akan berkurang 1. Dan jika di klik tombol "+1", angka counter akan bertambah 1. Silahkan anda coba sebentar rancang kode programnya. Untuk tampilan tidak harus persis, cukup kode Reactnya saja.

Baik, berikut kode yang saya pakai:

26.counter_click.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14      h1 {
15          background-color: gainsboro;
16          font-size: 3em;
17          text-align: center;
18          padding: 10px 0;
19          margin: 15px 0;
20      }
21
22      button {
23          background-color: white;
24          cursor: pointer;
25          padding: 10px 20px;
26          font-weight: bold;
27          border: 1px solid gray;
28          box-shadow: 2px 2px 2px black;
29          margin: 10px;
30      }
31
32      button:hover {
33          box-shadow: 1px 1px 1px black;
```

```

34         background-color: #f1f1f1;
35     }
36   </style>
37 </head>
38
39 <body>
40   <div id="root"></div>
41
42   <script src="js/react.development.js"></script>
43   <script src="js/react-dom.development.js"></script>
44   <script src="js/babel.js"></script>
45   <script type="text/babel">
46
47     class MyApp extends React.Component {
48       constructor() {
49         super()
50         this.state = { counter: 0 };
51       }
52
53       handleDec = () => {
54         this.setState({ counter: this.state.counter - 1 });
55       }
56
57       handleInc = () => {
58         this.setState({ counter: this.state.counter + 1 });
59       }
60
61       render() {
62         return (
63           <div>
64             <h1>{this.state.counter}</h1>
65             <button onClick={this.handleDec}> - 1</button>
66             <button onClick={this.handleInc}> + 1</button>
67           </div>
68         )
69       }
70     }
71
72     ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
73
74   </script>
75 </body>
76
77 </html>

```

Saya yakin anda sudah bisa memahami cara kerja kode ini. Sebagai penampung angka counter, buat state counter seperti di baris 50. Pada saat tombol di klik, update nilai counter apakah dikurangi 1 atau ditambah 1 dari nilai `this.state.counter` yang sudah ada.

8.8. Function Sebagai Nilai State

Agar lebih efisien, proses update state dijalankan React secara *asynchronous*. Atau dengan kata

lain, update nilai state "diantrikan" terlebih dahulu. Efeknya, ada kemungkinan kita mengakses nilai state yang belum update.

Ini bisa menjadi masalah saat mengupdate state yang bergantung kepada nilai state sebelumnya. Sebagai contoh, dalam mini project "counter click", proses update state dilakukan dengan perintah berikut:

```
this.setState({ counter: this.state.counter - 1 });
```

Di sini, nilai state counter diambil dari **nilai counter sebelumnya** dan dikurangi 1. Inilah yang dimaksud dengan "nilai state saat ini bergantung pada nilai state sebelumnya".

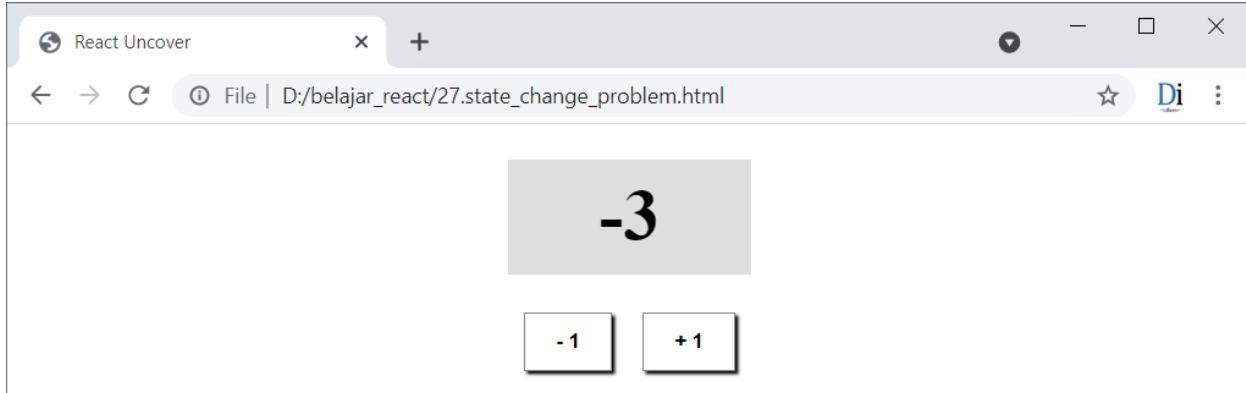
Berikut contoh kasus yang lebih jelas:

27.state_change_problem.html

```
1  class MyApp extends React.Component {
2      constructor() {
3          super()
4          this.state = { counter: 0 };
5      }
6
7      handleDec = () => {
8          this.setState({ counter: this.state.counter - 1 });
9          this.setState({ counter: this.state.counter - 1 });
10     }
11
12     handleInc = () => {
13         this.setState({ counter: this.state.counter + 1 });
14         this.setState({ counter: this.state.counter + 1 });
15     }
16
17     render() {
18         return (
19             <div>
20                 <h1>{this.state.counter}</h1>
21                 <button onClick={this.handleDec}> - 1</button>
22                 <button onClick={this.handleInc}> + 1</button>
23             </div>
24         )
25     }
26 }
27
28 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Kode ini hanya memodifikasi sedikit mini project "counter click", tambahan ada di dalam method `handleDec()` dan `handleInc()`. Sekarang setiap kali tombol di klik, ada 2 perintah `this.setState()` yang berjalan (baris 8-9 dan 13-14).

Dengan perubahan ini, nilai state counter harusnya akan berkurang 2 angka atau naik 2 angka dalam setiap klik. Akan tetapi hasilnya tidak demikian:



Gambar: Nilai variabel counter tetap naik/turun 1 angka

Ternyata setiap kali tombol di klik, state counter naik/turun 1 angka saja, bukan 2 angka. Jika di tambah 1 atau 10 lagi perintah `this.setState()`, hasilnya masih tetap sama. Ini terjadi karena kita mendapatkan nilai state counter yang belum update.

Untuk mengatasi masalah ini, React menghadirkan cara kedua penulisan method `this.setState()`, yakni dengan mengisinya dengan *function* atau *callback*.

Function tersebut otomatis menerima 2 argument. Argument pertama diisi React dengan nilai state sebelumnya, dan argument kedua diisi nilai *props* saat proses update berjalan. Kedua argument tidak harus ditulis, biasanya kita hanya butuh nilai state saja. Nilai state ini dijamin React sebagai hasil final update, tidak terpengaruh proses *asynchronous*.

Berikut modifikasi dari method event handler sebelumnya:

`28.state_change_solution.html`

```

1 ...
2   handleDec = () => {
3     this.setState((prevState) => ({ counter: prevState.counter - 1 }));
4     this.setState((prevState) => ({ counter: prevState.counter - 1 }));
5   }
6
7   handleInc = () => {
8     this.setState((prevState) => ({ counter: prevState.counter + 1 }));
9     this.setState((prevState) => ({ counter: prevState.counter + 1 }));
10  }
11 ...

```

Perhatikan isi method `this.setState()`, itu adalah function yang saya tulis dalam format *arrow notation*. Function ini dibuat dengan 1 parameter `prevState`. Nama parameter boleh bebas, dan karena ini parameter pertama maka akan diisi React dengan nilai state sebelumnya.

Function ini harus mengembalikan nilai dalam bentuk pasangan object yang sama seperti cara update state biasa. Dalam contoh ini, nilai state counter sebelumnya bisa diakses dengan perintah `prevState.counter`.

Sekarang, nilai counter langsung lompat 2 angka ketika tombol di klik.

Pertanyaan mendasar, kapan harus mengisi `this.setState()` dengan nilai langsung dan kapan menggunakan *function callback* seperti ini?

Jika nilai yang akan diinput berbentuk nilai langsung, maka tidak perlu pakai function.

Penulisan function hanya perlu jika nilai state itu bergantung pada nilai state sebelumnya.

Dengan alasan ini, method *event handler* di mini project `26.counter_click.html` seharusnya juga ditulis dalam format function. Silahkan jika ingin diubah.

Untuk mayoritas kode program, meskipun kita mengupdate nilai state berdasarkan state sebelumnya, kadang tidak masalah jika tetap ditulis dengan cara biasa (bukan dalam bentuk function). Mini project di file `26.counter_click.html` membuktikan hal ini.

Namun untuk menghindari hasil yang tidak terduga, tetap lebih disarankan ditulis dalam bentuk function.

8.9. Mengirim Nilai State ke Child Component

Untuk aplikasi yang melibatkan banyak komponen, state milik *parent component* bisa dikirim ke *child component* untuk pemrosesan lebih lanjut. Caranya, kirim state sebagai *props* seperti contoh berikut:

`29.state_down.html`

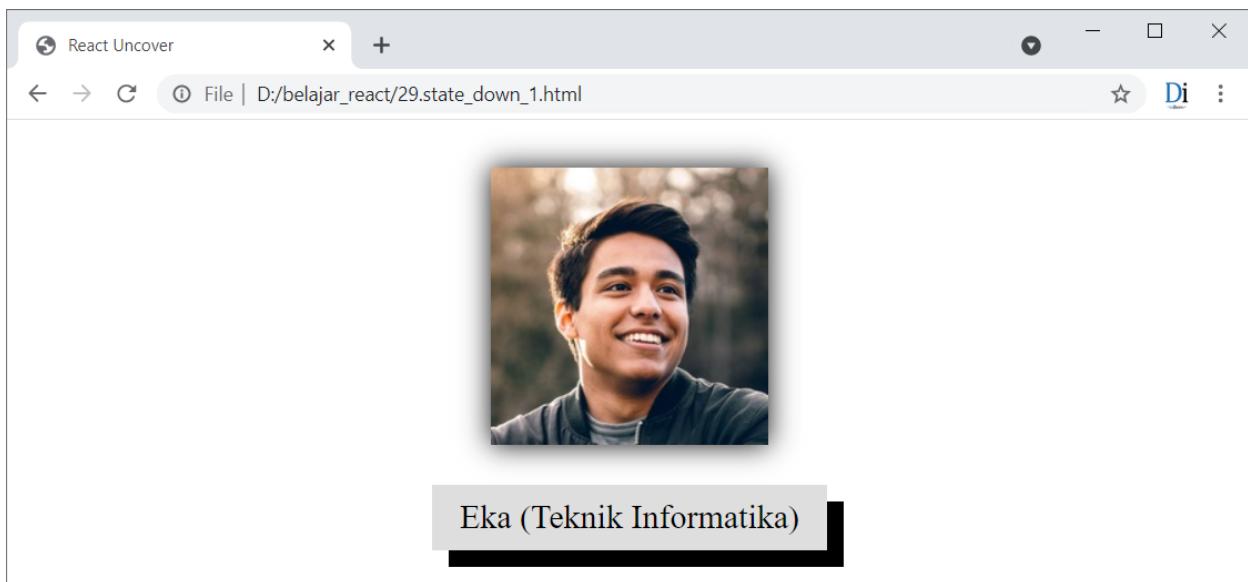
```

1  class Mahasiswa extends React.Component {
2      render() {
3          return (
4              <figure>
5                  <img
6                      src={"img/" + this.props.mahasiswa.pasFoto}
7                      alt={this.props.mahasiswa.nama}
8                  />
9                  <figcaption>
10                     {this.props.mahasiswa.nama} ({this.props.mahasiswa.jurusan})
11                 </figcaption>
12             </figure>
13         )
14     }
15 }
16
17 class MyApp extends React.Component {
18     constructor() {
19         super()
20         this.state = {
21             nama: "Eka",
22             jurusan: "Teknik Informatika",
23             pasFoto: "people1.jpg"

```

React State

```
24     );
25   }
26
27   render() {
28     return (
29       <div>
30         <Mahasiswa mahasiswa={this.state} />
31       </div>
32     )
33   }
34 }
35
36 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Menampilkan nilai state di child component

Berangkat dari constructor di komponen `MyApp`, saya membuat 3 state: `nama`, `jurusan` dan `pasFoto` (baris 20-24). State ini kemudian diinput sebagai atribut `mahasiswa` ke dalam tag `<Mahasiswa/>` di baris 30.

Sesampainya di dalam komponen `Mahasiswa`, ketiga atribut akan menjadi `props` yang bisa diakses dari perintah `this.props.mahasiswa.nama`, `this.props.mahasiswa.jurusan` dan `this.props.mahasiswa.pasFoto`. Inilah cara mengirim nilai state dari parent component ke child component.

Terdapat beberapa variasi tambahan. Misal, jika isi variabel `this.state` cukup banyak, kita bisa kirim state yang diperlukan saja secara terpisah:

30.state_down_2.html

```
1  class MyApp extends React.Component {
2    ...
3    render() {
4      return (
5        <div>
```

React State

```
6      <Mahasiswa nama={this.state.nama} jurusan={this.state.jurusan}
7          pasFoto={this.state.pasFoto} />
8    </div>
9  )
10 }
11 }
```

Di baris 6-7 saya mengirim state `nama`, `jurusan` dan `pasFoto` secara terpisah. Sesampainya di komponen `Mahasiswa`, ini bisa diakses dari `this.props.nama`, `this.props.jurusan` dan `this.props.pasFoto`.

Teknik lain yang juga sering dipakai adalah menyimpan nilai `props` ke dalam property di *child component* agar tidak terlalu panjang. Ini biasa dilakukan dari dalam constructor:

31.state_down_3.html

```
1 class Mahasiswa extends React.Component {
2   constructor(props) {
3     super(props);
4     this.nama = this.props.mahasiswa.nama;
5     this.jurusan = this.props.mahasiswa.jurusan;
6     this.pasFoto = this.props.mahasiswa.pasFoto;
7   }
8   render() {
9     return (
10       <figure>
11         <img
12           src={"img/" + this.pasFoto}
13           alt={this.nama}
14         />
15         <figcaption>
16           {this.nama} ({this.jurusan})
17         </figcaption>
18       </figure>
19     )
20   }
21 }
22
23 class MyApp extends React.Component {
24   constructor() {
25     super()
26     this.state = {
27       nama: "Eka",
28       jurusan: "Teknik Informatika",
29       pasFoto: "people1.jpg"
30     };
31   }
32
33   render() {
34     return (
35       <div>
36         <Mahasiswa mahasiswa={this.state} />
37       </div>
```

```

38      )
39    }
40  }
41
42 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Proses pengisian property ada di baris 4-6. Dengan cara ini, penulisan *props* di dalam method *render()* milik komponen Mahasiswa bisa lebih singkat.

8.10. Mengubah Nilai State dari dalam Child Component

Materi sebelumnya membahas cara mengirim nilai state dari *parent component* ke *child component*. Dalam beberapa kasus, kita juga ingin mengubah nilai state tersebut dari dalam *child component*.

Misalnya saya ingin menambah tombol "Tukar Mahasiswa" di komponen Mahasiswa yang ketika di klik, akan mengganti data mahasiswa dengan data lain. Masalahnya, data mahasiswa asli tersimpan dalam bentuk state di komponen *MyApp*. Data yang diterima oleh komponen Mahasiswa hanya dalam bentuk *props* yang bersifat "*read only*" (tidak bisa diubah).

Salah satu solusi adalah dengan membuat state mahasiswa baru di dalam komponen *Mahasiswa*. Namun ini melanggar prinsip dasar yang disebut React sebagai "*single source of truth*". Maksudnya, dalam satu aplikasi disarankan hanya ada satu tempat untuk menyimpan suatu nilai. Tidak masalah jika komponen *Mahasiswa* ingin memiliki state internal sendiri, tapi jika ada 2 state yang sama di dua komponen berbeda, pengelolaan data menjadi tidak sinkron.

Solusi lain adalah memindahkan state mahasiswa dari *MyApp* ke dalam komponen *Mahasiswa*. Ini bisa saja dilakukan, tapi kurang praktis untuk aplikasi yang kompleks.

Pengelolaan state di aplikasi React bisa diibaratkan sebuah struktur pohon. Dengan menempatkan state mahasiswa di *MyApp*, data itu bisa diakses oleh berbagai *child component* lain dari *MyApp*. Jika state mahasiswa di pindah ke komponen *Mahasiswa*, maka *child component* *MyApp* lain tidak bisa lagi mengaksesnya (sebenarnya tetap bisa, tapi jadi lebih panjang).

Solusi yang sering dipakai adalah membuat satu method di *parent element* yang berisi perintah untuk mengupdate state, lalu kirim method ini sebagai *event handler* melalui *props* ke *child element*.

Pola penulisan seperti ini memang sedikit rumit, namun harus bisa dipahami karena sangat sering digunakan dalam React.

Di Bab **React Event** terdapat bahasan tentang "[Menurunkan \(passing down\) event handler](#)". Kita akan pakai pola yang sama di sini, oleh karena itu silahkan baca kembali jika dirasa perlu.

Sebagai contoh kasus, saya akan tambah tombol "Tukar Mahasiswa" di komponen MyApp terlebih dahulu:

32.state_prepared_up.html

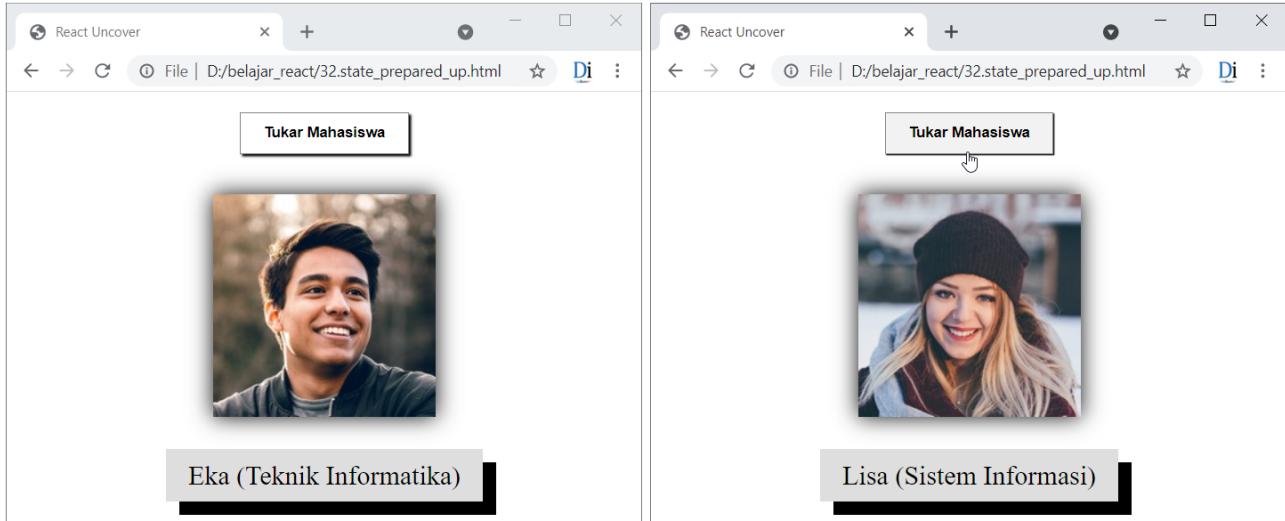
```

1  class Mahasiswa extends React.Component {
2      render() {
3          return (
4              <React.Fragment>
5                  <figure>
6                      <img
7                          src={"img/" + this.props.mahasiswa.pasFoto}
8                          alt={this.props.mahasiswa.nama}
9                      />
10                     <figcaption>
11                         {this.props.mahasiswa.nama} ({this.props.mahasiswa.jurusan})
12                     </figcaption>
13                 </figure>
14             </React.Fragment>
15         )
16     }
17 }
18
19 class MyApp extends React.Component {
20     constructor() {
21         super()
22         this.state = {
23             nama: "Eka",
24             jurusan: "Teknik Informatika",
25             pasFoto: "people1.jpg"
26         };
27     }
28
29     handleTukarMahasiswa = () => {
30         this.setState({
31             nama: "Lisa",
32             jurusan: "Sistem Informasi",
33             pasFoto: "people2.jpg"
34         })
35     }
36
37     render() {
38         return (
39             <div>
40                 <button onClick={this.handleTukarMahasiswa}>Tukar Mahasiswa</button>
41                 <Mahasiswa mahasiswa={this.state} />
42             </div>
43         )
44     }
45 }
46
47 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Kode ini sangat mirip seperti contoh sebelumnya. Terdapat tambahan method handleTukar

Mahasiswa() di baris 29-35 yang akan diakses oleh tag <button> di baris 40. Pada saat tombol di klik, nilai state akan berubah:



Gambar: Tombol "Tukar Mahasiswa" untuk mengganti state

Tidak ada masalah di sini.

Sekarang, jika tombol "Tukar Mahasiswa" di pindah dari komponen `MyApp` ke komponen `Mahasiswa`, bagaimana agar event `click` tetap berjalan? Method `handleTukarMahasiswa()` tidak bisa ikut dibawa pindah karena perintah `this.setState()` tidak bisa dipakai untuk mengakses state milik `MyApp`.

Solusinya, kirim `handleTukarMahasiswa()` sebagai `props`:

33.state_up.html

```

1  class Mahasiswa extends React.Component {
2
3      render() {
4          return (
5              <React.Fragment>
6
7                  <button onClick={this.props.onTukarMahasiswa}>
8                      Tukar Mahasiswa
9                  </button>
10
11                  <figure>
12                      <img
13                          src={"img/" + this.props.mahasiswa.pasFoto}
14                          alt={this.props.mahasiswa.nama}
15                      />
16                      <figcaption>
17                          {this.props.mahasiswa.nama} ({this.props.mahasiswa.jurusan})
18                      </figcaption>
19                  </figure>
20
21              </React.Fragment>

```

React State

```
22     )
23   }
24 }
25
26 class MyApp extends React.Component {
27   constructor() {
28     super()
29     this.state = {
30       nama: "Eka",
31       jurusan: "Teknik Informatika",
32       pasFoto: "people1.jpg"
33     };
34   }
35
36   handleTukarMahasiswa = () => {
37     this.setState({
38       nama: "Lisa",
39       jurusan: "Sistem Informasi",
40       pasFoto: "people2.jpg"
41     })
42   }
43
44   render() {
45     return (
46       <div>
47         <Mahasiswa
48           mahasiswa={this.state}
49           onTukarMahasiswa={this.handleTukarMahasiswa}>
50           />
51         </div>
52       )
53     }
54   }
55
56 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Di sini kita memanfaatkan teknik *passing down event handler*. Method `handleTukarMahasiswa()` tetap ditulis di komponen `MyApp`, lalu dikirim sebagai `props` di baris 49.

Sesampainya di komponen `Mahasiswa`, atribut `onClick={this.props.onTukarMahasiswa}` milik tag `<button>` di baris 7 secara tidak langsung akan menjalankan method `handleTukarMahasiswa()` yang ada di komponen `MyApp`. Alur proses seperti ini memang sedikit rumit, pemahaman seputar konsep *event handler* sangat diperlukan.

Lebih jauh lagi, kita bisa mengirim data baru dari komponen `Mahasiswa` untuk diisi sebagai nilai state ke komponen `MyApp`. Berikut modifikasinya:

```
34.state_up_argument_1.html
1  class Mahasiswa extends React.Component {
2
3   handleTukarMahasiswaBaru = () => {
4     let mahasiswaBaru = {
```

React State

```
5      nama: "Lisa",
6      jurusan: "Sistem Informasi",
7      pasFoto: "people2.jpg"
8  }
9  this.props.onTukarMahasiswa(mahasiswaBaru);
10 }
11
12 render() {
13   return (
14     <React.Fragment>
15
16       <button onClick={this.handleTukarMahasiswaBaru}>Tukar Mahasiswa</button>
17
18       <figure>
19         <img
20           src={"img/" + this.props.mahasiswa.pasFoto}
21           alt={this.props.mahasiswa.nama}
22         />
23         <figcaption>
24           {this.props.mahasiswa.nama} ({this.props.mahasiswa.jurusan})
25         </figcaption>
26       </figure>
27
28     </React.Fragment>
29   )
30 }
31 }
32
33 class MyApp extends React.Component {
34   constructor() {
35     super()
36     this.state = {
37       nama: "Eka",
38       jurusan: "Teknik Informatika",
39       pasFoto: "people1.jpg"
40     };
41   }
42
43   handleTukarMahasiswa = (mahasiswaBaru) => {
44     this.setState({
45       nama: mahasiswaBaru.nama,
46       jurusan: mahasiswaBaru.jurusan,
47       pasFoto: mahasiswaBaru.pasFoto
48     })
49   }
50
51   render() {
52     return (
53       <div>
54         <Mahasiswa
55           mahasiswa={this.state}
56           onTukarMahasiswa={this.handleTukarMahasiswa}
57         />
58       </div>
59     )
60   }
61 }
```

```

60     }
61   }
62
63 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Perhatikan isi method `handleTukarMahasiswa()` di baris 43-49. Method ini menerima 1 parameter: `mahasiswaBaru`.

Parameter `mahasiswaBaru` harus berbentuk object karena property `mahasiswaBaru.nama`, `mahasiswaBaru.jurusan` dan `mahasiswaBaru.pasFoto` akan dipakai untuk mengisi nilai state. Maka kode apapun yang ingin mengakses `handleTukarMahasiswa()`, harus mengirim 1 argument untuk mengisi parameter `mahasiswaBaru` ini.

Kemudian sama seperti sebelumnya, method `handleTukarMahasiswa()` di turunkan ke komponen `Mahasiswa` sebagai `props` di baris 56.

Di baris 16 dalam komponen `Mahasiswa`, atribut `onClick` milik tag `<button>` sekarang mengakses method internal sendiri yakni `this.handleTukarMahasiswaBaru`.

Pendefinisian method `handleTukarMahasiswaBaru()` ada di baris 3-10. Pada bagian awal, variabel `mahasiswaBaru` diisi object dengan 3 property: `nama: "Lisa"`, `jurusan: "Sistem Informasi"`, dan `pasFoto: "people2.jpg"`. Variabel ini kemudian menjadi argument dari method `this.props.onTukarMahasiswa()`.

Dengan kode ini, saat tombol "Tukar Mahasiswa" di klik, nilai mahasiswa yang ada di baris 5-7 akan menjadi nilai state baru di dalam komponen `MyApp`. Inilah cara mengubah nilai state parent element dari dalam child component.

Sedikit tambahan, pengiriman nilai argumen di dalam method `handleTukarMahasiswaBaru()` juga bisa langsung ditulis sebagai berikut:

35.state_up_argument_2.html

```

1 ...
2   handleTukarMahasiswaBaru = () => {
3     this.props.onTukarMahasiswa({
4       nama: "Lisa",
5       jurusan: "Sistem Informasi",
6       pasFoto: "people2.jpg"
7     })
8   }
9 ...

```

Dengan cara ini, kita tidak butuh bantuan variabel `mahasiswaBaru`, tapi langsung menulis object sebagai argument saat menjalankan method `this.props.onTukarMahasiswa()`.

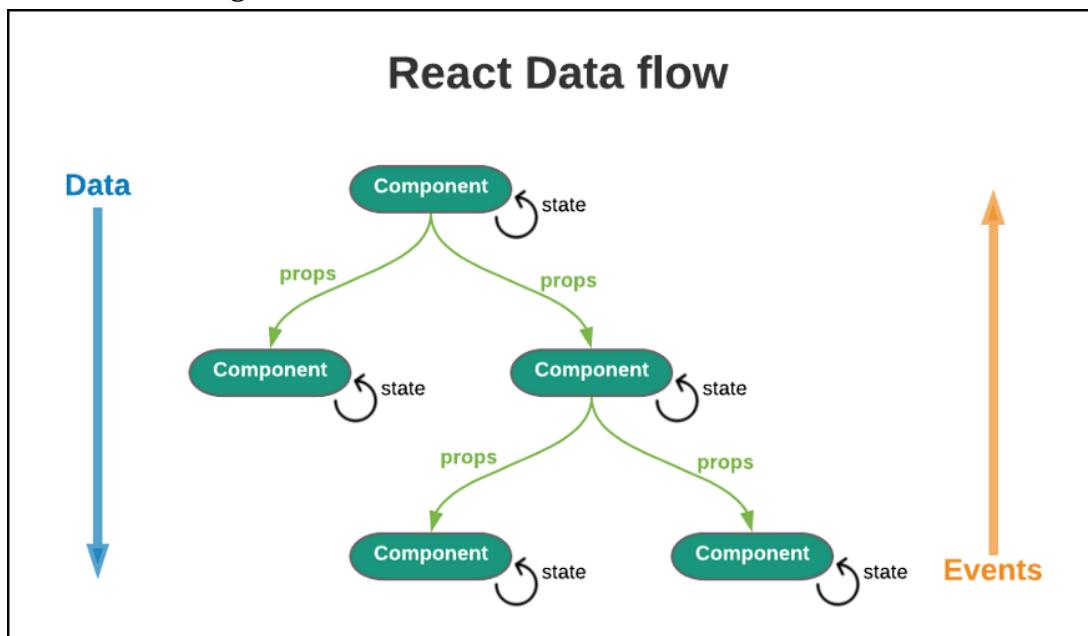
Konsep ini saya akui memang cukup rumit, kita harus paham seperti apa perputaran data dari komponen `MyApp` dengan komponen `Mahasiswa`. Jika masih agak bingung, saya sangat anjurkan untuk dipelajari ulang karena konsep ini sangat penting.

Dan.... itu baru melibatkan 2 buah komponen. Bisa saja nanti kita memiliki 3, 4 bahkan 10 komponen yang saling mengirim data satu sama lain (via *props*). Data tersebut bisa berbentuk nilai biasa, atau mengirim *event handler*.

8.11. Alur Pengiriman Data di React

Materi state yang kita bahas dalam bab ini melengkapi konsep utama yang ada di React, yakni: **Component**, **Props**, dan **State**. Tiga konsep inilah yang menjadi inti dari sebuah aplikasi React.

Memahami alur pengiriman data antar komponen juga tidak kalah penting, yang bisa diilustrasikan dalam diagram berikut:



Gambar: Aliran data di React (sumber gambar: javascript.plainenglish.io)

Di React, perpindahan data antar **component** dilakukan melalui **props**. Sedangkan data yang diperlukan untuk proses internal, disimpan sebagai **state**. Jika state ingin dikirim ke komponen lain, itu juga harus lewat *props*.

Props hanya bisa dikirim bertingkat dari *parent component* ke *child component*, lalu baru ke *grand child component*, tidak bisa langsung lompat.

Jika kita perlu mengupdate nilai state dari *child component*, bisa memakai teknik *passing down event handler* seperti bahasan sebelumnya.

Namun tidak masalah jika alur data ini masih agak samar. Kita akan bahas dengan berbagai studi kasus dan latihan hingga akhir buku nanti.

Itulah bahasan detail tentang state. Meskipun sudah cukup panjang, tapi sebenarnya masih belum selesai. Yup, kita baru membahas cara penulisan state di *class component* saja, bagaimana dengan di *functional component*? Materi ini akan kita bahas dalam bab berikutnya.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

9. React useState Hook

Hingga beberapa waktu lalu, **state** hanya bisa dipakai pada *class component* saja. Sedangkan *functional component* tidak bisa memiliki state sehingga disebut sebagai *stateless component*.

Namun di awal 2019, tepatnya pada React versi 16.8, diperkenalkanlah **useState hook** sebagai cara untuk membuat state di functional component.

9.1. Pengertian React Hook

Di React, **hook** adalah function/method yang memungkinkan kita mengakses berbagai fitur di *functional component* yang sebelumnya hanya tersedia di *class component*. Pengertian ini memang agak "mengambang", tapi itulah kesimpulan yang saya dapat setelah mempelajari hook.

Hook hadir dalam berbagai nama tergantung fitur apa yang ingin dipakai. Sebagai contoh, untuk membuat state, nama hooknya adalah **useState()**. Nantinya kita akan pelajari berbagai hook lain seperti **useRef()** dan **useEffect()**. Untuk materi lanjutan juga tersedia **useContext()**, **useReducer()**, **useMemo()**, **useCallback()**, dll. Bahkan, kita juga bisa membuat custom hook sendiri.

Salah satu ciri khas hook adalah, semua perintah diawali dengan kata "use".

9.2. Cara Penggunaan useState Hook

Di atas sudah kita singgung kalau **useState()** adalah hook yang dipakai untuk membuat **state** di *functional component*. State ini fungsinya tetap sama seperti di *class component*, yakni sebagai tempat menyimpan data yang jika nilainya berubah, komponen akan di render ulang.

Cara penggunaan **useState()** akan lebih mudah dijelaskan dengan contoh langsung:

01.usestate_1.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

React useState Hook

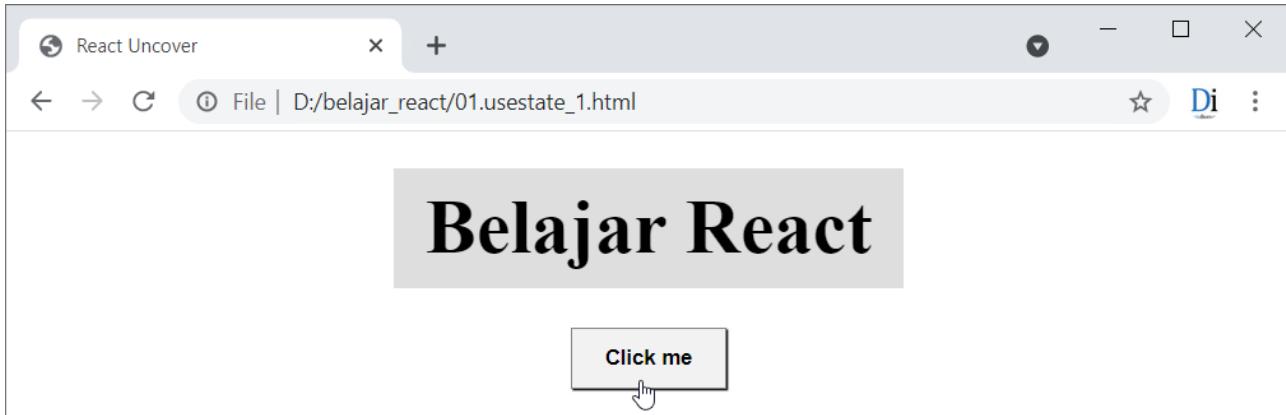
```
7   <title>React Uncover</title>
8   <style>
9     #root {
10       display: flex;
11       justify-content: center;
12     }
13
14   div {
15     display: flex;
16     justify-content: center;
17     flex-direction: column;
18     align-items: center;
19   }
20
21   h1 {
22     background-color: gainsboro;
23     font-size: 3em;
24     text-align: center;
25     padding: 10px 20px;
26     margin: 15px 0;
27   }
28
29   button {
30     background-color: white;
31     cursor: pointer;
32     padding: 10px 20px;
33     font-weight: bold;
34     border: 1px solid gray;
35     box-shadow: 2px 2px 2px black;
36     margin: 10px;
37   }
38
39   button:hover {
40     box-shadow: 1px 1px 1px black;
41     background-color: #f1f1f1;
42   }
43   </style>
44 </head>
45
46 <body>
47   <div id="root"></div>
48
49   <script src="js/react.development.js"></script>
50   <script src="js/react-dom.development.js"></script>
51   <script src="js/babel.js"></script>
52   <script type="text/babel">
53
54     const MyApp = () => {
55       const [judul, setJudul] = React.useState("Belajar React");
56
57       const handleButtonClick = () => {
58         setJudul("Belajar JavaScript");
59       }
60
61       return (

```

```

62      <div>
63          <h1>{judul}</h1>
64          <button onClick={handleButtonClick}>Click me</button>
65      </div>
66  )
67 }
68
69 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
70
71 </script>
72 </body>
73
74 </html>

```



Gambar: Menukar judul menggunakan useState hook

Tampilan kode program ini sama seperti contoh yang kita pakai pada awal bab sebelumnya. Di tengah halaman terdapat judul "Belajar React" yang akan berganti menjadi "Belajar JavaScript" saat tombol "Click me" di klik.

Kodenya cukup panjang karena saya mencantumkan lengkap kode HTML, CSS dan JavaScript. Ini semata-mata agar mudah di copy-paste bagi yang ingin mencoba langsung. Beberapa contoh setelah ini juga akan memakai kode CSS yang sama.

Fokus utama kita ada di baris 54-69. Seperti biasa, pembacaan kode program akan lebih mudah dengan melihat apa komponen yang diinput ke dalam `ReactDOM.createRoot().render()` di baris 69.

Komponen yang dimaksud adalah `MyApp`, maka lanjut ke bagian `return` dari komponen ini. Di situ terlihat kalau tag `<h1>` akan menampilkan isi dari variabel `judul` (baris 63), serta atribut `onClick` milik tag `<button>` akan mengakses fungsi bernama `handleButtonClick` (baris 64).

Namun dibagian atas komponen `MyApp` tidak terdapat pendefinisian variabel `judul`. Di situ `judul` bukanlah sebuah variabel biasa, tapi nama sebuah state yang dibuat menggunakan `useState()` hook di baris 55.

Di baris tersebut saya mendefinisikan 2 konstanta: `judul`, dan `setJudul`. Keduanya ditulis dalam tanda kurung kotak "[]" karena dipakai untuk "membuka" array yang didapat sebagai

hasil dari method `React.useState("Belajar React")`. Ini adalah penerapan dari fitur [array destructuring](#) yang pernah kita bahas di Bab 2.

Method `React.useState()` mengembalikan array dengan dua element. Element pertama menjadi nama state, dan element kedua menjadi function yang dipakai untuk mengupdate state tersebut.

Dalam contoh di atas, state judul langsung berisi string "Belajar React" karena argument yang diinput ke dalam `React.useState()` akan menjadi nilai awal.

Berdasarkan penjelasan ini, format perintah `useState()` adalah sebagai berikut:

```
const [<nama_state>, <function_untuk_update>] = React.useState(<nilai_awal>);
```

Ketika tombol "Click me" di klik, itu akan menjalankan method `handleButtonClick()`. Method ini berisi perintah `setJudul("Belajar JavaScript")` yang berfungsi untuk mengupdate nilai state `judul`. Nilai yang diinput sebagai argument akan menjadi nilai baru, sehingga judul `<h1>` akan berubah menjadi "Belajar JavaScript".

Nama yang dipakai sebagai fungsi untuk proses update state boleh bebas. Namun sebagian besar programmer React memberi nama dengan awalan kata "set", lalu diikuti dengan nama state yang akan di update seperti `setCounter`, `setMahasiswa` atau `setJudul`.

Contoh lain, jika kita ingin membuat state `counter` dan mengisinya dengan nilai awal 8, bisa menggunakan perintah berikut:

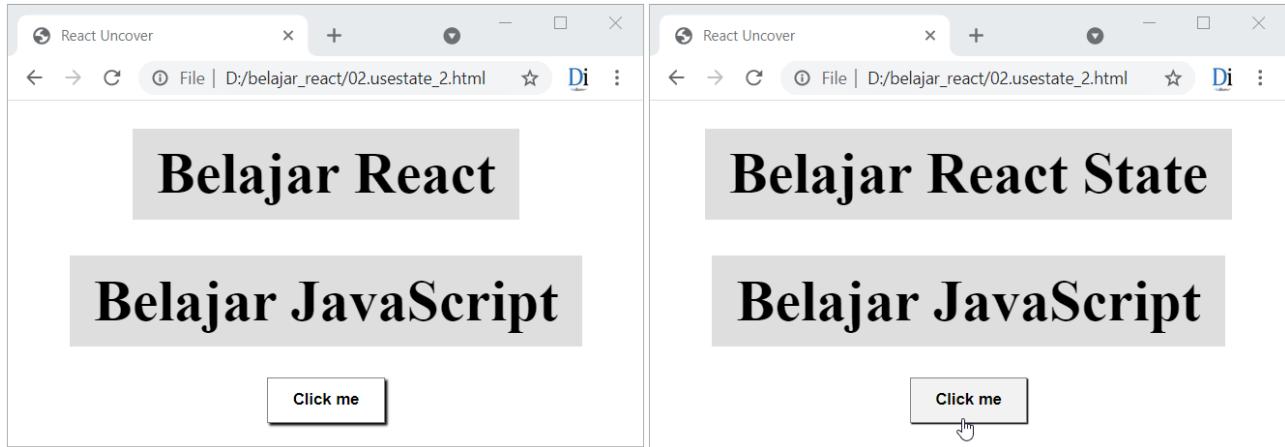
```
const [counter, setCounter] = React.useState(8);
```

Berbeda dengan di class component, kita bisa memanggil beberapa kali perintah `React.useState()` untuk membuat 2 state atau lebih. Berikut contoh prakteknya:

02.usestate_2.html

```
1 const MyApp = () => {
2   const [judul1, setJudul1] = React.useState("Belajar React");
3   const [judul2, setJudul2] = React.useState("Belajar JavaScript");
4
5   const handleButtonClick = () => {
6     setJudul1("Belajar React State");
7   }
8
9   return (
10    <div>
11      <h1>{judul1}</h1>
12      <h1>{judul2}</h1>
13      <button onClick={handleButtonClick}>Click me</button>
14    </div>
15  )
16}
17
```

```
18 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengupdate salah satu state

Di baris 2 dan 3 terdapat perintah `useState()` hook untuk membuat state judul1 dan judul2. Kedua state kemudian ditampilkan ke dalam tag `<h1>` di baris 11-13.

Pada saat tombol "Click me" di klik, isi method `handleButtonClick()` akan menjalankan perintah `setJudul1("Belajar React State")`, hasilnya judul tag `<h1>` pertama akan berganti menjadi "Belajar React State".

Syarat lain dalam penggunaan `useState` hook adalah, perintah ini harus ada di urutan paling awal *functional component*, serta tidak bisa diletakkan dalam kondisi if maupun di dalam function terpisah. Kode berikut akan error saat tombol di klik:

03.usestate_problem.html

```
1 const MyApp = () => {
2   const [judul1, setJudul1] = React.useState("Belajar React");
3   const [judul2, setJudul2] = React.useState("Belajar JavaScript");
4
5   const handleButtonClick = () => {
6     setJudul1("Belajar React State");
7     const [judul3, setJudul3] = React.useState("Belajar Web Programming");
8   }
9 }
```

Perintah di baris 7 akan menampilkan pesan "Uncaught Error: Invalid hook call. Hooks can only be called inside of the body of a function component".

Mengisi Object Sebagai Nilai State (Nested Object)

Sama seperti di *class component*, `useState` hook juga bisa diisi dengan nilai berbentuk object. Namun terdapat beberapa kendala yang bisa terjadi.

Pertama, fungsi yang dipakai untuk mengupdate nilai state akan menimpa seluruh isi state. Maka untuk nilai state yang berbentuk object, kita harus tambah dengan *spread operator*.

Berikut contoh kasusnya:

```
04.usestate_object_value.html

1 const MyApp = () => {
2   const [judul, setJudul] = React.useState(
3     {
4       satu: "Belajar React",
5       dua: "Belajar JavaScript"
6     });
7
8   const handleButtonClick = () => {
9     setJudul({ ...judul, satu: "Belajar React State" });
10  }
11
12  return (
13    <div>
14      <h1>{judul.satu}</h1>
15      <h1>{judul.dua}</h1>
16      <button onClick={handleButtonClick}>Click me</button>
17    </div>
18  )
19}
20
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Di baris 2 saya membuat state `judul` yang nilainya berbentuk object, yakni `satu` dan `dua`. Kedua nilai bisa diakses dengan perintah `judul.satu` dan `judul.dua` seperti di baris 14-15.

Jika kita ingin mengupdate salah satu nilai object, tidak bisa langsung dengan perintah `setJudul({satu: "Belajar React State"})` karena perintah ini akan menghapus isi state `judul.dua`.

Agar nilai state yang sudah ada tidak ikut terhapus, gunakan *spread operator* untuk "menyebarkan" isi state sebelumnya, lalu baru tulis nama state yang ingin ditukar:

```
setJudul({ ...judul, satu: "Belajar React State" });
```

Perintah ini akan diproses sebagai:

```
setJudul({
  satu: "Belajar React",
  dua: "Belajar JavaScript"
  satu: "Belajar React State"
});
```

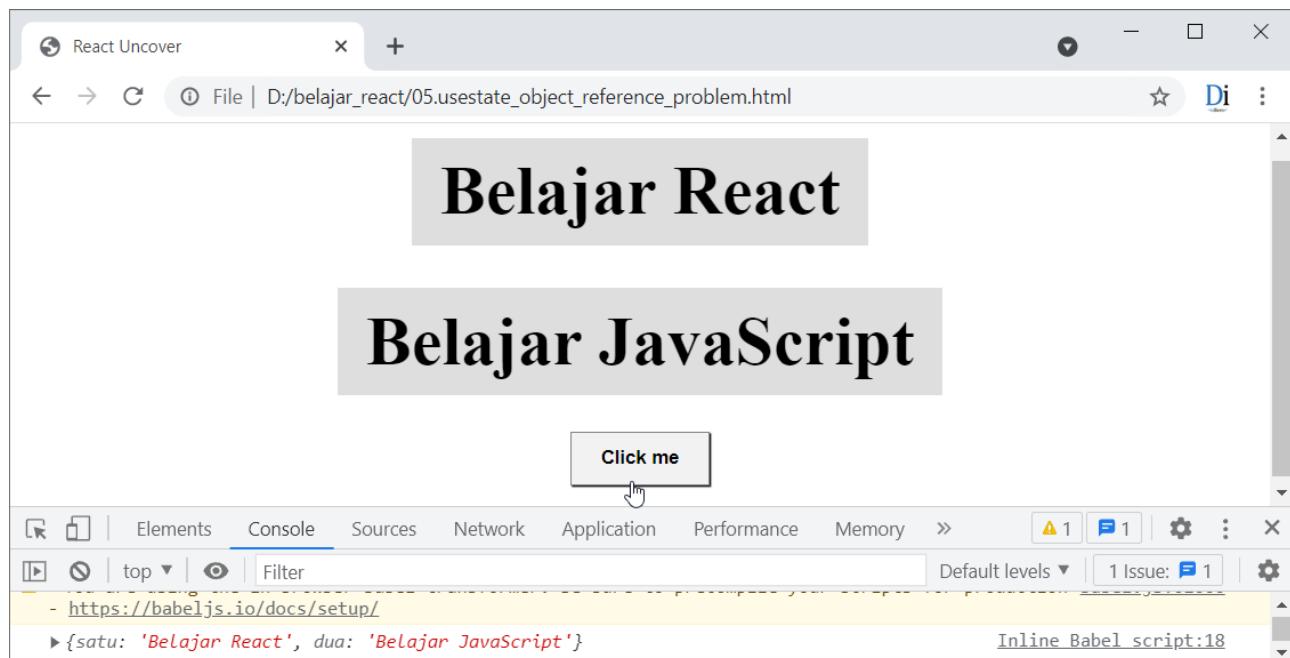
Di JavaScript, tidak masalah jika terdapat 2 nama property di sebuah object. Property yang ditulis paling akhir akan menimpa property sebelumnya.

Masalah kedua yang bisa terjadi saat mengupdate nilai state dalam berbentuk object adalah efek "copy by reference". Silahkan pelajari sejenak kode berikut, lalu coba jalankan:

React useState Hook

05.usestate_object_reference_problem.html

```
1 const MyApp = () => {
2   const [judul, setJudul] = React.useState(
3     {
4       satu: "Belajar React",
5       dua: "Belajar JavaScript"
6     });
7
8   const handleButtonClick = () => {
9     let judulSementara = judul;
10    judulSementara.satu = "Belajar React State";
11    setJudul(judulSementara);
12  }
13
14  return (
15    <div>
16      {console.log(judul)}
17      <h1>{judul.satu}</h1>
18      <h1>{judul.dua}</h1>
19      <button onClick={handleButtonClick}>Click me</button>
20    </div>
21  )
22}
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Judul "Belajar React" tidak berubah

Kode ini sangat mirip seperti sebelumnya, hanya saja proses update state sedikit dimodifikasi.

Di baris 9, variabel `judulSementara` diisi dengan nilai state `judul`. Ini berarti kita sedang men-copy semua isi state `judul`. Kemudian di baris 10 saya mengisi property `judulSementara.satu` dengan string "Belajar React State". Hasilnya akan menimpa isi property `judulSementara.satu`

yang sudah ada.

Terakhir, variabel `judulSementara` saya input ke dalam fungsi `setJudul()` agar object state ikut berubah.

Secara logika, tidak ada yang salah dari kode ini. Namun begitu tombol "Click me" di klik, tidak terjadi perubahan apapun. Di dalam tab console juga tidak muncul pesan error.

Hal pertama yang harus kita lakukan saat mendapatkan hasil seperti ini adalah memeriksa apa isi state `judul` setelah fungsi `setJudul()` berjalan. Caranya, tambah perintah `console.log()` ke dalam method `handleButtonClick()`:

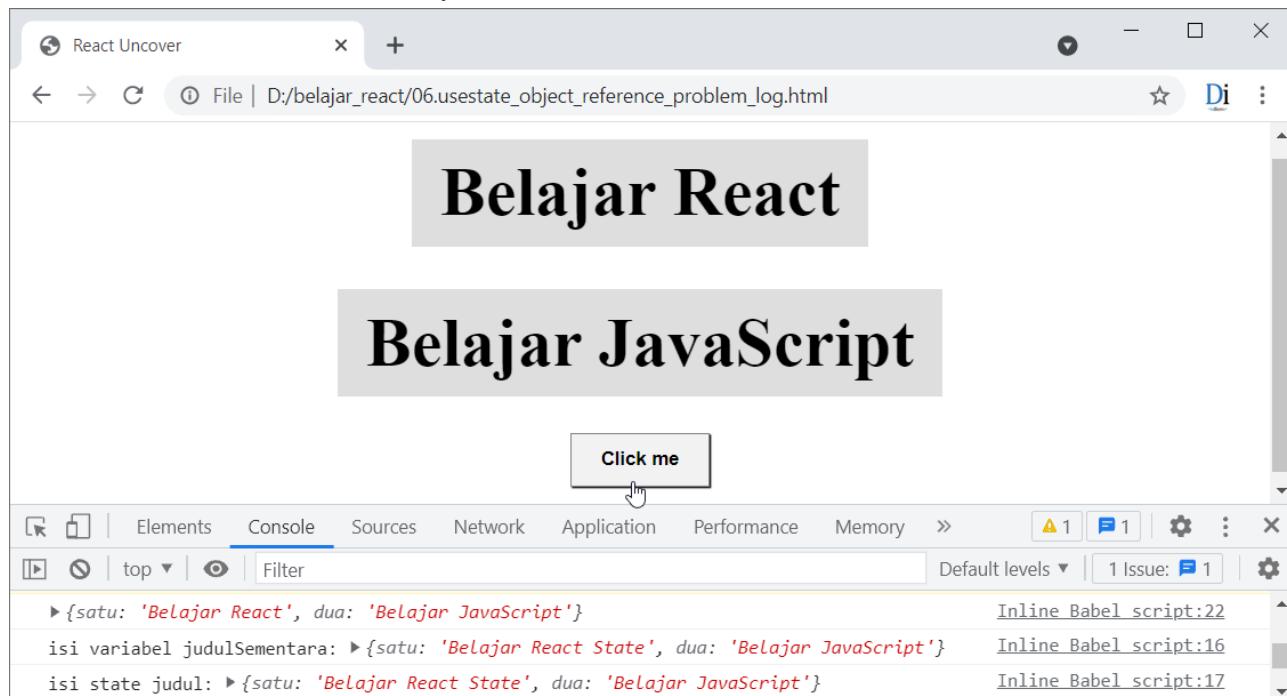
06.usestate_object_reference_problem_log.html

```

1 ...
2 const handleButtonClick = () => {
3     let judulSementara = judul;
4     judulSementara.satu = "Belajar React State";
5     setJudul(judulSementara);
6
7     console.log("isi variabel judulSementara:", judulSementara);
8     console.log("isi state judul:", judul);
9 }
10 ...

```

Tambahan perintah `console.log()` di baris 7-8 akan menampilkan isi variabel `judulSementara` serta state `judul`. Berikut hasilnya:



Gambar: Menampilkan isi variabel judulSementara serta variabel judul

Teks pertama dalam tab Console berasal dari perintah `console.log()` di bagian return

komponen `MyApp`. Ini akan menampilkan isi state awal dan juga saat proses re-render terjadi.

Begitu tombol "Click me" di klik, akan tampil isi variabel `judulSementara` serta state `judul`. Ternyata keduanya sudah berubah. Saya sendiri sempat bingung dan butuh waktu lama untuk mencari apa penyebabnya. State sudah berubah, tapi tidak terjadi proses re-render.

Sumber masalah ini ternyata ada di mekanisme re-render internal React, serta konsep dasar peng-copy-an object di JavaScript.

Seperti yang sudah kita pahami, React akan me-render ulang komponen jika nilai state berubah. Namun untuk `useState` hook, perubahan ini diproses dengan cara melihat *referensi* dari nilai state.

Copy by Value vs Copy by Reference

Penjelasan kita akan sedikit teknis, tapi cukup penting dipahami oleh programmer React dan programmer JavaScript secara umum.

Di JavaScript (dan juga mayoritas bahasa pemrograman lain) terdapat konsep *copy by value* dan *copy by reference*. Kedua istilah ini terjadi saat kita mengcopy suatu variabel dengan operator assignment "`=`".

Untuk tipe dasar (*primitive data type*), JavaScript menggunakan konsep **copy by value**. Maksudnya, yang di copy hanya nilai saja seperti contoh berikut:

`07.copy_by_value_concept.html`

```

1 let foo = "Belajar React";
2 let bar = foo;
3 console.log(foo);           // Belajar React
4 console.log(bar);           // Belajar React
5
6 bar = "Belajar React State";
7 console.log(foo);           // Belajar React
8 console.log(bar);           // Belajar React State
9
10 console.log(foo === bar)   // false

```

Di baris 1, variabel `foo` saya isi dengan string "Belajar React", kemudian `foo` ini dipakai sebagai nilai awal untuk variabel `bar` di baris 2. Disinilah terjadi *copy by value*, perintah `bar = foo` akan **mengcopy nilai** yang ada di dalam variabel `foo` ke dalam variabel `bar`.

Pada saat ditampilkan, kedua variabel berisi string yang sama, yakni "Belajar React".

Di baris 6 isi variabel `bar` saya tukar menjadi "Belajar React State". Perubahan ini hanya terjadi di variabel `bar` saja, sedangkan variabel `foo` tetap berisi "Belajar React".

Operasi perbandingan di baris 10 menghasilkan nilai `false` karena keduanya berisi string yang berbeda.

Sekarang kita coba dengan variabel yang berisi object:

08.copy_by_reference_concept.html

```

1 let foo = { judul: "Belajar React" };
2 let bar = foo;
3 console.log(foo);           // { judul: 'Belajar React' }
4 console.log(bar);          // { judul: 'Belajar React' }
5
6 bar.judul = "Belajar React State";
7 console.log(foo);          // { judul: 'Belajar React State' }
8 console.log(bar);          // { judul: 'Belajar React State' }
9
10 console.log(foo === bar)   // true

```

Di baris 1 saya mengisi variabel `foo` dengan sebuah object. Kemudian di baris 2 variabel `foo` menjadi nilai awal dari variabel `bar`. Seperti yang bisa ditebak, perintah `console.log()` memperlihatkan keduanya berisi nilai yang sama.

Kemudian di baris 6 saya mengganti nilai property `judul` di dalam variabel `bar` menjadi "Belajar React State". Ternyata perubahan ini juga berdampak ke variabel `foo`. Property `judul` dari kedua variabel sekarang sama-sama berisi string "Belajar React State".

Kenapa bisa seperti ini? Penyebabnya adalah perintah `let bar = foo` di baris 2 dijalankan JavaScript dengan konsep **copy by reference**. Yang di copy bukan nilai, tapi *reference* atau alamat memory dari variabel `foo`. Dengan perintah ini, `foo` dan `bar` merujuk ke alamat memory yang sama, sehingga pada saat nilai `bar` ditukar, `foo` juga ikut berubah.

Operasi perbandingan di baris 10 menghasilkan nilai `true` karena kedua variabel merujuk ke object yang sama di memory.

Di JavaScript, *copy by value* berlaku untuk semua tipe data primitive seperti string, number, boolean dan char. Sedangkan *copy by reference* berlaku untuk array dan object.

Kembali ke React, proses update state di deteksi dengan melihat apakah ada perbedaan *reference* dari state awal dengan state akhir. Jika tidak berubah, maka **state tidak akan di render ulang**. Inilah yang menjadi penyebab masalah dari kode program kita sebelumnya.

Dalam JavaScript native, bisa diilustrasikan dengan kode berikut:

09.react_state_comparison_1.html

```

1 let judul = { satu: "Belajar React", dua: "Belajar JavaScript" };
2 let judulSementara = judul;
3 judulSementara.satu = "Belajar React State";
4
5 console.log(judul === judulSementara); // true

```

Kode ini mirip seperti yang ada di dalam method `handleButtonClick()`.

Di baris 1 saya membuat variabel `judul` dan mengisinya dengan sebuah object. Variabel ini kemudian di copy ke dalam variabel `judulSementara`.

Di baris 3, isi property `judulSementara.satu` diubah menjadi "Belajar React State". Akan tetapi meskipun kita hanya mengubah isi variabel `judulSementara` saja, operasi perbandingan di baris 5 menghasilkan nilai `true`. Alasannya, isi variabel `judul` juga ikut berubah karena efek *copy by reference*.

Operasi perbandingan ini bisa dibilang sebagai cara React untuk mendeteksi apakah ada perubahan state atau tidak.

Jadi, bagaimana solusinya? Kita harus ubah *copy by reference* menjadi *copy by value*. Salah satunya dengan memanfaatkan *spread operator*:

10.react_state_comparison_2.html

```

1 let judul = { satu: "Belajar React", dua: "Belajar JavaScript" };
2 let judulSementara = { ...judul };
3 judulSementara.satu = "Belajar React State";
4
5 console.log(judul === judulSementara); // false
6
7 console.log(judul);
8 // {satu: 'Belajar React', dua: 'Belajar JavaScript'}
9
10 console.log(judulSementara);
11 // {satu: 'Belajar React State', dua: 'Belajar JavaScript'}
```

Di baris 2, variabel `judulSementara` diisi dengan `{ ...judul }`. Cara penulisan ini akan "membuka" isi object `judul` untuk diinput ke dalam variabel `judulSementara`, seolah-olah kita membuat object baru, bukan mengcopy variabel `judul`.

Sekarang operasi perbandingan di baris 5 akan menghasilkan nilai `false`, yang artinya ada perbedaan. Perintah `console.log()` di baris 7 dan 10 memperlihatkan isi variabel `judul` tidak lagi sama dengan `judulSementara`.

Inilah solusi dari masalah kita. Silahkan modifikasi method `handleButtonClick()` dengan tambahan *spread operator*:

11.usestate_object_solution_1.html

```

1 ...
2 const handleButtonClick = () => {
3   let judulSementara = { ...judul }; // ini akan mengcopy by value
4   judulSementara.satu = "Belajar React State";
5   setJudul(judulSementara); // sekarang akan di re-render
6 }
7 ...
```

Hasilnya ketika tombol "Click me" di klik, judul akan berubah dari "Belajar React" menjadi "Belajar React State".

Sebagai alternatif penulisan, bisa juga dibuat sebagai berikut:

12.usestate_object_solution_2.html

```
1  ...
2  const handleButtonClick = () => {
3      let judulSementara = judul;          // ini akan mengcopy by reference!!
4      judulSementara.satu = "Belajar React State";
5      setJudul({ ...judulSementara });    // ini akan mengcopy by value
6  }
7  ...
```

Di sini, spread operator saya pakai di baris 5 di dalam method `setJudul()`. Ini akan membuat object baru pas saat mengupdate nilai state `judul`.

Jangan Pakai Nested State?

Beberapa [referensi](#)²⁰ ada yang menyebut kalau tidak disarankan membuat nested object state seperti contoh kita ini. Masalah terkait copy by reference disebabkan React hanya mendukung proses re-render jika state itu terdiri dari 1 nilai saja, bukan berbentuk object.

Penggunaan spread operator juga lebih kepada trik daripada fitur React. Jika anda setuju dengan pendapat ini, maka jangan buat nested object ke dalam state, tapi pisah menjadi beberapa state.

Daripada menulis seperti ini:

```
const [judul, setJudul] = React.useState(
  { satu: "Belajar React", dua: "Belajar JavaScript" }
);
```

Pisah menjadi 2 buah state sebagai berikut:

```
const [judul1, setJudul1] = React.useState("Belajar React");
const [judul2, setJudul2] = React.useState("Belajar JavaScript");
```

Tapi ini lebih ke pendapat saja. Untuk komponen yang butuh banyak state, kadang lebih mudah di kelola jika menggunakan nested object. Kita akan lihat praktiknya saat masuk ke materi tentang pemrosesan form.

9.3. Function Sebagai Nilai State

Dalam materi state di class component, kita pernah bahas bahwa jika proses update state

20. <https://stackoverflow.com/questions/43040721/how-to-update-nested-state-properties-in-react>

butuh nilai dari state sebelumnya, maka gunakan function sebagai nilai state. Ini juga berlaku di functional component.

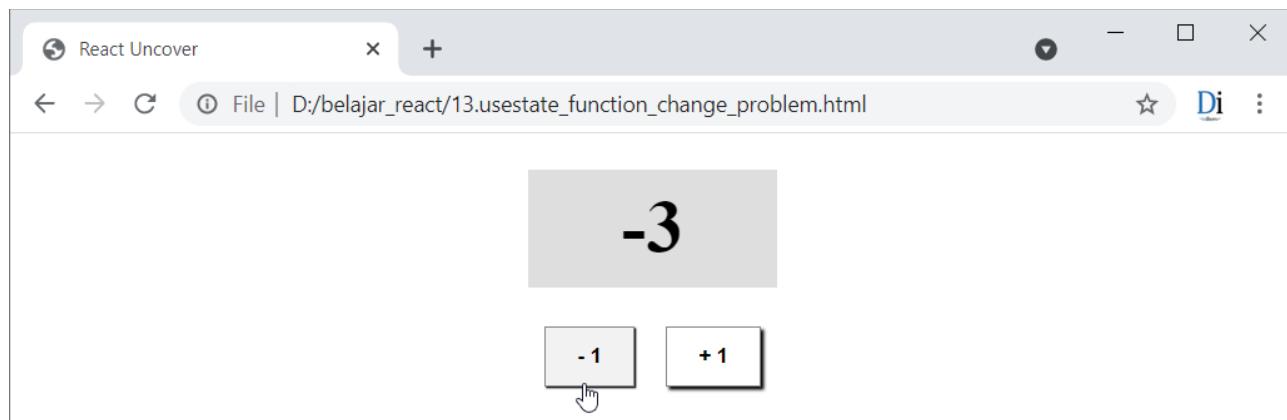
Berikut contoh masalah yang sama, tapi kali ini versi functional component:

13.usestate_function_change_problem.html

```

1  const MyApp = () => {
2      const [counter, setCounter] = React.useState(0);
3
4      const handleDec = () => {
5          setCounter(counter - 1);
6          setCounter(counter - 1);
7      }
8
9      const handleInc = () => {
10         setCounter(counter + 1);
11         setCounter(counter + 1);
12     }
13
14     return (
15         <div>
16             <h1>{counter}</h1>
17             <button onClick={handleDec}> - 1</button>
18             <button onClick={handleInc}> + 1</button>
19         </div>
20     )
21 }
22
23 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Tombol inc/dec hanya naik/turun 1 angka

Kasusnya persis seperti yang terjadi di *class component*. Ketika tombol di klik, counter hanya naik/turun 1 angka, padahal terdapat 2 kali pemanggilan fungsi `setCounter()`. Masalahnya ada di proses update state yang "diantrikan" terlebih dahulu oleh React.

Sebagai solusi, gunakan bentuk penulisan function ketika mengupdate `setCounter()`:

14.usestate_function_change_solution.html

```
1 ...  
2   const handleDec = () => {  
3     setCounter((prevCount) => { return prevCount - 1 });  
4     setCounter((prevCount) => { return prevCount - 1 });  
5   }  
6  
7   const handleInc = () => {  
8     setCounter((prevCount) => { return prevCount + 1 });  
9     setCounter((prevCount) => { return prevCount + 1 });  
10  }  
11 ...
```

Dengan cara ini, variabel `prevCount` berisi nilai state counter sebelumnya. Sekarang ketika tombol di klik, akan langsung lompat 2 angka.

Karena menggunakan penulisan arrow notation, kode di atas bisa ditulis lebih singkat lagi:

15.usestate_function_change_solution_short.html

```
1 ...  
2   const handleDec = () => {  
3     setCounter(prevCount => prevCount - 1);  
4     setCounter(prevCount => prevCount - 1);  
5   }  
6  
7   const handleInc = () => {  
8     setCounter(prevCount => prevCount + 1);  
9     setCounter(prevCount => prevCount + 1);  
10  }  
11 ...
```

Jika *arrow notation* berisi 1 parameter, tanda kurung penutup parameter boleh dihapus. Begitu juga jika isi function hanya terdiri dari 1 baris, tanda kurung kurawal dan perintah `return` juga boleh tidak ditulis.

Ini hanya penulisan alternatif. Jika agak bingung cara membacanya, tidak masalah memakai penulisan yang lebih panjang.

9.4. Mengirim / Mengupdate State dari Child Component

Di akhir bab sebelumnya (state versi *class component*), kita sempat membahas cara mengirim state ke child component (menggunakan `props`), serta mengupdate nilai state dari dalam child (menggunakan `event`). Hal yang sama juga bisa dilakukan di *functional component*.

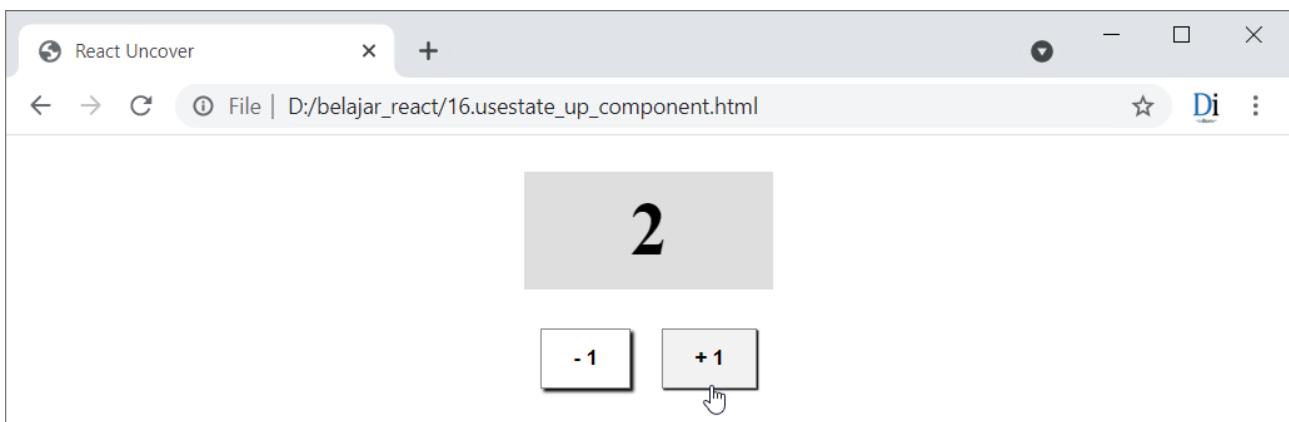
Sebagai contoh praktik, silahkan pelajari sejenak kode program berikut ini:

16.usestate_up_component.html

```
1 const Tombol = (props) => {
```

React useState Hook

```
2  const handleClick = () => {
3    if (props.tombolType === "dec") {
4      props.onTombolClick(-1);
5    }
6    else {
7      props.onTombolClick(+1);
8    }
9  }
10
11 return (
12   <button onClick={handleClick}>{props.children}</button>
13 )
14 }
15
16 const MyApp = () => {
17   const [counter, setCounter] = React.useState(0);
18
19   const handleTombolClick = (change) => {
20     setCounter((prevCount) => { return prevCount + change });
21   }
22
23   return (
24     <div>
25       <h1>{counter}</h1>
26       <Tombol onClick={handleTombolClick} tombolType="dec"> - 1</Tombol>
27       <Tombol onClick={handleTombolClick} tombolType="inc"> + 1</Tombol>
28     </div>
29   )
30 }
31
32 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Membuat counter dengan child component

Dalam kode ini terdapat 2 buah komponen: `MyApp` dan `Tombol`. Komponen `MyApp` dipakai sebagai nilai input untuk method `ReactDOM.createRoot().render()` di baris 32, maka kita mulai membahas dari komponen ini.

Sebagai nilai `return` di dalam komponen `MyApp`, tag `<h1>` diisi dengan variabel `counter` (baris 25), ternyata itu adalah state yang didefinisikan pada baris 17.

Selanjutnya terdapat pemanggilan 2 tag <Tombol>. Karena diawali dengan huruf besar, maka ini merujuk ke komponen Tombol. Sebagai atribut, terdapat `onTombolClick={handleTombolClick}`, dan `tombolType`.

Atribut pertama adalah sebuah *event handler* yang di definisikan pada baris 19-21. Method `handleTombolClick()` butuh satu parameter yang disimpan ke dalam variabel `change`, sehingga perintah apapun yang akan mengakses method ini, butuh menginput 1 argument. Pada saat diakses, fungsi `setCounter()` akan dijalankan untuk akan mengubah nilai state counter sebanyak isi state counter saat ini + isi variabel `change`.

Kembali ke dalam pemanggilan tag <Tombol>, method `handleTombolClick()` dikirim sebagai `props` menggunakan atribut `onTombolClick`. Dengan demikian sesampainya di komponen Tombol, props `onTombolClick` bisa diakses untuk mengubah state counter (teknik *state up*).

Atribut kedua di dalam tag <Tombol> adalah `tombolType`, nilainya berupa string "dec" atau "inc" (akhir baris 26-27).

Masuk ke dalam definisi komponen Tombol di baris 1-14, komponen ini mengembalikan struktur JSX berupa tag <button> dengan atribut `event onClick={handleClick}`, serta menampilkan `{props.children}` sebagai nilai teks.

Pada saat tombol di klik, method `handleClick()` yang ada di baris 2-9 akan di proses. Method ini berisi sebuah pemeriksaan kondisi, yaitu jika `props.tombolType === "dec"` menghasilkan nilai `true`, maka jalankan perintah `props.onTombolClick(-1)`, selain itu jalankan `props.onTombolClick(+1)`.

Jika dipelajari kembali, `onTombolClick` adalah `props` yang berisi method `handleTombolClick()`. Sehingga perintah `props.onTombolClick(-1)` dan `props.onTombolClick(+1)` akan mengupdate nilai state counter milik `MyApp` apakah turun 1 angka (-1), atau naik 1 angka (+1).

Itulah cara mengirim dan mengubah state dari child element di dalam functional component. Teknik yang dipakai kurang lebih sama seperti di class component.

Sebagai materi tambahan, saya ingin melakukan sedikit *refactoring* ke dalam kode kita, sekedar melihat cara lain dalam menulis kode program untuk masalah yang sama.

Misalnya, kondisi if untuk method `handleClick()` bisa ditulis lebih singkat dengan *conditional operator*:

17.usestate_up_component_refactor_1.html

```
1 const handleClick = () => {
2   (props.tombolType === "dec") ? props.onTombolClick(-1) : props.onTombolClick(+1)
3 }
```

Atau jika ingin menghapus pemeriksaan kondisi sama sekali juga bisa, caranya kirim angka "+1" dan "-1" langsung sebagai `props`:

React useState Hook

18.usestate_up_component_refactor_2.html

```
1 const Tombol = (props) => {
2
3     const handleClick = () => {
4         props.onTombolClick(parseInt(props.counterChange));
5     }
6
7     return (
8         <button onClick={handleClick}>{props.children}</button>
9     )
10}
11
12 const MyApp = () => {
13     const [counter, setCounter] = React.useState(0);
14
15     const handleTombolClick = (change) => {
16         setCounter((prevCount) => { return prevCount + change });
17     }
18
19     return (
20         <div>
21             <h1>{counter}</h1>
22             <Tombol onClick={handleTombolClick} counterChange="-1">- 1</Tombol>
23             <Tombol onClick={handleTombolClick} counterChange="+1">+ 1</Tombol>
24         </div>
25     )
26 }
27
28 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Di dalam kode JSX komponen App, saya mengganti atribut tombolType di dalam tag <Tombol> menjadi counterChange (baris 22-23). Atribut ini diisi dengan nilai "-1" dan "+1", sehingga bisa langsung diinput sebagai nilai `props.onTombolClick()` di baris 4.

Tambahan fungsi `parseInt()` diperlukan untuk mengkonversi string "+1" dan "-1" menjadi integer karena akan dipakai dalam operasi penambahan state sesampainya di baris 16.

Masih dirasa belum singkat, perintah `props.onTombolClick()` bisa langsung ditulis ke dalam event `onClick` secara inline:

19.usestate_up_component_refactor_3.html

```
1 const Tombol = (props) => {
2     return (
3         <button onClick={() => props.onTombolClick(parseInt(props.counterChange))}>
4             {props.children}
5         </button>
6     )
7 }
8 ...
```

Dengan menulis event inline di baris 3, kita tidak perlu method `handleClick()` secara terpisah.

Setiap alternatif penulisan memiliki plus-minus masing-masing. Kode yang singkat memang lebih cepat ditulis, tapi kadang sedikit susah untuk dipahami.

Exercise

Pada bahasan yang sama di *class component*, terdapat praktik "tukar mahasiswa". Sebagai latihan, silahkan konversi kode program tersebut ke dalam bentuk *functional component*.

Berikut saya tampilkan kembali kode versi *class component*:

bab 7 - State\35.state_up_argument_2.html

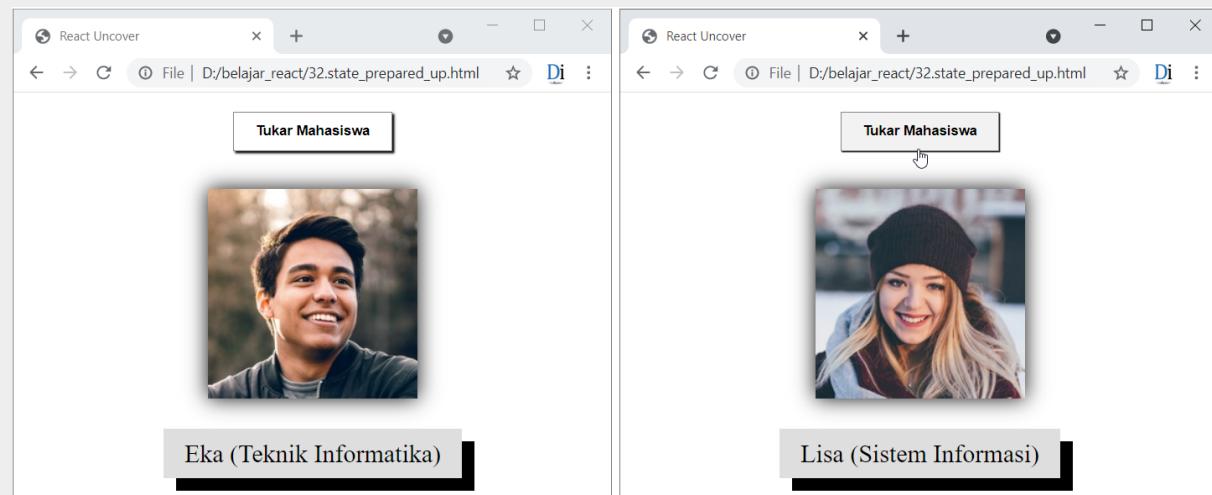
```

1 class Mahasiswa extends React.Component {
2
3     handleTukarMahasiswaBaru = () => {
4         this.props.onTukarMahasiswa({
5             nama: "Lisa",
6             jurusan: "Sistem Informasi",
7             pasFoto: "people2.jpg"
8         })
9     }
10
11    render() {
12        return (
13            <React.Fragment>
14
15                <button onClick={this.handleTukarMahasiswaBaru}>
16                    Tukar Mahasiswa
17                </button>
18
19                <figure>
20                    <img
21                        src={"img/" + this.props.mahasiswa.pasFoto}
22                        alt={this.props.mahasiswa.nama}
23                    />
24                    <figcaption>
25                        {this.props.mahasiswa.nama} ({this.props.mahasiswa.jurusan})
26                    </figcaption>
27                </figure>
28
29            </React.Fragment>
30        )
31    }
32 }
33
34 class MyApp extends React.Component {
35     constructor() {
36         super()
37         this.state = {
38             nama: "Eka",
39             jurusan: "Teknik Informatika",

```

```
40     pasFoto: "people1.jpg"
41   );
42 }
43
44 handleTukarMahasiswa = (mahasiswaBaru) => {
45   this.setState({
46     nama: mahasiswaBaru.nama,
47     jurusan: mahasiswaBaru.jurusan,
48     pasFoto: mahasiswaBaru.pasFoto
49   })
50 }
51
52 render() {
53   return (
54     <div>
55       <Mahasiswa
56         mahasiswa={this.state}
57         onTukarMahasiswa={this.handleTukarMahasiswa}>
58       />
59     </div>
60   )
61 }
62 }
63
64 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Silahkan konversi ke bentuk *functional component*.



Gambar: Tampilan exercies "tukar mahasiswa"

Answer

Berikut kode yang bisa digunakan:

React useState Hook

```
20.mahasiswa_exercise.html

1 const Mahasiswa = (props) => {
2
3   const handleTukarMahasiswa = () => {
4     props.onTukarMahasiswa({
5       nama: "Lisa",
6       jurusan: "Sistem Informasi",
7       pasFoto: "people2.jpg"
8     })
9   }
10
11  return (
12    <React.Fragment>
13      <button onClick={handleTukarMahasiswa}>Tukar Mahasiswa</button>
14
15      <figure>
16        <img
17          src={"img/" + props.mahasiswa.pasFoto}
18          alt={props.mahasiswa.nama}
19        />
20        <figcaption>
21          {props.mahasiswa.nama} ({props.mahasiswa.jurusan})
22        </figcaption>
23      </figure>
24    </React.Fragment>
25  )
26 }
27
28 const MyApp = () => {
29   const [mahasiswa, setMahasiswa] = React.useState({
30     nama: "Eka",
31     jurusan: "Teknik Informatika",
32     pasFoto: "people1.jpg"
33   });
34
35   const handleTukarMahasiswa = (mahasiswaBaru) => {
36     setMahasiswa({
37       nama: mahasiswaBaru.nama,
38       jurusan: mahasiswaBaru.jurusan,
39       pasFoto: mahasiswaBaru.pasFoto
40     })
41   }
42
43   return (
44     <div>
45       <Mahasiswa
46         mahasiswa={mahasiswa}
47         onTukarMahasiswa={handleTukarMahasiswa}
48       />
49     </div>
50   )
51 }
52 }
```

```
53 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Kodenya relatif lebih singkat dan kita juga tidak perlu menulis perintah `this` saat mengakses `state` dan `props`.

9.5. Mencegah State Dibuat Ulang

Prinsip dasar state adalah, jika nilainya berubah, seluruh komponen akan di render ulang. Di dalam `class component`, bagian yang di render ini mencakup kode yang terdapat di method `render()` saja. Kode di bagian lain seperti `constructor` tidak akan dijalankan ulang.

Akan tetapi di `functional component` kondisinya sedikit berbeda. Bagian yang di render ulang mencakup semua isi function, termasuk perintah `React.useState()`.

State sebenarnya sudah memiliki fitur khusus di React, yakni meskipun kodennya dijalankan ulang, isi state tetap dipertahankan dalam setiap proses re-render. Akan tetapi jika nilai state tersebut berasal dari `function`, function itu terus dijalankan ulang setiap kali proses re-render terjadi.

Penjelasan ini akan lebih mudah dengan praktik berikut:

21.usestate_init_function_re_render.html

```
1  const getNumber = () => {
2      console.log("getNumber dijalankan");
3      return 100000;
4  }
5
6  const MyApp = () => {
7      const [judul, setJudul] = React.useState("Belajar React");
8      const [counter, setCounter] = React.useState(getNumber());
9
10     const handleButtonClick = () => {
11         if (judul === "Belajar React") {
12             setJudul("Belajar JavaScript");
13         }
14         else {
15             setJudul("Belajar React");
16         }
17     }
18
19     const handleFooClick = () => {
20         setCounter(prevCounter => prevCounter + 1)
21     }
22
23     return (
24         <div>
25             <h1>{judul} {counter}</h1>
26             <button onClick={handleButtonClick}>Ganti Judul</button>
27             <button onClick={handleFooClick}> + 1 </button>

```

React useState Hook

```
28      </div>
29  )
30 }
31
32 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Melihat efek re-render di functional component

Dalam tag `<h1>` di baris 25, saya menampilkan isi state `judul` dan state `counter` sekaligus. Kedua state di definisikan pada baris 7 dan 8. Setelah itu terdapat 2 buah tag `<button>` dengan event `onClick` yang berbeda.

Pada saat tombol "Ganti Judul" di klik, isi state `judul` akan berganti antara "Belajar React" dengan "Belajar JavaScript". Proses pergantian ditangani oleh method `handleButtonClick()` di baris 10-17.

Untuk tombol "+1", ketika di klik akan menambah nilai state `counter` sebanyak 1 angka. Proses ini dilakukan oleh method `handleFooClick()` di baris 19-21.

Sampai di sini tidak ada masalah, namun perhatikan nilai awal state `counter` di baris 8.

Perintah `React.useState(getNumber())` berarti nilai awal state diambil dari hasil kembalian fungsi bernama `getNumber()`. Kode untuk fungsi ini ada di baris 1-4, berupa sebuah perintah `console.log("getNumber dijalankan")` serta `return 100000`. Artinya ketika fungsi ini diakses, akan tampil teks di tab console dan mengembalikan angka `100000`.

Pada saat dijalankan, di dalam tab console akan tampil teks "getNumber dijalankan" sebagai bukti bahwa fungsi `getNumber()` berhasil di akses. Namun ketika tombol "Ganti Judul" atau "+1" di klik, teks tersebut terus muncul.

Inilah yang menjadi fokus utama kita. Jika teks "getNumber dijalankan" terus tampil, artinya fungsi `getNumber()` selalu dijalankan setiap kali proses re-render terjadi.

Untuk kode sederhana seperti di atas efeknya memang tidak begitu terasa. Akan tetapi jika fungsi `getNumber()` berisi perhitungan yang cukup rumit, maka bisa terjadi jeda setiap kali terdapat perubahan state. State yang berubah tidak harus state `counter`, tapi state apa saja.

Sebagai solusi dari masalah ini adalah, tukar cara pengaksesan fungsi `getNumber()` dari akses langsung menjadi penulisan callback / arrow notation:

```
22.usestate_init_function_re_render_solution.html
```

```
...
const [counter, setCounter] = React.useState(() => getNumber());
...
```

Dengan penulisan seperti ini, fungsi `getNumber()` hanya dijalankan sekali di awal saja. Setelah itu tidak akan dijalankan lagi pada setiap proses re-render.

Situasi seperti ini memang relatif jarang. Biasanya kita mengisi state dengan nilai awal yang sudah pasti, bukan hasil pemanggilan function seperti ini. Namun jika diperlukan, inputlah fungsi tersebut tersebut sebagai *callback* ke dalam perintah `React.useState`.

Huff, selesai juga bahasan kita tentang state di functional component. Jika digabung dengan pembahasan di class component, total sudah 70 halaman!

State memang sangat penting di React, dan karena fitur ini cukup "unik" (tidak ada di JavaScript native), perlu pembahasan yang cukup detail. Penggunaan state juga harus hati-hati karena ada beberapa masalah yang bisa terjadi. Mulai dari bab ini dan selanjutnya, state hampir selalu ada di contoh-contoh pembahasan kita.

Lanjut, materi berikutnya akan membahas tentang **useReff hook**.

10. React ref dan useRef Hook

Di akhir bab sebelumnya sempat dibahas tentang proses re-render serta dampaknya pada pembuatan state. Sekarang kita akan pelajari masalah yang mirip, tapi untuk variabel biasa.

10.1. Pengertian ref dan useRef Hook

Di dalam React, **ref** adalah sebuah variabel untuk menyimpan data yang nilainya tidak terkena efek re-render. Karena sama-sama menyimpan data, **ref** ini menjadi mirip seperti state, akan tetapi **ref** tidak menyebabkan proses re-render saat nilainya diubah.

Dalam kondisi apa **ref** diperlukan? Kita akan bahas dengan masalah yang bisa terjadi di *functional component*. Silahkan pelajari sebentar kode program berikut:

01.state_function_normal.html

```

1  const MyApp = () => {
2    const [judul, setJudul] = React.useState("Belajar React");
3
4    const handleButtonClick = () => {
5      if (judul === "Belajar React") {
6        setJudul("Belajar JavaScript");
7      }
8      else {
9        setJudul("Belajar React");
10     }
11   }
12
13   return (
14     <div>
15       <h1>{judul}</h1>
16       <button onClick={handleButtonClick}>Click me</button>
17     </div>
18   )
19 }
20
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



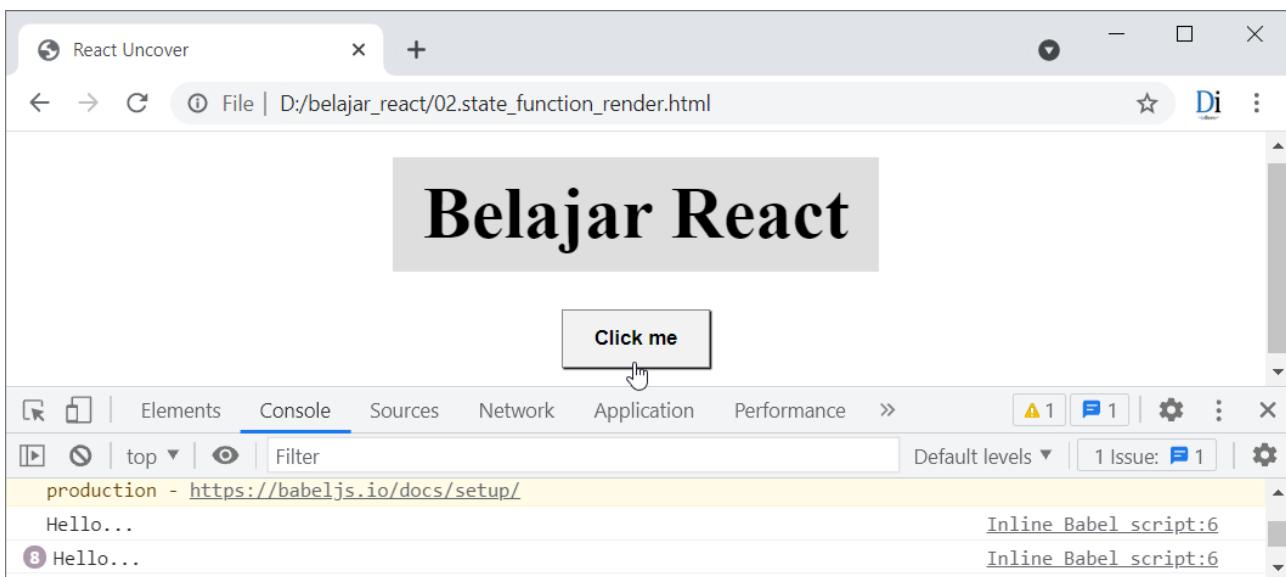
Gambar: Mengganti judul dengan tombol "Click me"

Kode ini sangat mirip seperti contoh pada bab useState. Ketika tombol "Click me" di klik, judul akan berganti dari "Belajar React" ke "Belajar JavaScript" dan sebaliknya.

Di dalam *functional component*, semua kode akan dijalankan ulang setiap kali nilai state berubah (proses re-render). Sebagai pembuktian, silahkan tambah 1 perintah `console.log()` setelah pembuatan state judul:

02.state_function_render.html

```
1 const MyApp = () => {
2   const [judul, setJudul] = React.useState("Belajar React");
3   console.log("Hello...");
4   ...
5 }
```



Gambar: Teks "Hello..." tampil di tab console setiap kali tombol di klik

Hasilnya, di tab console akan tampil teks "Hello..." setiap kali tombol "Click me" di klik. Mekanisme seperti ini berakibat kita tidak bisa menyimpan nilai ke dalam variabel. Berikut contoh prakteknya:

03.state_function_variable.html

```
1 const MyApp = () => {
2   const [judul, setJudul] = React.useState("Belajar React");
3   let foo = "Hei";
4
5   const handleButtonClick = () => {
6     foo = "Hello";
7     if (judul === "Belajar React") {
8       setJudul("Belajar JavaScript");
9     }
10    else {
11      setJudul("Belajar React");
12    }
13  }
14
15  const handleFooClick = () => {
16    console.log(foo);
17  }
18
19  return (
20    <div>
21      <h1>{judul}</h1>
22      <div>
23        <button onClick={handleButtonClick}>Click me</button>
24        <button onClick={handleFooClick}>Get foo</button>
25      </div>
26    </div>
27  )
28}
29
30 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



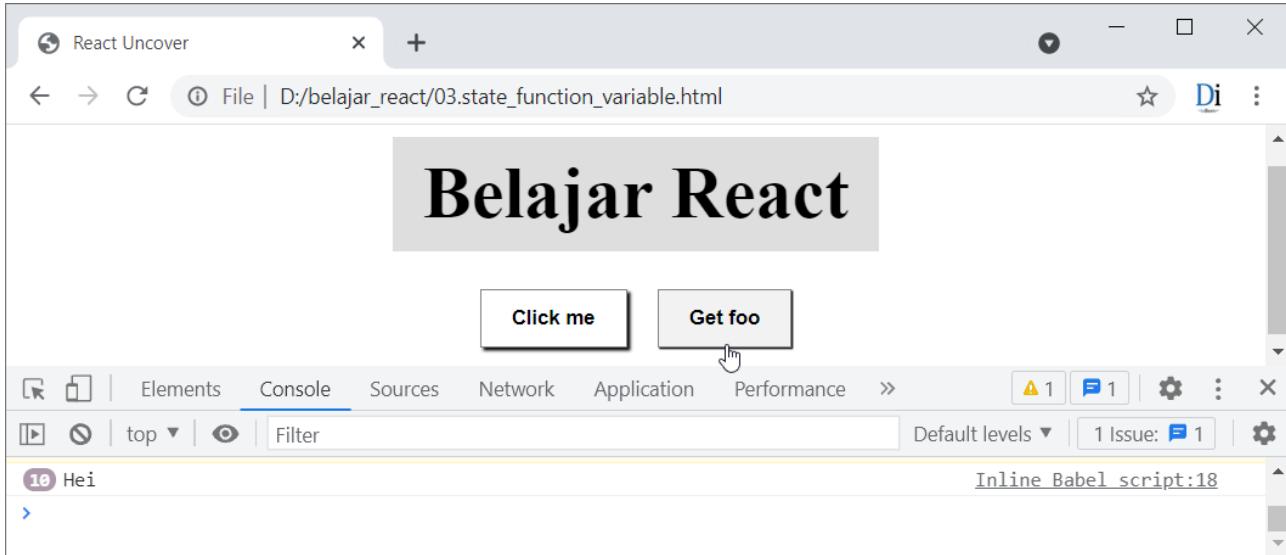
Gambar: Menampilkan isi variabel di dalam functional component

Di baris 24 saya menambah 1 tombol "Get foo". Pada saat di klik, tombol ini akan menjalankan method `handleFooClick()` yang ada di 15-17. Isinya hanya perintah `console.log(foo)` yang berarti tombol "Get foo" sekedar menampilkan isi variabel `foo` saja.

Variabel `foo` sendiri di deklarasikan pada baris 3 dengan string "Hei". Isi `foo` akan diubah dari dalam method `handleButtonClick()` menjadi "Hello".

Sedikit tantangan, bisakah anda menebak apa isi variabel `foo` pada saat halaman tampil pertama kali, dan setelah tombol "Click me" di klik?

Jawabannya, di tab Console akan selalu tampil teks "Hei", tidak pernah "Hello":



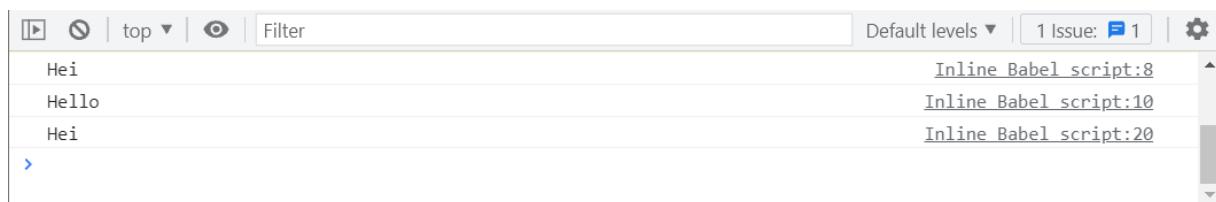
Gambar: Teks yang tampil selalu Hei

Sebenarnya, isi variabel `foo` sudah berubah menjadi "Hello" setiap kali tombol "Click me" di klik. Akan tetapi nilainya ter-reset saat proses re-render. Ingat, setiap perubahan `state`, semua kode yang ada di dalam `MyApp` akan dijalankan ulang.

Ini akan makin jelas dengan menambah perintah `console.log(foo)` sebelum dan sesudah perintah `foo = "Hello"`:

04.state_function_variable_log.html

```
1 ...
2 const handleButtonClick = () => {
3   console.log(foo);
4   foo = "Hello";
5   console.log(foo);
6   if (judul === "Belajar React") {
7     ...
8   }
9 }
```



Gambar: Tampilan tab console saat tombol "Click me" di klik 1x dan tombol "Get foo" di klik 1x

Saat tombol "Click me" di klik, dalam tab console akan tampil teks "Hei" dan "Hello", namun kembali menjadi "Hei" saat tombol "Get foo" di klik.

Dari percobaan ini kita bisa simpulkan bahwa *variabel tidak berperilaku normal di dalam functional component*. Variabel itu tetap bisa ditulis, tetapi nilainya terus di reset ulang karena ada proses re-render.

Untuk masalah inilah **ref** hadir sebagai solusi. Kita bisa ganti proses deklarasi **foo** menjadi **ref** dengan perintah berikut:

```
const foo = React.useRef("Hei");
```

Perintah **React.useRef**, adalah **hook** yang berfungsi untuk membuat **ref**. Dengan menjalankan perintah ini, **foo** akan berisi **object ref**.

Namun string "Hei" tidak bisa diakses langsung dari **foo** saja, tapi harus lewat property **current**:

```
console.log(foo);          // {current: 'Hello...'}
console.log(foo.current); // Hei
```

Berbeda dengan **useState** hook, proses mengupdate nilai **ref** cukup lewat operator assignment biasa " = " ke dalam property **current** ini:

```
foo.current = "Hello...";
console.log(foo.current); // Hello...
```

Sekarang nilai yang tersimpan di dalam **ref** **foo** tidak akan ter-reset saat proses re-render.

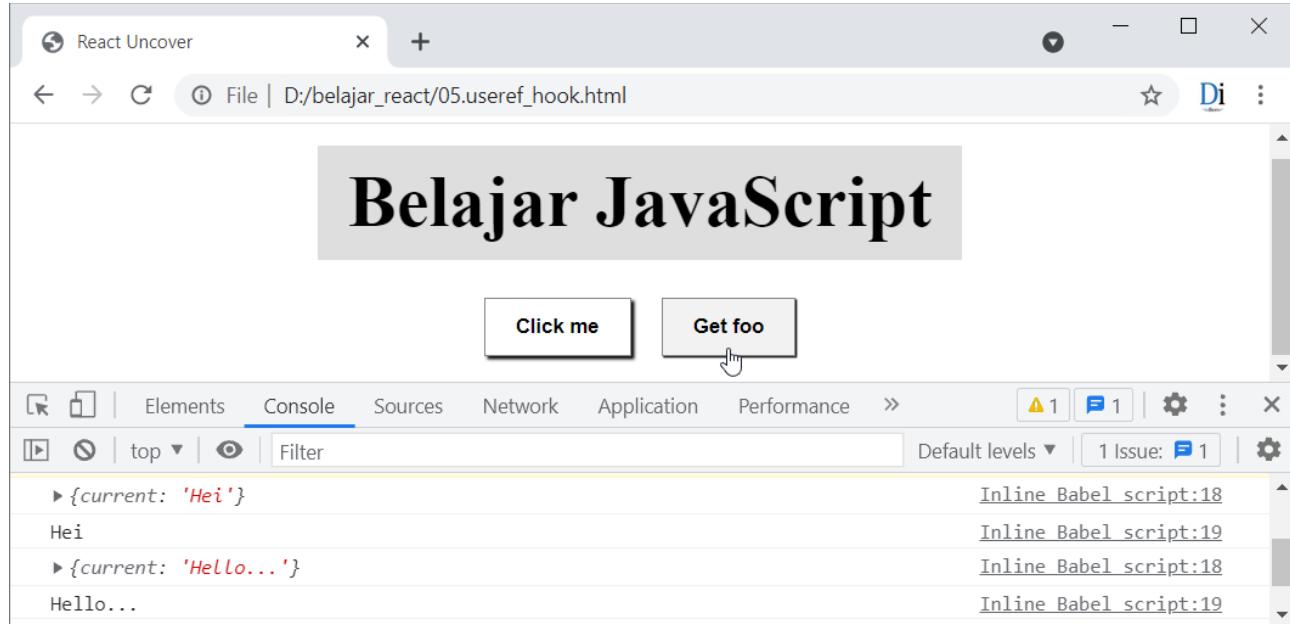
Berikut hasil modifikasi dari kode sebelumnya:

05.useref_hook.html

```
1 const MyApp = () => {
2   const [judul, setJudul] = React.useState("Belajar React");
3   const foo = React.useRef("Hei");
4
5   const handleButtonClick = () => {
6     foo.current = "Hello...";
7     if (judul === "Belajar React") {
8       setJudul("Belajar JavaScript");
9     }
10    else {
11      setJudul("Belajar React");
12    }
13  }
14
15  const handleFooClick = () => {
16    console.log(foo);
17    console.log(foo.current);
18  }
19
20  return (
21    <div>
22      <h1>{judul}</h1>
```

React ref dan useRef Hook

```
23     <div>
24         <button onClick={handleButtonClick}>Click me</button>
25         <button onClick={handleFooClick}>Get foo</button>
26     </div>
27   </div>
28 )
29 }
30
31 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Hasil tampilan setelah foo diubah menjadi ref

Di baris 3, variabel `foo` saya definisikan sebagai `ref` dengan nilai awal "Hei". Nilai ini akan di update saat tombol "Click me" di klik melalui perintah `foo.current = "Hello..."` di baris 6.

Ketika di jalankan pertama kali dan tombol "Get foo" di klik, isi ref `foo` masih "Hei" karena itulah nilai awal yang kita isi. Begitu tombol "Click me" di klik, nilainya akan berubah menjadi "Hello..." dan terus di pertahankan meskipun terjadi proses re-render (tombol "Click me" di klik berkali-kali).

10.2. Ref di Class Component

Sebagai pembanding di class component, `ref` ini tidak lain adalah `class property` yang biasa kita tulis dari dalam constructor.

06.class_state.html

```
1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { judul: "Belajar React" };
5      this.foo = "Hei";
```

```

6    }
7
8    handleButtonClick = () => {
9      this.foo = "Hello...";
10     if (this.state.judul === "Belajar React") {
11       this.setState({ judul: "Belajar JavaScript" });
12     }
13     else {
14       this.setState({ judul: "Belajar React" });
15     }
16   }
17
18   handleFooClick = () => {
19     console.log(this.foo);
20   }
21
22   render() {
23     return (
24       <div>
25         <h1>{this.state.judul}</h1>
26         <div>
27           <button onClick={this.handleButtonClick}>Click me</button>
28           <button onClick={this.handleFooClick}>Get foo</button>
29         </div>
30       </div>
31     )
32   }
33 }
34
35 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Proses pembuatan property `foo` ada di baris 5, lalu di update dari dalam method `handleButtonClick()` di baris 9.

Di dalam `class component`, property tidak terkena proses reset karena yang akan di jalankan ulang (terkenda proses re-render) hanya perintah yang ada di dalam method `render()` saja. Jadi, `ref` di `functional component` hanya sebagai pengganti property biasa di `class component`.

Itulah perkenalan singkat kita dengan `ref` dan `useRef hook`. Nantinya, ref ini bisa berfungsi untuk mengakses struktur DOM secara langsung yang akan kita bahas dalam bab terpisah.

Berikutnya kita akan bahas tentang cara pemrosesan form menggunakan React.

11. React Form Processing

Pemrosesan form termasuk fitur paling kompleks dalam sebuah aplikasi web. React menghadirkan cara yang lebih mudah dibandingkan JavaScript native, dan inilah yang akan kita bahas secara detail sepanjang bab ini.

11.1. Mengakses Nilai Form (Value Property)

Langkah paling awal untuk bisa memproses form adalah mengambil nilai dari inputan form tersebut. Menggunakan JavaScript native (atau tepatnya DOM web browser), nilai form bisa diambil dari property `value` milik `node object` inputan tersebut.

Sebagai contoh jika terdapat kode HTML berikut: `<input type="text" id="input" />`, nilainya bisa diperoleh dari `document.getElementById("input").value`.

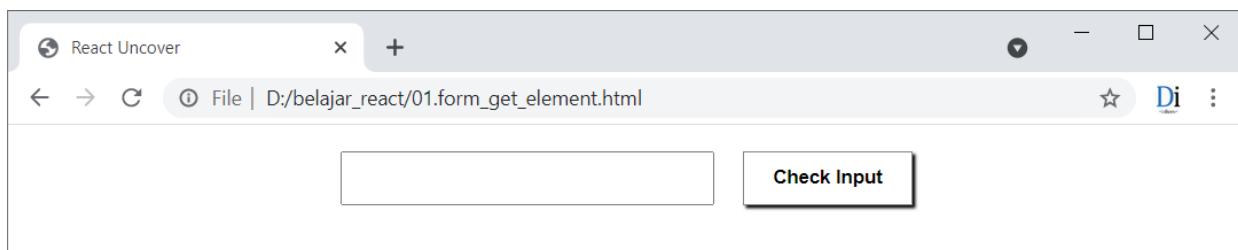
Berikut contoh menggunakan React class component:

01.form_get_element.html

```
36 <!DOCTYPE html>
37 <html lang="id">
38
39 <head>
40   <meta charset="UTF-8">
41   <meta name="viewport" content="width=device-width, initial-scale=1.0">
42   <title>React Uncover</title>
43   <style>
44     #root,
45     div {
46       display: flex;
47       justify-content: center;
48     }
49
50     input {
51       font-size: 1.3em;
52       margin: 10px;
53     }
54
55     button {
56       background-color: white;
57       cursor: pointer;
58       padding: 10px 20px;
59       font-weight: bold;
60       border: 1px solid gray;
```

React Form Processing

```
61     box-shadow: 2px 2px 2px black;
62     margin: 10px;
63 }
64
65 button:hover {
66     box-shadow: 1px 1px 1px black;
67     background-color: #f1f1f1;
68 }
69 </style>
70 </head>
71
72 <body>
73     <div id="root"></div>
74
75     <script src="js/react.development.js"></script>
76     <script src="js/react-dom.development.js"></script>
77     <script src="js/babel.js"></script>
78     <script type="text/babel">
79
80         class MyApp extends React.Component {
81
82             handleButtonClick() {
83                 let inputName = document.getElementById("input").value;
84                 console.log(inputName);
85             }
86
87             render() {
88                 return (
89                     <div>
90                         <input type="text" id="input" />
91                         <button onClick={this.handleButtonClick}> Check Input </button>
92                     </div>
93                 )
94             }
95         }
96
97         ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
98
99     </script>
100 </body>
101
102 </html>
```



Gambar: Tampilan form input dan sebuah tombol

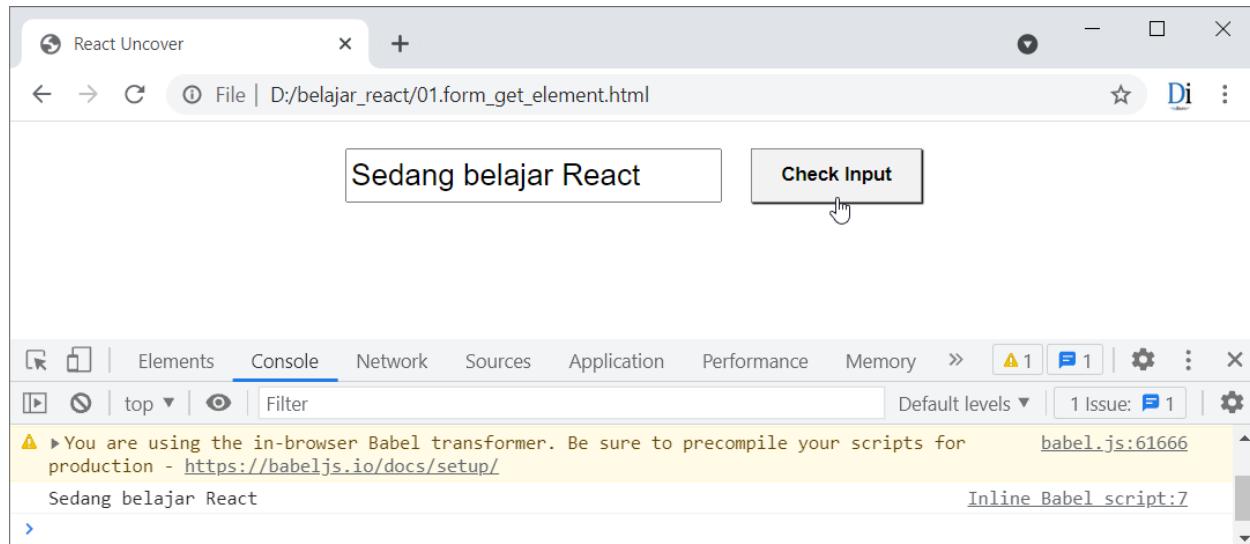
Agar tampilan form lebih menarik, saya menambah sedikit kode CSS di baris 8-34. Kode yang

sama juga akan dipakai pada contoh-contoh berikutnya.

Masuk ke kode React di baris 45, terdapat pendefinisian komponen `MyApp`. Di dalam method `render()`, komponen ini menampilkan 1 tag `<input>`, dan 1 tag `<button>`.

Pada saat tombol "Check Input" di klik, method `handleButtonClick()` akan berjalan, isinya berupa perintah untuk menyimpan nilai `document.getElementById("input").value` ke dalam variabel `inputName`, lalu ditampilkan dengan perintah `console.log(inputName)`.

Silahkan ketik sembarang teks, lalu klik tombol "Check Input":



Gambar: Nilai inputan form berhasil di akses

Setiap kali tombol "Check Input" di klik, nilai yang ada di inputan form akan tampil ke dalam tab Console. Ini merupakan cara umum yang kita lakukan di JavaScript native, yakni mengakses nilai form lewat property `value`.

11.2. Menggunakan onChange Event

Sekarang kita bahas cara React mengakses nilai form dengan memanfaatkan event **onChange**.

Di dalam JavaScript native, event `onChange` akan berjalan saat terjadi perubahan nilai. Misalnya ketika kita mengetik atau menghapus satu karakter ke dalam inputan teks box, event `onChange` ter-trigger.

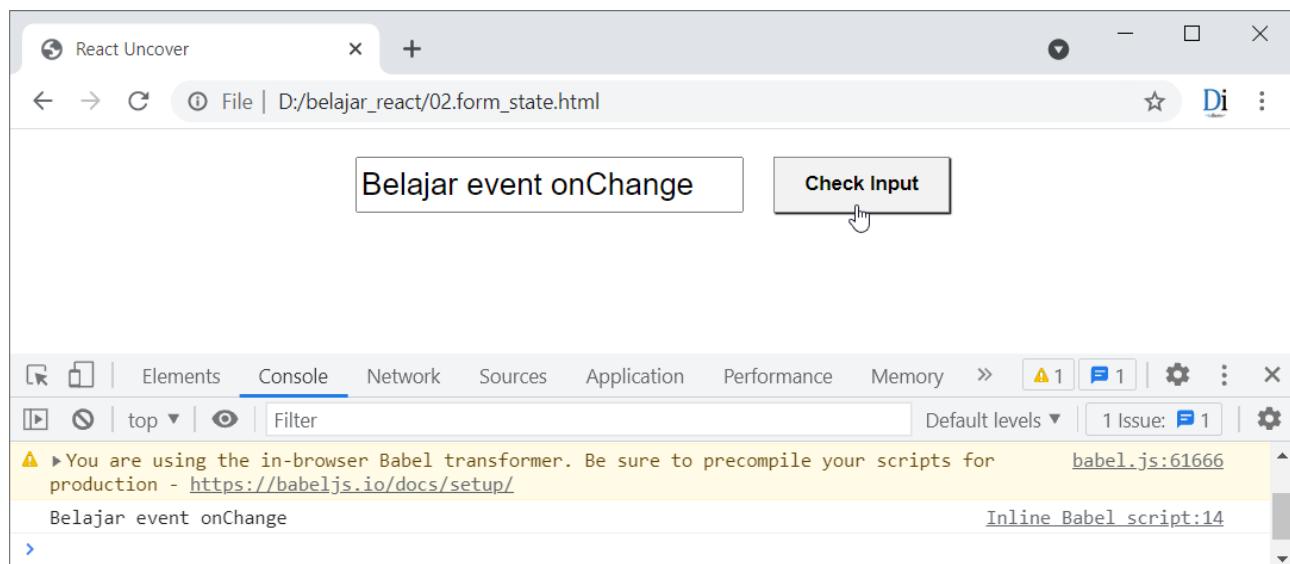
Idenya adalah, setiap kali terjadi perubahan (`onChange`), update nilai inputan tersebut ke dalam sebuah state. Berikut praktek dari penjelasan ini:

02.form_state.html

```
1 class MyApp extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = { input: "" };
```

React Form Processing

```
5     }
6
7     handleInputChange = (event) => {
8       this.setState({ input: event.target.value })
9     }
10
11    handleClick = () => {
12      console.log(this.state.input);
13    }
14
15    render() {
16      return (
17        <div>
18          <input type="text" onChange={this.handleInputChange} />
19          <button onClick={this.handleClick}> Check Input </button>
20        </div>
21      )
22    }
23  }
24
25 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengakses nilai form dari event onChange

Terdapat beberapa modifikasi dari kode kita sebelumnya. Di dalam tag `<input>`, saya menambah atribut `onChange={this.handleInputChange}` di baris 18. Artinya setiap kali terjadi perubahan nilai, method `handleInputChange()` akan berjalan.

Method `handleInputChange()` ada di baris 7-9, method ini menerima `event object` yang ditampung ke dalam parameter `event`. Isi method hanya 1 perintah yang menjalankan `this.setState({ input: event.target.value })`.

Property `event.target.value` merujuk ke `value` dari form yang saat ini sedang terjadi event `onChange`. Perintah inilah yang akan terus mengupdate nilai state input dengan nilai inputan dari form. State input sendiri didefinisikan dari dalam constructor di baris 4.

Isi method `handleButtonClick()` juga sedikit berubah, kali ini akan menampilkan perintah `console.log(this.state.input)`. Dengan kata lain, akan menampilkan isi dari state `input`.

Hasilnya kurang lebih sama seperti contoh kita sebelumnya, setiap kali tombol "Check Input" di klik, teks yang ada di inputan form juga tampil di tab Console.

Silahkan pahami sejenak alur kerja dari kode program ini, karena menjadi konsep dasar pengambilan nilai form menggunakan React.

11.3. Menampilkan Nilai Form Secara Realtime

Kita sudah berhasil menghubungkan event `onChange` dengan `state`, maka jika nilai `state` tersebut ditampilkan langsung ke dalam method `render()`, nilai form bisa tampil *realtime*:

03.form_realtime.html

```
1  class MyApp extends React.Component {  
2      constructor(props) {  
3          super(props);  
4          this.state = { input: "" };  
5      }  
6  
7      handleInputChange = (event) => {  
8          this.setState({ input: event.target.value })  
9      }  
10  
11     render() {  
12         return (  
13             <div>  
14                 <h1>{this.state.input}</h1>  
15                 <input type="text" onChange={this.handleInputChange} />  
16             </div>  
17         )  
18     }  
19 }  
20  
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengakses nilai form secara realtime

Di baris 14, terdapat perintah JSX `<h1>{this.state.input}</h1>` untuk menampilkan isi state `input`. Karena nilai state selalu di update setiap kali kita mengetik sesuatu ke dalam form (efek event `onChange`), maka isi teks di tag `<h1>` juga akan diupdate secara realtime.

Inilah cara React untuk mengakses nilai inputan form, yakni menghubungkan hasil event `onChange` dengan state. State akan selalu update setiap kali inputan form berubah.

Nilai Form Berubah = Proses Re-render?

Sebenarnya menghubungkan event `onChange` dengan state memiliki efek samping. Jika anda masih ingat, prinsip dasar state adalah setiap kali nilainya berubah, komponen itu akan di render ulang. Maka setiap kali inputan form berubah (setiap kali karakter di ketik), akan terjadi proses re-render.

Sekilas ini sangat boros performa, tapi dengan teknik optimasi dari React, teknik ini malah di sarankan untuk setiap inputan form. Tipsnya, pisahkan komponen form dengan komponen lain agar proses re-render hanya terjadi untuk komponen form saja, tidak untuk semua tampilan web.

11.4. Mengubah Nilai Form dari State

Karena state sudah tersambung ke nilai form, kita juga bisa mengubah nilai form dengan cara mengubah nilai statenya. Berikut praktek dari konsep ini:

04.form_change.html

```

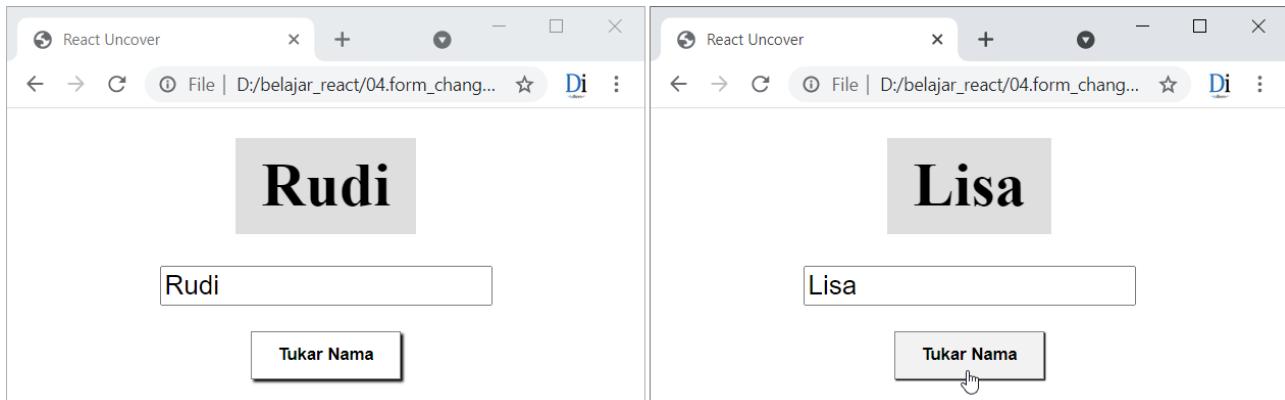
1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { input: "Rudi" };
5      }
6
7      handleInputChange = (event) => {
8          this.setState({ input: event.target.value })
9      }
10
11     handleButtonClick = () => {
12         this.setState({ input: "Lisa" })
13     }
14
15     render() {
16         return (
17             <div>
18                 <h1>{this.state.input}</h1>
19                 <input type="text" value={this.state.input}
20                     onChange={this.handleInputChange} />
21                 <button onClick={this.handleButtonClick}>Tukar Nama</button>
22             </div>

```

```

23     )
24   }
25 }
26
27 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Menukar inputan form dari state

Di baris 19 saya menambah atribut `value={this.state.input}` ke dalam tag `<input>`. Di HTML, atribut `value` berfungsi untuk menampilkan teks (biasanya sebagai nilai awal form). Karena sekarang isi atribut `value` merujuk ke state `input`, maka setiap kali terjadi perubahan `this.state.input`, nilai teks di inputan form juga akan berubah.

Di baris 4, saya mengisi state `input` dengan nilai awal "Rudi". Maka ketika halaman diakses pertama kali, inputan form juga akan berisi string "Rudi".

Kemudian di baris 21, terdapat tombol yang ketika diklik akan menjalankan method `handleButtonClick()` di baris 11-13. Isinya mengganti isi state `input` menjadi "Lisa".

Silahkan tes isi teks sembarang ke dalam tag `<input>` lalu klik tombol ini, nilai inputan akan langsung berubah menjadi "Lisa".

Controlled Components

React menyebut teknik menghubungkan atribut `value` form dengan `state` sebagai **controlled components**. Disebut seperti itu karena React-lah yang akan mengontrol isi teks form. State menjadi "*single source of truth*", jika kita ingin mengganti nilai teks form, cukup mengakses state saja.

Dalam bab terpisah, nantinya kita juga akan pelajari cara memproses form dengan konsep *uncontrolled components*.

11.5. Mengakses Nilai Textarea, Select, Checkbox dan Radio

HTML menyediakan banyak jenis inputan form. Untuk tampilan yang berbentuk `text box` seperti `<input type="text">`, `<input type="password">` atau `<input type="email">`, atribut

value bisa dipakai untuk dihubungkan dengan state. Ini sama persis seperti contoh kita sebelumnya.

Di luar itu, terdapat inputan textarea, select, checkbox dan radio. Kita akan bahas cara memprosesnya sebagai controlled components React.

Mengakses Nilai Textarea

Di HTML, nilai untuk tag <textarea> biasa ditulis antara tag pembuka dan tag penutup seperti: <textarea>nilai awal di sini...</textarea>. React menyeragamkan cara mengakses nilai textarea dengan tetap menggunakan atribut value. Berikut contoh prakteknya:

05.form_text_area.html

```

1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { alamat: "Jakarta" };
5    }
6
7    handleAlamatChange = (event) => {
8      this.setState({ alamat: event.target.value })
9    }
10
11   render() {
12     return (
13       <div>
14         <h1>{this.state.alamat}</h1>
15         <textarea
16           value={this.state.alamat}
17           onChange={this.handleAlamatChange}
18         />
19       </div>
20     )
21   }
22 }
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses nilai textarea

Karena tetap menggunakan atribut `value`, caranya sangat mirip seperti tag `<input type="text">`, yakni tempatkan pasangan atribut `value={this.state.alamat}` dan `onChange={this.handleAlamatChange}` ke dalam tag `<textarea>`.

Atribut `value={this.state.alamat}` di baris 16 dipakai untuk mengupdate isi teks textarea berdasarkan nilai state, sedangkan atribut `onChange={this.handleAlamatChange}` di baris 17 berfungsi untuk menjalankan method `this.setState()` untuk mengupdate nilai state untuk setiap event `onChange`.

Jika dilihat ke dalam inspect element web browser, teks tersebut tetap tampil di antara tag pembuka dan penutup `<textarea>`, React sudah memprosesnya secara internal:



Gambar: Kode HTML yang di generate React

Mengakses Nilai Select

Di JavaScript native, mengakses nilai tag `<select>` bisa cukup kompleks karena atribut `value` berada di dalam *child element* tag `<option>`, bukan di tag `<select>` langsung.

Akan tetapi sama seperti textarea, React menyederhanakan proses ini dengan mengizinkan kita mengakses atribut `value` langsung dari tag `<select>`. React otomatis akan mengisi atribut ini dengan nilai yang diambil dari tag `<option>`.

Berikut cara mengakses nilai tag `<select>` menggunakan React:

06.form_select.html

```

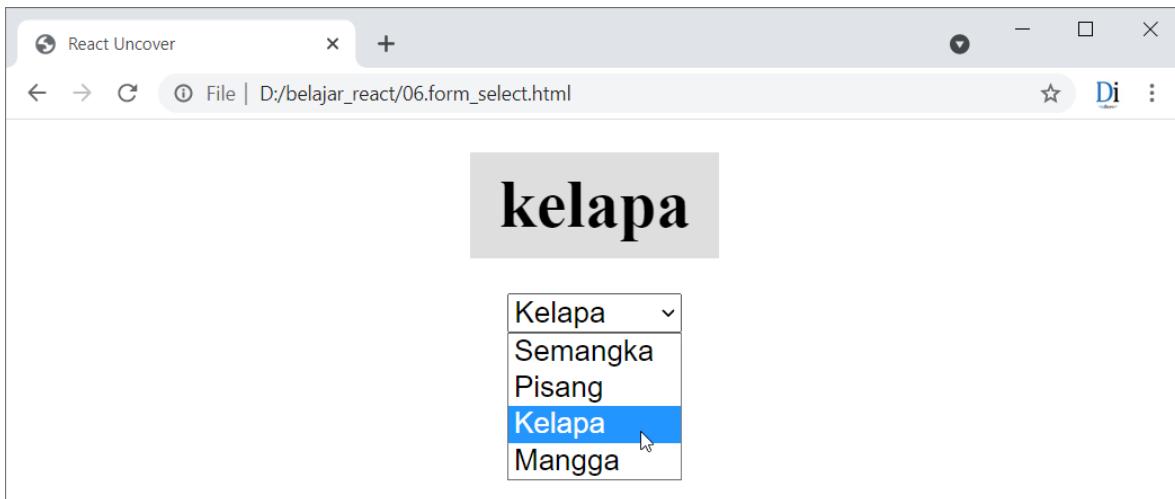
1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { buah: "mangga" };
5    }
6
7    handleBuahChange = (event) => {
8      this.setState({ buah: event.target.value })
9    }
10
11   render() {
12     return (
13       <div>
14         <h1>{this.state.buah}</h1>

```

```

15      <select value={this.state.buah} onChange={this.handleBuahChange}>
16          <option value="semangka">Semangka</option>
17          <option value="pisang">Pisang</option>
18          <option value="kelapa">Kelapa</option>
19          <option value="mangga">Mangga</option>
20      </select>
21  </div>
22 )
23 }
24 }
25
26 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses nilai tag <select>

Antara baris 15-20 terdapat tag `<select>` dengan berbagai nilai buah-buahan. Setiap tag `<option>` memiliki atribut `value` yang berisi nilai inputan form. Pada saat form di submit dengan cara biasa, nilai inilah yang akan dikirim.

Proses menghubungkan tag `<select>` dengan React state ada di baris 14, yakni lewat atribut `value={this.state.buah}` dan `onChange={this.handleBuahChange}`. Dengan penulisan ini, React akan mengupdate isi state buah sesuai nilai atribut `value` dari tag `<option>` yang sedang terpilih.

Sebagai bukti, silahkan tukar pilihan buah-buahan, maka isi tag `<h1>` juga ikut berubah.

Nilai untuk atribut `value` di tag `<select>` juga berjalan 2 arah. Jika kita mengubah isi state, pilihan tag `<select>` juga akan menyesuaikan. Berikut prakteknya:

07.form_select_change.html

```

1 class MyApp extends React.Component {
2     constructor(props) {
3         super(props);
4         this.state = { buah: "mangga" };
5     }
6

```

React Form Processing

```
7  handleBuahChange = (event) => {
8      this.setState({ buah: event.target.value })
9  }
10
11 handleButtonClick = () => {
12     this.setState({ buah: "semangka" })
13 }
14
15 render() {
16     return (
17         <div>
18             <h1>{this.state.buah}</h1>
19             <select value={this.state.buah} onChange={this.handleBuahChange}>
20                 <option value="semangka">Semangka</option>
21                 <option value="pisang">Pisang</option>
22                 <option value="kelapa">Kelapa</option>
23                 <option value="mangga">Mangga</option>
24             </select>
25             <button onClick={this.handleButtonClick}>Pilih Semangka</button>
26         </div>
27     )
28 }
29 }
30
31 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengubah pilihan tag <select>

Tambahan kode program ada di baris 25 dan 11-13. Di baris 25, saya membuat sebuah tombol "Pilih Semangka" yang ketika di klik, akan menjalankan perintah `this.setState({ buah: "semangka" })` di dalam method `handleButtonClick()`.

Perintah ini akan mengupdate state buah menjadi "semangka". Dan karena state buah terhubung ke dalam tag `<select>`, maka ketika di klik, pilihan tag `<select>` juga bertukar menjadi "Semangka".

Dengan bantuan atribut `value` di dalam tag `<select>`, proses pengambilan nilai menjadi lebih sederhana.

Mengakses Nilai Checkbox

Untuk inputan checkbox, yang dihubungkan ke state adalah nilai atribut checked. Berikut contoh penggunaannya:

08.form_checkbox.html

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { belajarReact: true };
5      }
6
7      handleBelajarReactChange = (event) => {
8          this.setState({ belajarReact: event.target.checked })
9      }
10
11     render() {
12         return (
13             <div>
14                 <h1>{this.state.belajarReact.toString()}</h1>
15                 <input
16                     type="checkbox"
17                     id="belajarReact"
18                     checked={this.state.belajarReact}
19                     onChange={this.handleBelajarReactChange}
20                 />
21                 <label htmlFor="belajarReact">Belajar React</label>
22             </div>
23         )
24     }
25 }
26
27 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses nilai atribut checked

Di baris 18-19, terdapat atribut `checked={this.state.belajarReact}` yang berfungsi untuk menghubungkan checkbox dengan state `belajarReact`, serta atribut `onChange={this.handleBelajarReactChange}` untuk proses update nilai state.

Perhatikan juga perintah di baris 8, property yang kita akses bukan lagi `event.target.value`, tapi `event.target.checked`. Nilai yang akan diterima berupa boolean `true` jika checkbox di

centang, atau boolean `false` jika checkbox tidak dipilih. Silahkan tes kode program di atas dan isi teks tag `<h1>` akan berubah antara `true` dan `false`.

Mengakses Nilai Radio Button

Radio button umumnya terdiri dari beberapa inputan yang saling terhubung dan kita hanya bisa memilih salah satu nilai saja.

Dalam banyak hal, radio button mirip seperti checkbox karena butuh atribut `checked`. Akan tetapi karena radio saling terhubung dengan inputan lain, cara mengakses nilainya jadi sedikit kompleks.

Untuk proses *binding* (membuat hubungan) antara event `onChange` dengan React state, bisa ditempatkan ke dalam parent element dari beberapa tag `<input type="radio">`. Berikut contoh prakteknya:

09.form_radiobutton.html

```

1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { materi: "" };
5    }
6
7    handleMateriChange = (event) => {
8      this.setState({ materi: event.target.value })
9    }
10
11   render() {
12     return (
13       <div>
14         <h1>{this.state.materi}</h1>
15         <div onChange={this.handleMateriChange}>
16           <input type="radio" value="React" name="materi" id="materi1" />
17           <label htmlFor="materi1">Belajar React</label>
18           <input type="radio" value="JavaScript" name="materi" id="materi2" />
19           <label htmlFor="materi2">Belajar JavaScript</label>
20           <input type="radio" value="Front End" name="materi" id="materi3" />
21           <label htmlFor="materi3">Belajar Front End</label>
22         </div>
23       </div>
24     )
25   }
26 }
27
28 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses nilai radio button

Antara baris 16-21 saya membuat 3 buah pasangan tag `<input type="radio">` dan `<label>`. Tambahan tag `<label>` sekedar memudahkan proses memilih radio dengan cara men-klik teks nya saja (tidak harus pas men-klik tombol radio). Di setiap radio button terdapat atribut `value` yang akan menjadi nilai saat radio button di pilih.

Proses update nilai form ke state dilakukan dari atribut `onChange={this.handleMateriChange}` di dalam tag `<div>` pada baris 15. Perhatikan bahwa event `onChange` ini berada dalam *parent element* dari ketiga radio button, bukan langsung ke setiap tag `<input>`. Ini mirip seperti yang dipakai pada tag `<select>`, meskipun jika ditempatkan berulang ke setiap tag `<input>` juga tidak masalah.

React secara internal akan mengambil nilai atribut `value` dari radio button yang sedang dipilih, dan bisa diakses dari `event.target.value` di baris 8. Dengan cara ini, state `materi` akan berisi salah satu dari string "React", "JavaScript" atau "Front End" sesuai nilai atribut `value` di setiap radio button.

Kode di atas baru satu arah, yakni dari radio button ke dalam state. Untuk arah sebaliknya, perlu sedikit trik karena jika kita langsung mengisi state `materi` dengan string "JavaScript", itu tidak otomatis memilih radio button "JavaScript". Kita harus buat sebuah kondisi pemeriksaan di setiap radio button:

`10.form_radiobutton_checked.html`

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { materi: "JavaScript" };
5      }
6
7      handleMateriChange = (event) => {
8          this.setState({ materi: event.target.value })
9      }
10
11     render() {
12         return (
13             <div>
14                 <h1>{this.state.materi}</h1>

```

```

15     <div>
16         <input type="radio" name="materi" id="materi1" value="React"
17             checked={this.state.materi === "React"}
18             onChange={this.handleMateriChange}
19         />
20         <label htmlFor="materi1">Belajar React</label>
21         <input type="radio" name="materi" id="materi2" value="JavaScript"
22             checked={this.state.materi === "JavaScript"}
23             onChange={this.handleMateriChange}
24         />
25         <label htmlFor="materi2">Belajar JavaScript</label>
26         <input type="radio" name="materi" id="materi3" value="Front End"
27             checked={this.state.materi === "Front End"}
28             onChange={this.handleMateriChange}
29         />
30         <label htmlFor="materi3">Belajar Front End</label>
31     </div>
32 </div>
33 )
34 }
35 }
36
37 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Untuk setiap radio button, terdapat tambahan atribut `checked` di baris 17, 22 dan 27. Isinya, memeriksa apakah `this.state.materi` berisi string "React", "JavaScript" atau "Front End". Jika kondisi ini bernilai `true`, maka radio button tersebut akan terpilih.

Di baris 4 saya langsung mengisi state `materi` dengan string "JavaScript". Maka ketika halaman tampil, radio button yang valuenya berisi string "JavaScript" akan langsung terpilih (`ter-checked`).

11.6. Mengakses Nilai Form Utuh

Sejak awal bab kita telah pelajari cara mengakses nilai berbagai jenis inputan form. Sekarang mari satukan semua inputan tersebut menjadi sebuah form utuh. Kodanya memang cukup panjang karena saya menambah sedikit kode CSS serta mencakup 6 inputan form:

```

11.form_all.html

1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;

```

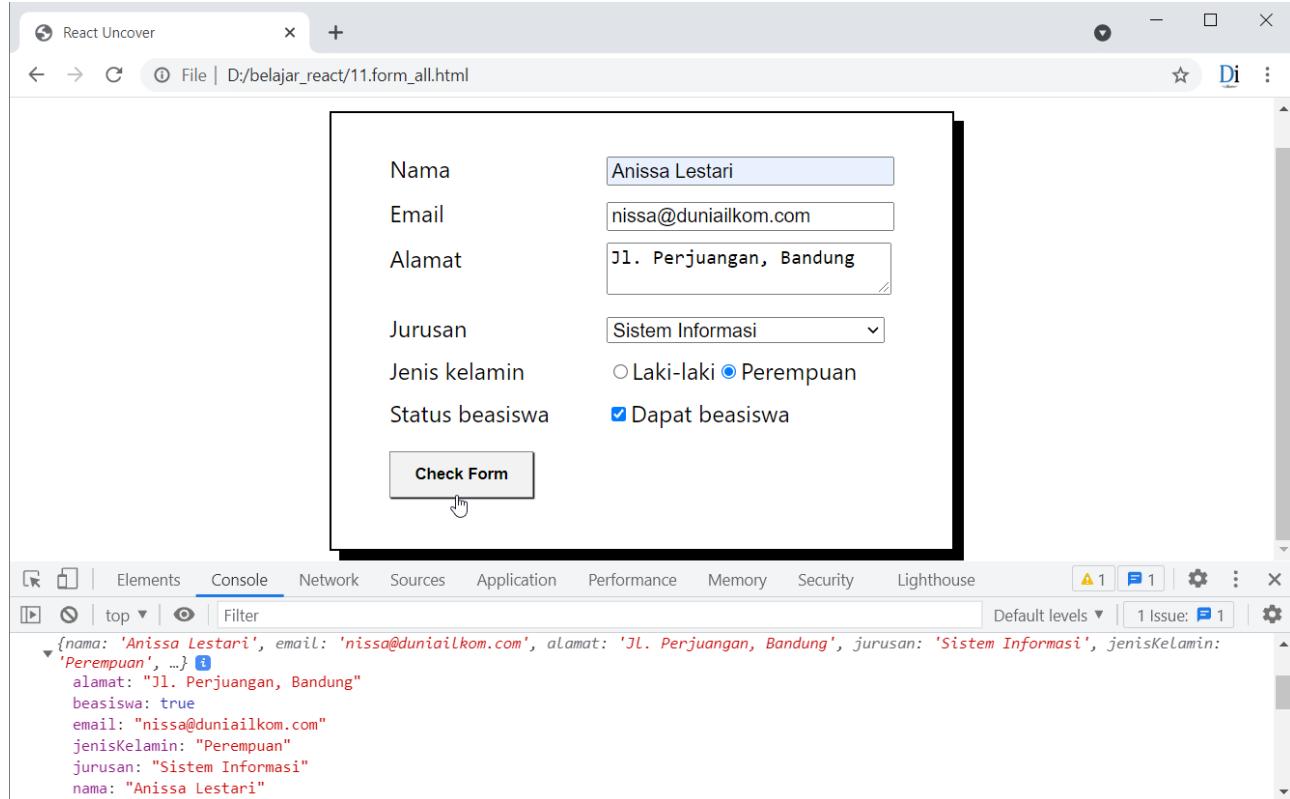
React Form Processing

```
12     font-size: 1.2rem;
13     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
14 }
15
16 .container {
17     margin-top: 2rem;
18     padding: 2rem 3rem;
19     box-shadow: 8px 8px 0px black;
20     border: 2px solid black;
21 }
22
23 .container div {
24     padding-bottom: 0.6rem;
25 }
26
27 div label:first-child {
28     display: inline-block;
29     width: 180px;
30 }
31
32 label {
33     vertical-align: top;
34 }
35
36 input[type="text"],
37 input[type="email"],
38 textarea,
39 select {
40     width: 230px;
41     font-size: 1rem;
42 }
43
44 button {
45     background-color: white;
46     cursor: pointer;
47     padding: 10px 20px;
48     font-weight: bold;
49     border: 1px solid gray;
50     box-shadow: 2px 2px 2px black;
51     margin: 10px 0;
52 }
53
54 button:hover {
55     box-shadow: 1px 1px 1px black;
56     background-color: #f1f1f1;
57 }
58 </style>
59 </head>
60
61 <body>
62     <div id="root"></div>
63
64     <script src="js/react.development.js"></script>
65     <script src="js/react-dom.development.js"></script>
66     <script src="js/babel.js"></script>
```

```
67 <script type="text/babel">
68
69     class MyApp extends React.Component {
70         constructor(props) {
71             super(props);
72             this.state = {
73                 nama: "",
74                 email: "",
75                 alamat: "",
76                 jurusan: "",
77                 jenisKelamin: "",
78                 beasiswa: false
79             };
80         }
81
82         handleInputChange = (event) => {
83             if (event.target.type === "checkbox") {
84                 this.setState({ [event.target.name]: event.target.checked })
85             }
86             else {
87                 this.setState({ [event.target.name]: event.target.value })
88             }
89         }
90
91         handleButtonClick = () => {
92             console.log(this.state);
93         }
94
95         render() {
96             return (
97                 <div className="container">
98
99                     <div>
100                         <label htmlFor="nama">Nama </label>
101                         <input type="text" name="nama" id="nama"
102                             onChange={this.handleInputChange} />
103                     </div>
104
105                     <div>
106                         <label htmlFor="email">Email </label>
107                         <input type="email" name="email" id="email"
108                             onChange={this.handleInputChange} />
109                     </div>
110
111                     <div>
112                         <label htmlFor="alamat">Alamat </label>
113                         <textarea name="alamat" id="alamat"
114                             onChange={this.handleInputChange} />
115                     </div>
116
117                     <div>
118                         <label htmlFor="jurusan">Jurusan </label>
119                         <select name="jurusan" value={this.state.jurusan}>
120                             <option value="Ilmu Komputer">Ilmu Komputer</option>
```

```
122         <option value="Sistem Informasi">Sistem Informasi</option>
123         <option value="Teknik Informatika">Teknik Informatika</option>
124         <option value="Teknik Komputer">Teknik Komputer</option>
125     </select>
126   </div>
127
128   <div>
129     <label>Jenis kelamin </label>
130     <input type="radio" name="jenisKelamin" id="jenisKelamin1"
131       value="Laki-laki"
132       checked={this.state.jenisKelamin === "Laki-laki"}
133       onChange={this.handleInputChange}>
134   />
135   <label htmlFor="jenisKelamin1">Laki-laki</label>
136   <input type="radio" name="jenisKelamin" id="jenisKelamin2"
137       value="Perempuan"
138       checked={this.state.jenisKelamin === "Perempuan"}
139       onChange={this.handleInputChange}>
140   />
141   <label htmlFor="jenisKelamin2">Perempuan</label>
142 </div>
143
144   <div>
145     <label>Status beasiswa </label>
146     <input type="checkbox" id="beasiswa" name="beasiswa"
147       onChange={this.handleInputChange} />
148     <label htmlFor="beasiswa">Dapat beasiswa</label>
149   </div>
150
151   <button onClick={this.handleButtonClick}>
152     Check Form
153   </button>
154 </div>
155 )
156 }
157 }
158
159 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
160
161 </script>
162 </body>
163
164 </html>
```

React Form Processing



Gambar: Mengakses nilai dari form utuh

Form ini terdiri dari 6 jenis inputan:

- `<input type="text">`
- `<input type="email">`
- `<textarea>`
- `<select>`
- `<input type="radio">`
- `<input type="checkbox">`

Cara mengakses semua nilai inputan sudah kita pelajari sebelumnya, yakni tempatkan event `onChange` untuk mengupdate state, serta isi atribut `value` dengan state yang sesuai.

State di definisikan dari dalam constructor pada baris 72-79, dimana setiap inputan form akan disimpan ke dalam state terpisah.

Yang menjadi perhatian kita ada di method `handleInputChange()` pada baris 82-89. Ini merupakan *event handler* yang dijalankan oleh semua inputan form untuk event `onChange`. Dengan mengakses *event object*, method ini bisa menyesuaikan diri dengan jenis inputan yang saat ini sedang di akses.

Sebagai contoh, di baris 113 terdapat tag berikut:

```
<textarea name="alamat" id="alamat" onChange={this.handleInputChange} />
```

Setiap inputan form memiliki atribut `name` yang nilainya saya buat sama persis dengan nama state untuk form tersebut.

Ketika kita mulai mengetik teks ke dalam textarea, event `onChange` akan ter-trigger dan menjalankan isi method `handleInputChange()`. Di dalamnya, perintah `event.target.name` akan diproses sebagai "alamat", sehingga kode berikut:

```
this.setState({ [event.target.name]: event.target.value })
```

Akan diproses sebagai:

```
this.setState({ alamat: event.target.value })
```

Hasilnya, nilai state alamat akan di-update.

Dengan "trik" menambah atribut `name` pada setiap inputan form, kita hanya perlu satu method `handleInputChange()` untuk semua inputan form. Nilai `name`-lah yang menjadi penentu state apa yang perlu di update.

Kondisi `if (event.target.type === "checkbox")` diperlukan karena proses update inputan checkbox sedikit berbeda. Jika dalam form tidak ada checkbox, maka isi method `handleInputChange()` bisa ditulis langsung dengan kode berikut:

```
handleInputChange = (event) => {
  this.setState({ [event.target.name]: event.target.value })
}
```

Terakhir, ketika tombol "Check Form" di klik, semua isi state akan tampil di tab Console menggunakan perintah `console.log(this.state)` di baris 92.

11.7. Validasi Form

Nilai form sudah berhasil kita akses, tahap selanjutnya adalah membuat validasi form, yakni memeriksa apakah isi inputan sudah sesuai dengan keinginan atau tidak. Proses validasi ini akan banyak menggunakan fitur JavaScript native, seperti memeriksa panjang string dari property `length`, menggunakan *regular expression* untuk pengecekan pola, dst.

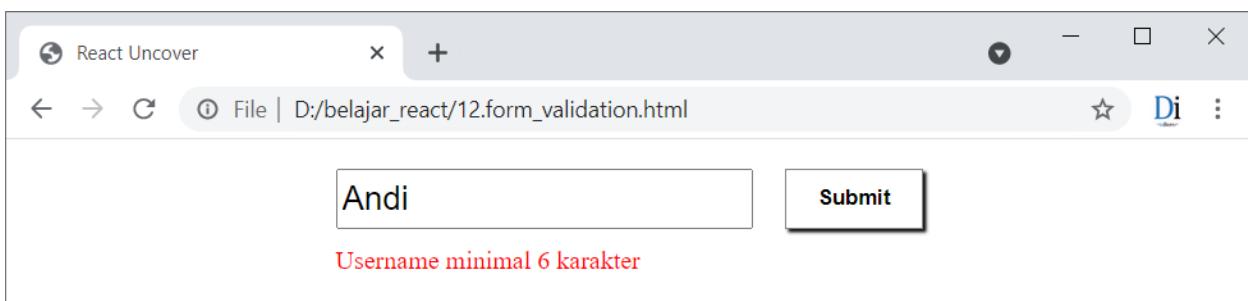
Berikut contoh proses validasi inputan form:

```
12.form_validation.html
```

```
1 class MyApp extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       username: "",
6       pesanError: "",
```

React Form Processing

```
7      };
8  }
9
10 handleUsernameChange = (event) => {
11   this.setState({ username: event.target.value })
12 }
13
14 handleFormSubmit = (event) => {
15   event.preventDefault();
16   console.log(this.state.username);
17
18   if (this.state.username.trim() === "") {
19     this.setState({ pesanError: "Username tidak boleh kosong" })
20   }
21   else if (this.state.username.length < 6) {
22     this.setState({ pesanError: "Username minimal 6 karakter" })
23   }
24   else {
25     this.setState({ pesanError: "" })
26     alert("Username valid");
27   }
28 }
29
30 render() {
31   return (
32     <div>
33       <form onSubmit={this.handleFormSubmit}>
34         <input type="text" value={this.state.username}
35           onChange={this.handleUsernameChange} />
36         <input type="submit" style={{ marginLeft: "10px" }} />
37       </form>
38       {this.state.pesanError && <p>{this.state.pesanError}</p>}
39     </div>
40   )
41 }
42 }
43
44 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Tampilan pesan error validasi

Di dalam method `render()`, kali ini saya membuat struktur form yang "sebenarnya", yakni berada di dalam tag `<form>` di baris 33-37. Seperti yang sudah kita praktekkan sejak awal bab, tag `<form>` tidak berpengaruh ke proses pengambilan nilai. Akan tetapi struktur ideal form

memang berada di dalam tag <form>.

Tag <form> memiliki atribut onSubmit={this.handleSubmit}. Di dalam JavaScript, event onSubmit akan ter-trigger pada saat form di submit. Ini bisa kita pakai sebagai titik untuk pemrosesan form, termasuk membuat validasi.

Di dalam tag <form> terdapat tag <input> dengan atribut value={this.state.username} dan onChange={this.handleUsernameChange}. Keduanya dipakai untuk proses koneksi dengan state username.

Setelah itu tag <input type="submit"> di baris 36 berfungsi sebagai tombol submit.

Di baris 37, ada perintah {this.state.pesanError && <p>{this.state.pesanError}</p>}. Meskipun kita belum membahas apa isi state pesanError, tapi bisa disimpulkan kalau kode ini dipakai untuk memeriksa apakah this.state.pesanError berisi suatu nilai atau tidak. Jika iya, tampilkan isinya di dalam tag <p>.

Masuk ke dalam constructor komponen MyApp, di baris 4-7 saya membuat 2 buah state: username dan pesanError. State username dipakai untuk menampung inputan form, sedangkan state pesanError dipakai untuk menampung pesan error.

Method handleUsernameChange() di baris 10-12 berfungsi sebagai *handler* dari event onChange. Ketika kita mengetik sesuatu, nilai state username akan langsung di update.

Fokus utama kita ada di method handleSubmit() antara baris 14-28, method ini dijalankan ketika form di-submit (event onSubmit). Parameter event dipakai untuk menampung event object.

Di baris 15, perintah event.preventDefault() berfungsi untuk mencegah form di kirim ke server. Jika perintah ini tidak ditulis, maka begitu tombol submit di tekan, halaman akan refresh dan mencari file di web server untuk pemrosesan form. Efek ini harus di blokir karena form akan kita proses di sisi client saja.

Di baris 16, perintah console.log(this.state.username) sekedar menampilkan apa isi state username pada saat proses submit. Ini bisa terlihat di dalam tab Console.

Proses validasi ada di baris 18-27. Kondisi if (this.state.username.trim() === "") akan bernilai true jika form inputan tidak diisi karakter apapun. Jika ini terjadi, isi state pesanError dengan string "Username tidak boleh kosong".

Validasi kedua adalah else if (this.state.username.length < 6). Artinya saya ingin inputan username berisi minimal 6 karakter. Jika kurang, maka isi state pesanError dengan string "Username minimal 6 karakter". Pemeriksaan kondisi ini diawali dengan keyword "else if", sehingga akan dijalankan hanya jika lolos dari validasi pertama.

Apabila kedua syarat validasi ini terpenuhi, maka akan sampai ke blok else di baris 25-26. Isinya berupa perintah untuk mengosongkan state pesanError dan menampilkan jendela

```
alert("Username valid").
```

Silahkan buka halaman di atas, lalu tes apakah 2 syarat validasi ini bisa berjalan atau tidak.

Selanjutnya, saya akan tambah satu inputan email. Berikut kode programnya:

13.form_validation_with_email.html

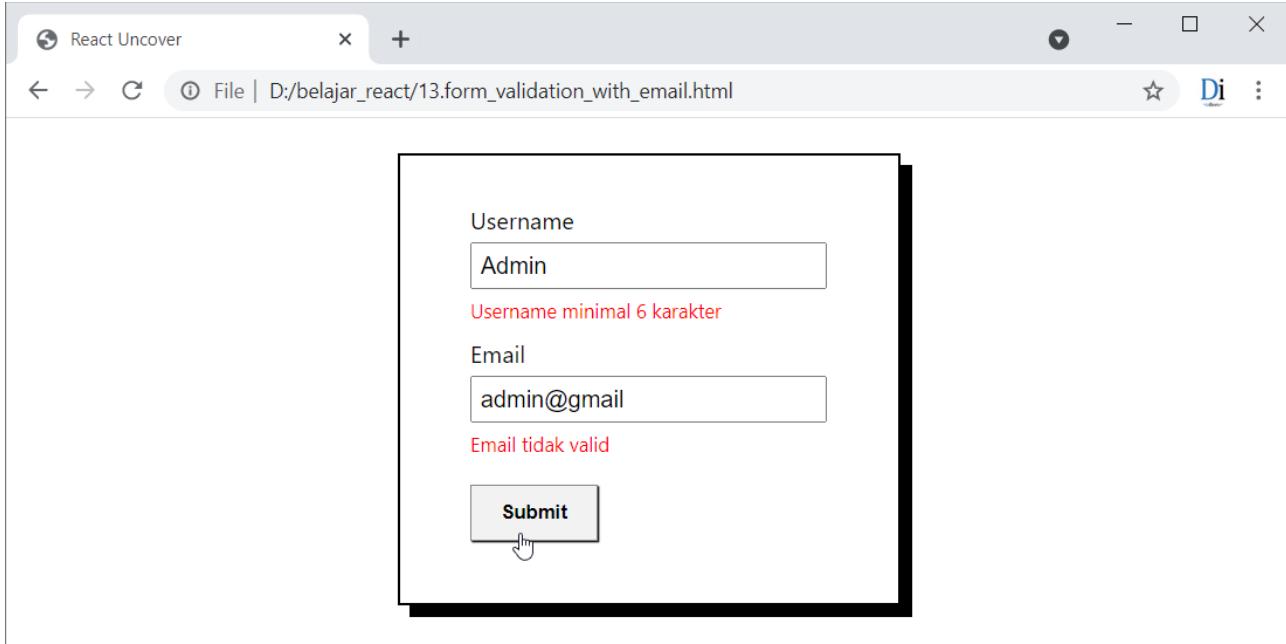
```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = {
5              username: "",
6              email: "",
7              errors: { username: "", email: "" },
8          };
9      }
10
11     handleInputChange = (event) => {
12         this.setState({ [event.target.name]: event.target.value })
13     }
14
15     handleSubmit = (event) => {
16         event.preventDefault();
17
18         const username = this.state.username;
19         const email = this.state.email;
20         let errors = this.state.errors;
21
22         // validasi username
23         if (!username) {
24             errors.username = "Username tidak boleh kosong";
25         }
26         else if (username.length < 6) {
27             errors.username = "Username minimal 6 karakter";
28         }
29         else {
30             errors.username = "";
31         }
32
33         // validasi email
34         if (!email) {
35             errors.email = "Email tidak boleh kosong";
36         }
37         else if ((!/\S+@\S+\.\S+/.test(email))) {
38             errors.email = "Email tidak valid";
39         }
40         else {
41             errors.email = "";
42         }
43
44         // update error state
45         this.setState({ errors: errors });
46
47         // cek apakah seluruh form valid atau masih ada error

```

React Form Processing

```
48     let formValid = true;
49     for (let propName in errors) {
50         if (errors[propName].length > 0) {
51             formValid = false;
52         }
53     }
54
55     // proses data jika form valid
56     if (formValid) {
57         console.log(formValid)
58         alert("Username dan Email valid")
59     }
60
61 }
62
63 render() {
64     return (
65         <div className="container">
66             <form onSubmit={this.handleSubmit} noValidate>
67
68                 <div>
69                     <label htmlFor="username">Username </label>
70                     <input
71                         type="text" id="username" name="username"
72                         value={this.state.username}
73                         onChange={this.handleInputChange} />
74                     {this.state.errors.username &&
75                         <small>{this.state.errors.username}</small>}
76                 </div>
77
78                 <div>
79                     <label htmlFor="email">Email </label>
80                     <input
81                         type="email" id="email" name="email"
82                         value={this.state.email}
83                         onChange={this.handleInputChange} />
84                     {this.state.errors.email &&
85                         <small>{this.state.errors.email}</small>}
86                 </div>
87
88                 <input type="submit" style={{ marginTop: "10px" }} />
89             </form>
90         </div>
91     )
92 }
93 }
94
95 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Validasi username dan email

Tambahan satu inputan ini membuat kode program menjadi semakin panjang, karena kita juga harus membuat validasi untuk inputan tersebut.

Method `render()` berisi 2 inputan form, yakni `username` antara baris 68-76, dan `email` di baris 78-86. Keduanya berisi struktur yang sama, yakni memiliki atribut `onChange` untuk proses update ke state `username` dan state `email`.

Di bagian akhir setiap inputan terdapat perintah untuk menampilkan pesan error, misalnya `{this.state.errors.username && <small>{this.state.errors.username}</small>}` di baris 74-75.

Ke dalam tag `<form>` terdapat tambahan atribut `onSubmit={this.handleSubmit}` untuk pemrosesan form, serta atribut `noValidate` untuk menonaktifkan validasi bawaan HTML5. Sebagaimana yang kita ketahui, HTML5 memiliki validasi bawaan untuk beberapa tipe inputan seperti email. Agar tidak menghalangi proses validasi yang kita tulis secara manual, validasi bawaan ini untuk sementara di nonaktifkan saja.

Proses deklarasi state ada di baris 4-8. Perhatikan struktur state ini, saya membuat 3 state: state `username` dan `email` untuk menampung inputan form, serta 1 state `errors` yang berisi object `{ username: "", email: "" }`. Struktur seperti ini sekedar untuk memudahkan logika pembuatan pesan error yang akan saya buat.

Masuk ke method `handleSubmit()` di baris 15, pada awal method terdapat perintah `event.preventDefault()` untuk mencegah form di submit ke server.

Kemudian saya mencopy ketiga state ke dalam konstanta dan variabel `username`, `email` dan `errors` di baris 18-20. Ini semata-mata untuk mempersingkat penulisan state.

Proses validasi `username` ada di baris 23-31. Isinya kurang lebih mirip seperti contoh sebelumnya. Validasi pertama adalah `if (!username)`, kondisi ini akan terpenuhi jika `username` tidak berisi karakter apapun. Ini merupakan penulisan alternatif dari `if (username.trim() === "")`.

Meskipun sedikit lebih panjang, saya pribadi cenderung lebih menyukai penulisan kedua yang menggunakan method `trim()`. Ini lebih mudah dipahami dibandingkan `if (!username)`.

Jika validasi `if (!username)` tidak lolos, maka isi property `username` dari object `errors` dengan string "Username tidak boleh kosong". Berbeda dengan sebelumnya, kali ini saya tidak langsung mengupdate state, tapi ditampung dahulu ke dalam variabel `errors`.

Jika validasi pertama berhasil lolos, masuk ke validasi kedua untuk memeriksa apakah panjang `username` kurang dari 6 karakter. Jika `if (username.length < 6)` juga tidak lolos, isi kembali property `username` ke dalam object `errors` dengan string "Username minimal 6 karakter".

Jika kedua validasi dilewati, isi string kosong ke dalam `errors.username`.

Proses validasi inputan `email` ada di baris 34-42. Sama seperti validasi `username`, kondisi `if (!email)` dipakai untuk memeriksa apakah inputan masih kosong atau tidak. Jika masih kosong, isi string "Email tidak boleh kosong" ke dalam `errors.email`.

Validasi `email` kedua berisi pola *regular expression* `if ((!/\S+@\S+\.\S+/.test(email))` untuk memeriksa apakah `email` memiliki karakter '@', serta nama domain. Ini merupakan pola sederhana untuk pemeriksaan `email`. Jika tidak lolos, isi string "Email tidak valid" ke dalam variabel `errors.email`.

Barulah jika kedua syarat validasi ini dipenuhi, isi string kosong ke dalam `errors.email`.

Itulah beberapa syarat validasi yang saya tulis. Jika anda ingin membuat aturan tambahan, bisa menggunakan pola yang sama, tinggal menambah kondisi `else if` saja dan mengisi pesan error ke dalam variabel `erro`s.

Dalam setiap validasi, pesan error itu baru ditampung ke dalam variabel `errors`. Perintah `this.setState({ errors: errors })` di baris 45 akan mengcopy isi variabel `errors` ke dalam state `this.state.errors`.

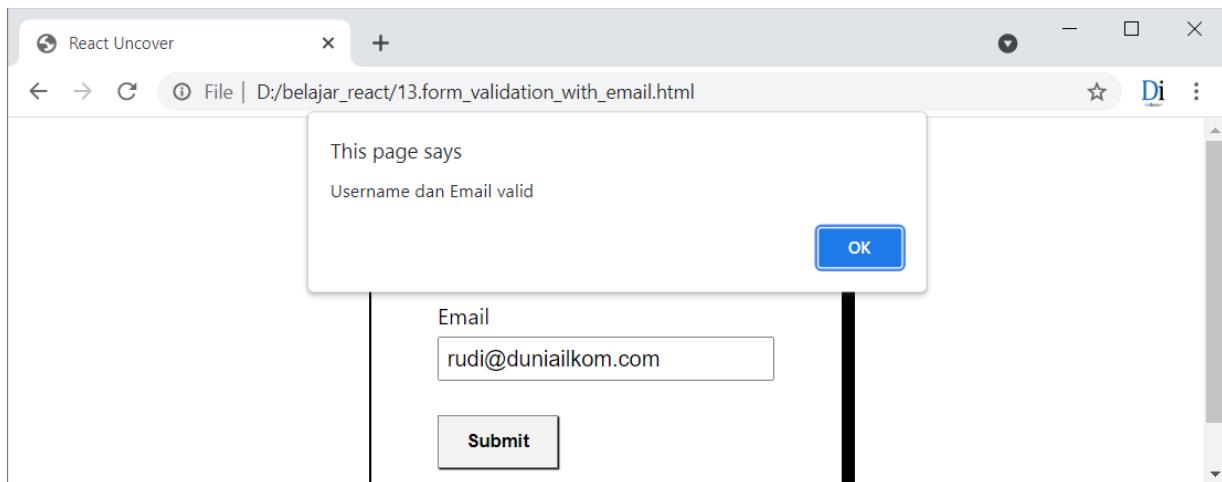
Perintah di baris 48-53 berfungsi untuk mencari tahu apakah variabel `errors` masih berisi suatu string atau tidak. Ini dilakukan dengan membuat perulangan `for in` untuk mengakses setiap property yang ada di dalam variabel `errors`, lalu periksa apakah berisi string yang panjangnya > 0 .

Jika masih berisi suatu string, artinya form masih tidak lolos validasi dan set variabel `formValid` sebagai `false`.

Variabel `formValid` inilah yang menjadi kunci untuk memastikan seluruh inputan form sudah lolos validasi. Jika isinya bernilai `true`, maka validasi sukses dan tampilkan pesan

```
alert("Username dan Email valid").
```

Dalam praktek aslinya nanti, pesan alert ini bisa diganti dengan perintah AJAX atau fetch API yang akan mengirim data ke server untuk selanjutnya disimpan ke dalam database.



Gambar: Inputan form sudah lolos validasi

Itulah cara membuat validasi form di React. Di beberapa studi kasus pada bagian akhir buku nanti, kita akan banyak pakai konsep yang sama.

Setiap programmer memiliki teknik dan solusi sendiri untuk membuat validasi form serta menampilkan pesan error. Anda tidak harus mengikuti cara yang saya pakai, misalnya jika merasa penggunaan nested state agak susah (menyimpan state sebagai object), bisa menggunakan alternatif cara lain.

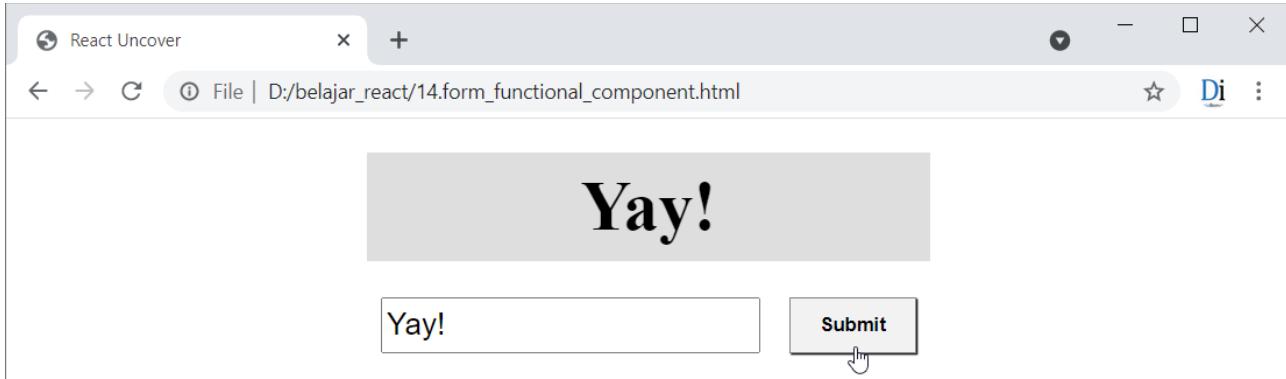
11.8. Form Processing di Functional Component

Untuk versi functional component, teknik yang dipakai kurang lebih sama. Form inputan tetap terhubung ke state via onChange event. Berikut contoh penulisannya:

```
1 const MyApp = () => {
2   const [username, setUsername] = React.useState("");
3
4   const handleUsernameChange = (event) => {
5     setUsername(event.target.value)
6   }
7
8   const handleFormSubmit = (event) => {
9     event.preventDefault();
10    console.log(username);
11  }
12
13  return (
14    <div>
15      <h1>{username}</h1>
```

React Form Processing

```
16     <form onSubmit={handleFormSubmit}>
17         <input type="text" value={username} onChange={handleUsernameChange} />
18         <input type="submit" />
19     </form>
20   </div>
21 )
22 }
23
24 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Tampilan form dengan functional component

Di dalam perintah return JSX, komponen `MyApp` memiliki tag `<h1>{username}</h1>` untuk menampilkan state `username`, serta tag `<form>` yang terdiri dari satu inputan text box serta tombol submit.

Proses koneksi antara tag `<input type="text">` dengan state `username` dilakukan oleh atribut `value={username}` dan atribut `onChange={handleUsernameChange}` di baris 17.

Pada saat form di submit, fungsi `handleFormSubmit()` akan memprosesnya sebagaimana isi atribut `onSubmit` di baris 16.

Di baris 2 saya membuat state `username` dengan nilai awal string kosong dan mendefinisikan function `setUsername` untuk proses update state.

Fungsi `handleUsernameChange()` di baris 4-6 akan dijalankan untuk event `onChange`. Isinya, update nilai state `username` dengan perintah `setUsername(event.target.value)`.

Sedangkan fungsi `handleFormSubmit()` di baris 8-11 akan dijalankan untuk event `onSubmit`. Isinya, batalkan pengiriman nilai form dengan perintah `event.preventDefault()`, serta tampilkan nilai state dengan perintah `console.log(username)`.

Tidak ada perubahan besar di versi *functional component*, hanya butuh penyesuaian kode saja. Sekarang mari lihat kode program untuk validasi form di versi *function component*:

15.form_functional_validation_with_email.html

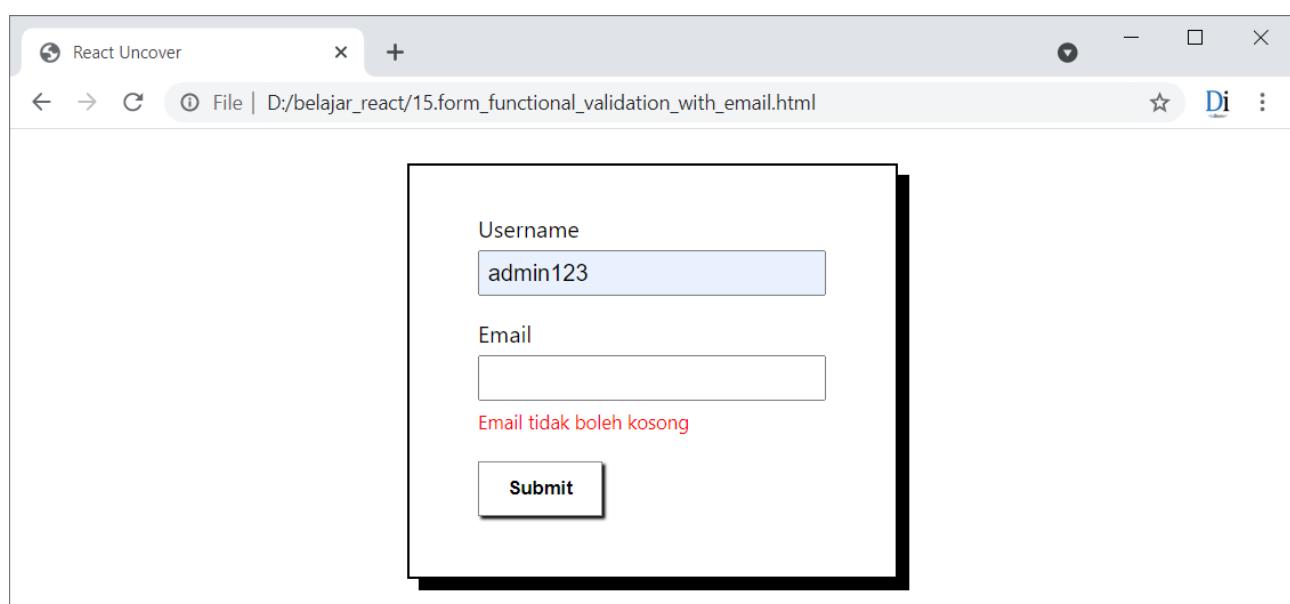
```
1 const MyApp = () => {
2   const [username, setUsername] = React.useState("");
3   const [email, setEmail] = React.useState("");
```

React Form Processing

```
4  const [errors, setErrors] = React.useState({ username: "", email: "" });
5
6  const handleUsernameChange = (event) => {
7    setUsername(event.target.value)
8  }
9
10 const handleEmailChange = (event) => {
11   setEmail(event.target.value)
12 }
13
14 const handleSubmit = (event) => {
15   event.preventDefault();
16   let pesanErrors = {};
17
18   // validasi username
19   if (!username.trim()) {
20     pesanErrors.username = "Username tidak boleh kosong";
21   }
22   else if (username.length < 6) {
23     pesanErrors.username = "Username minimal 6 karakter";
24   }
25   else {
26     pesanErrors.username = "";
27   }
28
29   // validasi email
30   if (!email) {
31     pesanErrors.email = "Email tidak boleh kosong";
32   }
33   else if ((!/\S+@\S+\.\S+/.test(email))) {
34     pesanErrors.email = "Email tidak valid";
35   }
36   else {
37     pesanErrors.email = "";
38   }
39
40   // update error state
41   setErrors(pesanErrors);
42
43   // cek apakah seluruh form valid atau masih ada error
44   let formValid = true;
45   for (let propName in pesanErrors) {
46     if (pesanErrors[propName].length > 0) {
47       formValid = false;
48     }
49   }
50
51   // proses data jika form valid
52   if (formValid) {
53     console.log(formValid)
54     alert("Username dan Email valid")
55   }
56 }
57
58 return (
```

React Form Processing

```
59 <div className="container">
60   <form onSubmit={handleSubmit} noValidate>
61
62     <div>
63       <label htmlFor="username">Username </label>
64       <input
65         type="text" id="username" name="username"
66         value={username}
67         onChange={handleUsernameChange} />
68       {errors.username && <small>{errors.username}</small>}
69     </div>
70
71     <div>
72       <label htmlFor="email">Email </label>
73       <input
74         type="email" id="email" name="email"
75         value={email}
76         onChange={handleEmailChange} />
77       {errors.email && <small>{errors.email}</small>}
78     </div>
79
80     <input type="submit" style={{ marginTop: "10px" }} />
81   </form>
82 </div>
83 )
84 }
85 }
86
87 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Validasi form dengan functional component

Secara konsep, kode ini sama seperti praktik kita di versi *class component*. Yang berubah hanya di perintah untuk pengelolaan state saja. Sekarang state `username`, `email` dan `errors` dibuat dengan perintah terpisah antara baris 2-4. Nilai awal untuk state `errors` juga sudah

dalam bentuk object.

Sisa kode lain mirip seperti penjelasan di class component sehingga tidak akan saya bahas lagi. Bisa terlihat versi *functional component* ini lebih ringkas karena tidak perlu menggunakan "this" untuk merujuk ke state.

11.9. Validasi Secara Realtime

Salah satu keunggulan membuat validasi form di sisi *client* (via JavaScript) adalah kita bisa menjalankan validasi secara *realtime*. Pesan error akan tampil pada saat yang bersamaan ketika user mengisi form tersebut. Dan karena state sudah di update secara *realtime*, tidak sulit membuat fitur ini di React:

16.form_functional_validation_realtime.html

```

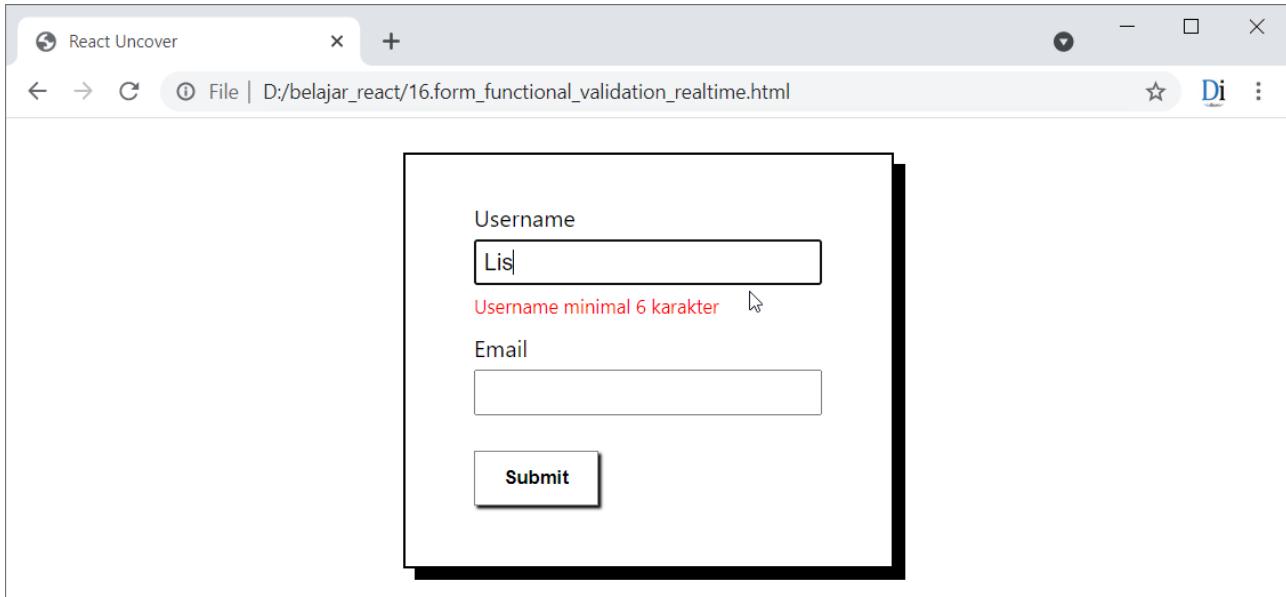
1  const MyApp = () => {
2      const [username, setUsername] = React.useState("");
3      const [email, setEmail] = React.useState("");
4      const [errors, setErrors] = React.useState({ username: "", email: "" });
5
6      const handleUsernameChange = (event) => {
7          setUsername(event.target.value);
8
9          let pesanError;
10         // validasi username
11         if (!event.target.value.trim()) {
12             pesanError = "Username tidak boleh kosong";
13         }
14         else if (event.target.value.length < 6) {
15             pesanError = "Username minimal 6 karakter";
16         }
17         else {
18             pesanError = "";
19         }
20
21         // update error state
22         setErrors((prevState) => ({ ...prevState, username: pesanError }));
23     }
24
25     const handleEmailChange = (event) => {
26         setEmail(event.target.value)
27
28         let pesanError;
29         // validasi email
30         if (!email) {
31             pesanError = "Email tidak boleh kosong";
32         }
33         else if ((!/\S+@\S+\.\S+/.test(email))) {
34             pesanError = "Email tidak valid";
35         }
36         else {

```

React Form Processing

```
37     pesanError = "";
38 }
39
40 // update error state
41 setErrors((prevState) => ({ ...prevState, email: pesanError }));
42 }
43
44 const handleSubmit = (event) => {
45   event.preventDefault();
46
47   // cek apakah seluruh form valid atau masih ada error
48   let formValid = true;
49   for (let propName in errors) {
50     if (errors[propName].length > 0) {
51       formValid = false;
52     }
53   }
54
55   // tambahan validasi jika user langsung submit di awal
56   if ((!username) && (!email)){
57     setErrors({ email: "Email tidak boleh kosong",
58     username: "Username tidak boleh kosong" });
59     formValid = false;
60   }
61
62   // proses data jika form valid
63   if (formValid) {
64     console.log(formValid)
65     alert("Username dan Email valid")
66   }
67 }
68
69 return (
70   <div className="container">
71     <form onSubmit={handleSubmit} noValidate>
72
73       <div>
74         <label htmlFor="username">Username </label>
75         <input
76           type="text" id="username" name="username"
77           value={username}
78           onChange={handleUsernameChange} />
79         <small>{errors.username}</small>
80       </div>
81
82       <div>
83         <label htmlFor="email">Email </label>
84         <input
85           type="email" id="email" name="email"
86           value={email}
87           onChange={handleEmailChange} />
88         {errors.email && <small>{errors.email}</small>}
89
90     </div>
91   
```

```
92         <input type="submit" style={{ marginTop: "10px" }} />
93     </form>
94   </div>
95 )
96 }
```



Gambar: Proses validasi berjalan secara realtime

Sebelumnya, method **event handler onChange** dipakai hanya untuk mengupdate nilai state saja, sekarang ditambah dengan proses validasi.

Sebagai contoh, di baris 78 terdapat atribut `onChange={handleUsernameChange}`. Maka fungsi `handleUsernameChange` akan dijalankan setiap kali karakter di ketik / dihapus.

Pendefinisian fungsi `handleUsernameChange()` ada di baris 6-23. Di awal, perintah `setUsername(event.target.value)` akan mengupdate nilai state saat ini, kemudian saya membuat variabel `pesanError` untuk menampung pesan error di setiap syarat validasi.

Misal, jika inputan `username` masih kosong, isi `pesanError` dengan string "Username tidak boleh kosong" (baris 12), atau jika inputan `username` kurang dari 6 angka, isi `pesanError` dengan string "Username minimal 6 karakter" (baris 16). Namun jika ternyata 2 syarat validasi di atas sudah dipenuhi, isi string kosong ke dalam variabel `pesanError` (baris 18).

Variabel `pesanError` kemudian diinput ke dalam state `errors` dengan perintah berikut:

```
setErrors((prevState) => ({ ...prevState, username: pesanError }));
```

Proses update state `errors` perlu ditulis seperti ini karena `errors` berbentuk *nested object*, bukan nilai state biasa. Perintah `...prevState` akan "menyebarluaskan" isi state sebelumnya, lalu nilai `errors.username` akan ditimpas oleh `username: pesanError`. Penjelasan lebih detail tentang teknik ini sudah pernah kita bahas pada bab **useState hook**.

Hasilnya, setiap kali karakter di ketik ke inputan `username`, pesan error akan selalu update dan

tampil dibagian bawah inputan.

Cara yang sama juga saya buat untuk inputan email, yang event handlernya ada di baris 25-42.

Ketika form di submit, kita tinggal cek apakah state `errors` masih berisi suatu teks atau tidak. Jika masih ada, artinya masih ada validasi yang tidak lolos. Proses pemeriksaan ini ada di baris 48-53 yang sama seperti contoh-contoh sebelumnya.

Di baris 56-60 terdapat tambahan proses validasi yang secara khusus ditujukan jika user langsung men-klik tombol submit tanpa sempat mengisi form. Jika tanpa tambahan pemeriksaan ini, proses validasi form bisa dilewati karena event `change` untuk inputan `username` dan `email` tidak ter-trigger.

Kita hanya perlu memeriksa 1 kondisi khusus, yakni jika state `username` dan `email` masih kosong. Jika ini terpenuhi, isi state `errors` dan update variabel `formValid` menjadi `false`.

Barulah jika state `errors` sudah kosong yang berarti semua syarat validasi sudah dipenuhi, tampilkan pesan `alert("Username dan Email valid")` tampil di layar.

Menampilkan pesan error secara realtime memang sangat menarik, tapi pertimbangkan juga sisi performa dari aplikasi kita. Sebab, proses validasi perlu berjalan setiap saat.

Untuk form yang kompleks dan terdiri dari banyak inputan, pertimbangkan juga agar proses validasi dilakukan sekali di akhir saja (pada saat event `onSubmit`).

Itulah materi kita tentang pemrosesan form di React. Menghubungkan inputan form dengan state membuat pengelolaan data form menjadi lebih mudah. Namun kelemahannya, proses update nilai state akan merender ulang seluruh komponen.

Sebagai alternatif, React juga mengizinkan kita mengambil nilai form tanpa harus menyimpannya ke dalam state. Inilah yang menjadi bahasan pada bab selanjutnya.

12. Ref DOM Manipulation

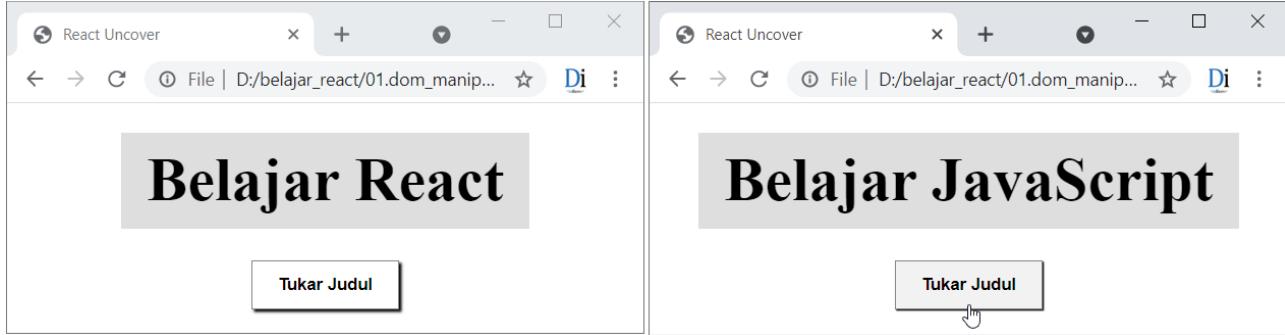
Dalam beberapa bab sebelum ini, kita sempat membahas **ref** sebagai pengganti variabel di *functional component*. Namun sebenarnya ref juga bisa dipakai untuk hal lain seperti DOM manipulation, termasuk mengambil nilai inputan form.

12.1. Mengakses DOM dengan Ref

Pada saat membuat aplikasi dengan React, state dan *props* menjadi fokus utama kita. Sedapat mungkin manfaatkan kedua fitur ini untuk memproses data. Berikut salah satu contoh yang dimaksud:

01.dom_manipulation_state.html

```
1  class MyApp extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { judul: "Belajar React" };
5    }
6
7    handleButtonClick = () => {
8      this.setState({ judul: "Belajar JavaScript" });
9    }
10
11   render() {
12     return (
13       <div>
14         <h1>{this.state.judul}</h1>
15         <button onClick={this.handleButtonClick}>Tukar Judul</button>
16       </div>
17     )
18   }
19 }
20
21 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengganti judul dengan state

Kode ini sudah berulang kali kita pakai. Pada saat tombol "Tukar Judul" di klik, isi teks tag `<h1>` akan berubah dari "Belajar React" menjadi "Belajar JavaScript". Teks tersebut berasal dari `this.state.judul`.

Namun bisa jadi ada situasi yang mengharuskan kita mengakses DOM secara langsung. Dari JavaScript native, ini bisa dilakukan dengan perintah `getElementById()` seperti contoh berikut:

02.dom_manipulation_js.html

```
1  class MyApp extends React.Component {  
2      handleButtonClick = () => {  
3          document.getElementById("judul").innerHTML = "Belajar JavaScript";  
4      }  
5  
6      render() {  
7          return (  
8              <div>  
9                  <h1 id="judul">Belajar React</h1>  
10                 <button onClick={this.handleButtonClick}>Tukar Judul</button>  
11             </div>  
12         )  
13     }  
14 }  
15  
16 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Perhatikan perintah di baris 3, di situ saya mengakses DOM tag `<h1>` dengan perintah `document.getElementById("judul")`, lalu mengganti isi property `innerHTML` menjadi string "Belajar JavaScript". Di sini kita mengakses node object HTML secara langsung, bukan lewat state.

React menyediakan `ref` sebagai alternatif perintah `document.getElementById()`. Berikut contoh prakteknya:

03.dom_manipulation_ref.html

```
1  class MyApp extends React.Component {  
2  
3      constructor(props) {
```

```

4     super(props);
5     this.judulRef = React.createRef();
6   }
7
8   handleButtonClick = () => {
9     this.judulRef.current.innerHTML = "Belajar JavaScript";
10  }
11
12  render() {
13    return (
14      <div>
15        <h1 ref={this.judulRef}>Belajar React</h1>
16        <button onClick={this.handleButtonClick}>Tukar Judul</button>
17      </div>
18    )
19  }
20}
21
22 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Di dalam constructor pada baris 5, saya membuat **ref** dengan perintah `React.createRef()` dan menyimpannya ke dalam property `this.judulRef`. Property ini kemudian menjadi nilai untuk atribut `ref` di baris 15.

Pada saat tombol "Tukar Judul" di klik, isi teks dari tag `<h1>` bisa diakses dari `this.judulRef.current.innerHTML` seperti di baris 9. Dari praktik ini terlihat kalau `ref` bisa dipakai sebagai "penanda" untuk mempermudah pengaksesan node object HTML.

12.2. Mengakses Nilai Form dengan ref

Karena bisa dipakai untuk mengakses DOM, `ref` juga bisa kita manfaatkan untuk mengakses nilai form. Berikut contohnya:

04.form_ref.html

```

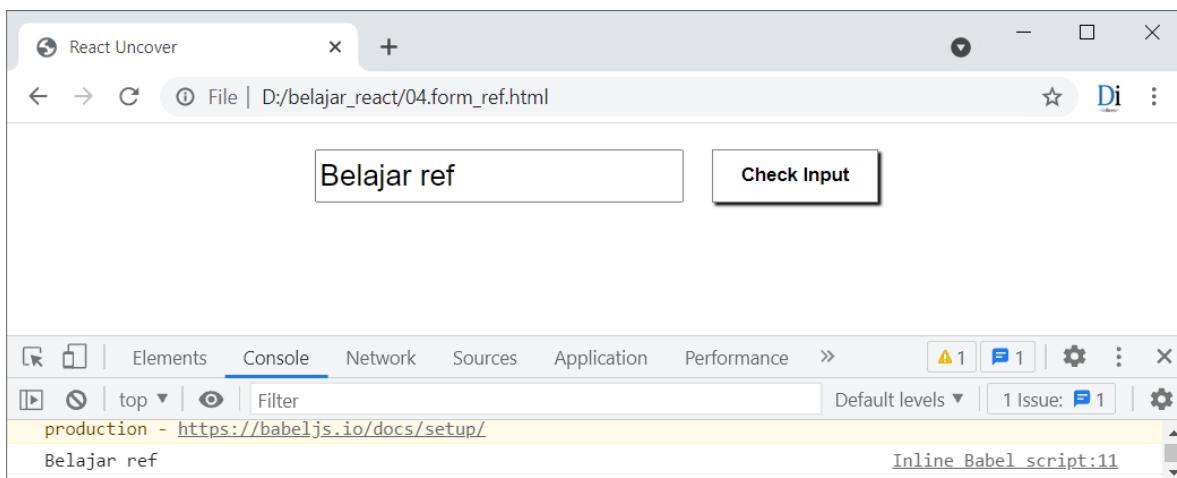
1  class MyApp extends React.Component {
2
3    constructor(props) {
4      super(props);
5      this.inputRef = React.createRef();
6    }
7
8    handleButtonClick = () => {
9      console.log(this.inputRef.current.value);
10   }
11
12  render() {
13    return (
14      <div>
15        <input type="text" ref={this.inputRef} />
16        <button onClick={this.handleButtonClick}>Check Input</button>

```

```

17      </div>
18    )
19  }
20 }
21
22 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses nilai form dengan ref

Mirip seperti sebelumnya, saya membuat `ref this.inputRef` di baris 5, lalu menghubungkan `ref` ini dengan tag `<input>` di baris 15 melalui atribut `ref={this.inputRef}`.

Hasilnya, kita bisa mengakses node object tag `<input>` dengan perintah `inputRef.current`. Di dalam struktur DOM, nilai form ada di dalam property `value`, sehingga isi inputan teks bisa diakses dari perintah `this.inputRef.current.value` seperti di baris 9.

Uncontrolled Components

Cara mengakses nilai form menggunakan `ref` seperti ini dikenal dalam React sebagai **uncontrolled components**. Disebut seperti itu karena React tidak memiliki kontrol atas nilai inputan.

Jika nilai inputan form disimpan ke dalam state, maka itu disebut sebagai **controller components** karena nilainya ada di sisi React (di kontrol langsung oleh React)

Sebagai pembanding, berikut cara pengaksesan 6 nilai form seperti praktik kita di bab sebelumnya, tapi kali ini diambil menggunakan `ref`:

05.form_all_ref.html

```

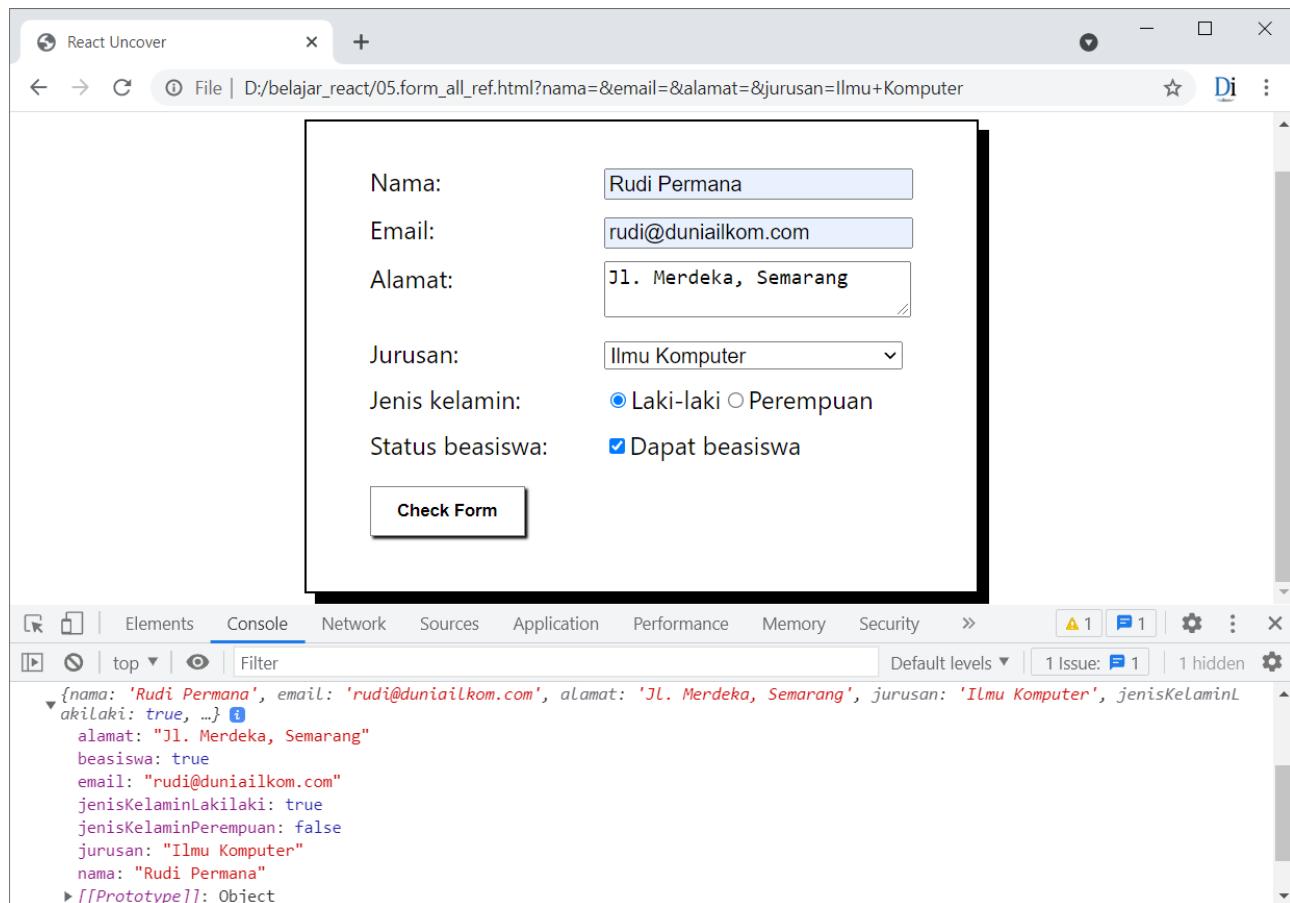
1 class MyApp extends React.Component {
2   constructor(props) {
3     super(props);
4     this.namaRef = React.createRef();
5     this.emailRef = React.createRef();
6     this.alamatRef = React.createRef();
7     this.jurusanRef = React.createRef();

```

```
8      this.jenisKelaminLakilakiRef = React.createRef();
9      this.jenisKelaminPerempuanRef = React.createRef();
10     this.beasiswaRef = React.createRef();
11   }
12
13   handleFormSubmit = (event) => {
14     event.preventDefault();
15     let hasil = {
16       nama: this.namaRef.current.value,
17       email: this.emailRef.current.value,
18       alamat: this.alamatRef.current.value,
19       jurusan: this.jurusanRef.current.value,
20       jenisKelaminLakilaki: this.jenisKelaminLakilakiRef.current.checked,
21       jenisKelaminPerempuan: this.jenisKelaminPerempuanRef.current.checked,
22       beasiswa: this.beasiswaRef.current.checked,
23     }
24
25     console.log(hasil);
26   }
27
28   render() {
29     return (
30       <div className="container">
31         <form onSubmit={this.handleFormSubmit} noValidate>
32
33           <div>
34             <label htmlFor="nama">Nama: </label>
35             <input type="text" name="nama" id="nama" ref={this.namaRef} />
36           </div>
37
38           <div>
39             <label htmlFor="email">Email: </label>
40             <input type="email" name="email" id="email" ref={this.emailRef} />
41           </div>
42
43           <div>
44             <label htmlFor="alamat">Alamat: </label>
45             <textarea name="alamat" id="alamat" ref={this.alamatRef} />
46           </div>
47
48           <div>
49             <label htmlFor="jurusan">Jurusan: </label>
50             <select name="jurusan" ref={this.jurusanRef}>
51               <option value="Ilmu Komputer">Ilmu Komputer</option>
52               <option value="Sistem Informasi">Sistem Informasi</option>
53               <option value="Teknik Informatika">Teknik Informatika</option>
54               <option value="Teknik Komputer">Teknik Komputer</option>
55             </select>
56           </div>
57
58           <div>
59             <label>Jenis kelamin: </label>
60             <input type="radio" name="jenisKelamin" id="jenisKelamin1"
61               value="Laki-laki" ref={this.jenisKelaminLakilakiRef} />
62             <label htmlFor="jenisKelamin1">Laki-laki</label>
63           </div>
64
65           <div>
66             <label>Beasiswa: </label>
67             <input type="checkbox" name="beasiswa" id="beasiswa1"
68               ref={this.beasiswaRef} />
69             <label htmlFor="beasiswa1">Beasiswa</label>
70           </div>
71
72         </form>
73       </div>
74     )
75   }
76
77   componentDidMount() {
78     this.setState({ nama: "Andrea", email: "andrea@email.com", alamat: "Jl. Pahlawan", jurusan: "Ilmu Komputer", jenisKelamin: "Laki-laki", beasiswa: false })
79   }
80
81   componentDidUpdate(prevProps, prevState) {
82     if (prevState.nama !== this.state.nama) {
83       console.log(`Nama berubah menjadi ${this.state.nama}`);
84     }
85     if (prevState.email !== this.state.email) {
86       console.log(`Email berubah menjadi ${this.state.email}`);
87     }
88     if (prevState.alamat !== this.state.alamat) {
89       console.log(`Alamat berubah menjadi ${this.state.alamat}`);
90     }
91     if (prevState.jurusan !== this.state.jurusan) {
92       console.log(`Jurusan berubah menjadi ${this.state.jurusan}`);
93     }
94     if (prevState.jenisKelamin !== this.state.jenisKelamin) {
95       console.log(`Jenis Kelamin berubah menjadi ${this.state.jenisKelamin}`);
96     }
97     if (prevState.beasiswa !== this.state.beasiswa) {
98       console.log(`Beasiswa berubah menjadi ${this.state.beasiswa}`);
99     }
100   }
101
102   render() {
103     return (
104       <div>
105         <h1>Formulir Pendaftaran</h1>
106         <hr/>
107         <div>
108           <div>
109             <label>Nama:</label>
110             <input type="text" name="nama" value={this.state.nama} ref={this.namaRef} />
111           </div>
112           <div>
113             <label>Email:</label>
114             <input type="email" name="email" value={this.state.email} ref={this.emailRef} />
115           </div>
116           <div>
117             <label>Alamat:</label>
118             <input type="text" name="alamat" value={this.state.alamat} ref={this.alamatRef} />
119           </div>
120           <div>
121             <label>Jurusan:</label>
122             <select name="jurusan" value={this.state.jurusan} ref={this.jurusanRef}>
123               <option value="Ilmu Komputer">Ilmu Komputer</option>
124               <option value="Sistem Informasi">Sistem Informasi</option>
125               <option value="Teknik Informatika">Teknik Informatika</option>
126               <option value="Teknik Komputer">Teknik Komputer</option>
127             </select>
128           </div>
129           <div>
130             <label>Jenis kelamin:</label>
131             <input checked="" type="radio" name="jenisKelamin" value="Laki-laki" ref={this.jenisKelaminLakilakiRef} />
132             <label>Laki-laki</label>
133           </div>
134           <div>
135             <label>Beasiswa:</label>
136             <input checked="" type="checkbox" name="beasiswa" ref={this.beasiswaRef} />
137             <label>Beasiswa</label>
138           </div>
139         </div>
140         <hr/>
141         <button type="button" onClick={this.handleSubmit}>Submit</button>
142       </div>
143     )
144   }
145 }
```

Ref DOM Manipulation

```
63     <input type="radio" name="jenisKelamin" id="jenisKelamin2"
64         value="Perempuan" ref={this.jenisKelaminPerempuanRef} />
65     <label htmlFor="jenisKelamin2">Perempuan</label>
66 </div>
67
68 <div>
69     <label>Status beasiswa: </label>
70     <input type="checkbox" id="beasiswa" name="beasiswa"
71         ref={this.beasiswaRef} />
72     <label htmlFor="beasiswa">Dapat beasiswa</label>
73 </div>
74
75     <button type="submit"> Check Form </button>
76
77 </form>
78 </div>
79 )
80 }
81 }
82
83 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Mengakses 6 inputan form menggunakan ref

Di dalam constructor, saya membuat 7 ref yang akan ditempatkan ke dalam atribut ref di setiap inputan form. Ketika form di submit, nilai form diakses dari ref antara baris 16-22.

Untuk inputan checkbox dan radio button, nilai form tersimpan dalam property `<namaRef>.current.checked`. Nilai yang dihasilkan berbentuk boolean `true` atau `false` bergantung apakah inputan tersebut sedang dipilih atau tidak. Untuk inputan lain, nilai form bisa diakses dari `<namaRef>.current.value`.

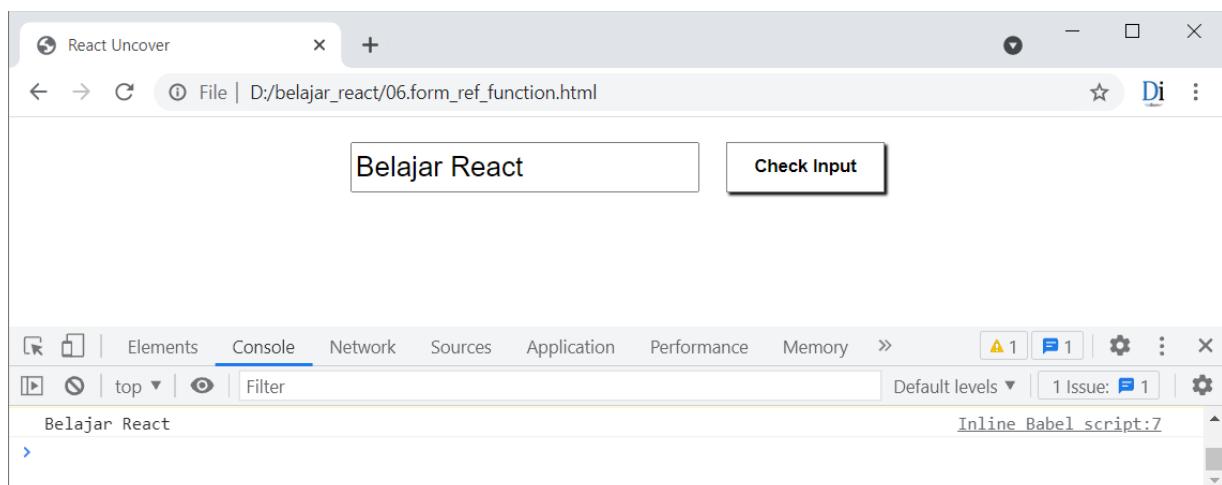
Di versi *functional component*, ref bisa dibuat dari **useRef hook** seperti contoh berikut:

06.form_ref_function.html

```

1  const MyApp = () => {
2      const inputRef = React.useRef();
3
4      const handleButtonClick = () => {
5          console.log(inputRef.current.value);
6      }
7
8      return (
9          <div>
10             <input type="text" ref={inputRef} />
11             <button onClick={handleButtonClick}>Check Input</button>
12         </div>
13     )
14 }
15
16 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Mengakses form dari ref di functional component

Proses pembuatan ref `inputRef` ada di baris 2, yang kemudian diinput sebagai atribut `ref` ke dalam tag `<input>` di baris 10. Cara kerjanya kurang lebih sama seperti di versi *class component*.

Sebagai latihan, silahkan coba konversi contoh 6 inputan form sebelumnya dari *class component* menjadi *functional component*.

Materi dalam bab ini sebenarnya lebih ke alternatif dari proses pengaksesan nilai form menggunakan state dalam bab sebelumnya.

Dokumentasi React menyarankan untuk selalu memakai controller components (menggunakan state), dan barulah di beberapa situasi tertentu memakai uncontrolled components (menggunakan ref) untuk mengakses nilai form.

Bab berikutnya kita masuk ke materi yang lebih serius, yakni mengenal **React Lifecycle**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

13. React Component Lifecycle

Ketika membahas state, salah satu fiturnya adalah komponen akan di render ulang jika terjadi perubahan nilai. Ini merupakan bagian dari *React component lifecycle*, yakni "siklus hidup" dari sebuah komponen. Dalam bab ini kita akan bahas lebih jauh apa yang dimaksud dengan **React Component Lifecycle**.

Sebagian sumber ada yang menyingkat materi ini sebagai **React Lifecycle** saja.

13.1. Memahami React Component Lifecycle

Setiap komponen React memiliki 3 siklus dasar, yakni:

- Mounting
- Updating
- Unmounting

Mounting adalah waktu dimana komponen pertama kali dibuat, termasuk proses render pertama kali. **Updating** adalah peristiwa saat komponen tersebut di update, misalnya terjadi perubahan nilai state, menjalankan method/fungsi tertentu, dst. **Unmounting** adalah waktu saat komponen itu dihapus dari struktur DOM, misalnya ketika menutup sebuah form atau menutup jendela modal/ jendela alert.

React mengizinkan kita melakukan sesuatu di antara ketiga tahap ini, karena ada kalanya sebuah tugas lebih cocok dilakukan di saat *mounting*, daripada *updating* dan sebaliknya.

Caranya adalah dengan menulis kode program di salah satu dari 5 method khusus bawaan React, yakni:

- `constructor()`
- `render()`
- `componentDidMount()`
- `componentDidUpdate()`
- `componentWillUnmount()`

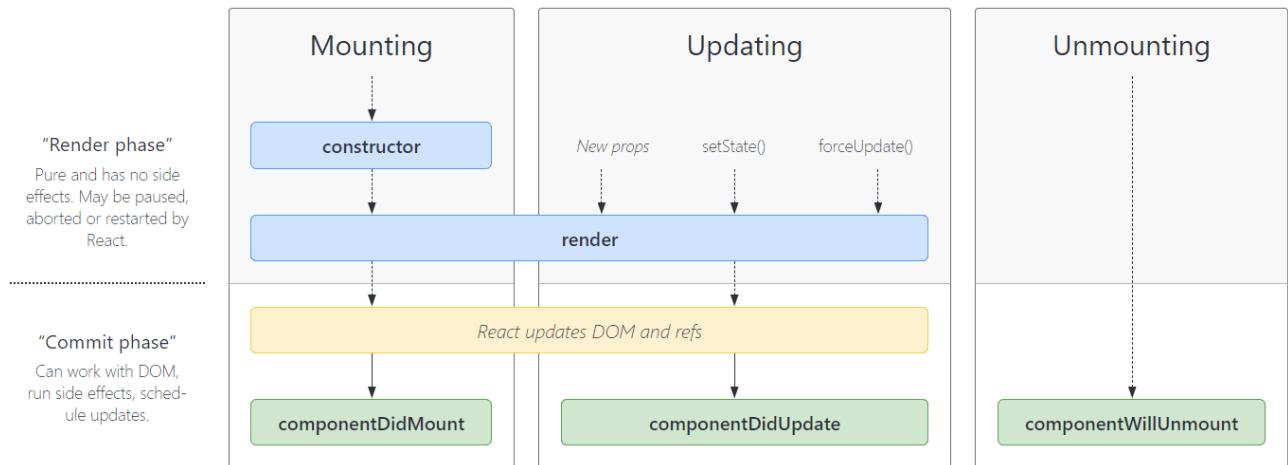
Dua method pertama sudah sering kita pakai.

Method `constructor()` dijalankan sebelum komponen tampil ke dalam web browser, atau lebih tepatnya sebelum terjadi proses *mounting*. Sedangkan method `render()` akan diproses

sekali setelah *mounting*, dan akan terus dijalankan setiap terjadi proses *updating*.

Untuk 3 method lain belum pernah kita pakai, karena yang wajib ada di setiap komponen React hanya method `render()` saja, sisanya method lain boleh saja tidak ditulis (opsional).

Agar lebih memahami 3 method tersebut, kita akan bahas dengan diagram di bawah ini:



Gambar: Diagram React component lifecycle (sumber: projects.wojtekmaj.pl)

Kita mulai dari proses **Mounting** terlebih dahulu. Pada saat komponen pertama kali dibuat, method `constructor()` akan dijalankan, setelah itu giliran method `render()` dan DOM web browser akan di update. Pada titik ini, komponen sudah bisa terlihat di web browser.

Setelah komponen tampil, React tetap mengizinkan kita menjalankan kode tambahan apabila diperlukan. Kode tersebut bisa ditulis ke dalam method `componentDidMount()`, dan proses *mounting* selesai.

Masuk ke proses **Updating**. Tahap ini akan berjalan saat terjadi perubahan nilai state. Hal pertama yang akan di proses adalah menjalankan kembali method `render()` kemudian mengupdate perubahan itu ke DOM ke web browser. Setelah komponen di update, React juga mengizinkan kita menjalankan kode tambahan di dalam method `componentDidUpdate()`.

Proses *updating* ini akan berjalan berulang kali sepanjang aplikasi React digunakan. Setiap kali terjadi perubahan nilai state, method `render()` dan `componentDidUpdate()` akan dijalankan kembali.

Proses ketiga adalah **Unmounting**. Tahap ini baru berjalan ketika komponen dihapus dari struktur DOM. Jika kita butuh menjalankan kode program pas di waktu tersebut, bisa ditulis ke dalam method `componentWillUnmount()`.

Itulah gambaran singkat tentang react component lifecycle. Setiap komponen React akan mengalami ketiga tahapan di atas, tapi tidak semua method harus ditulis. Dari kelima method lifecycle, hanya `render()` yang wajib ada, sisanya bersifat opsional.

13.2. Praktek Mounting, Updating dan Unmounting

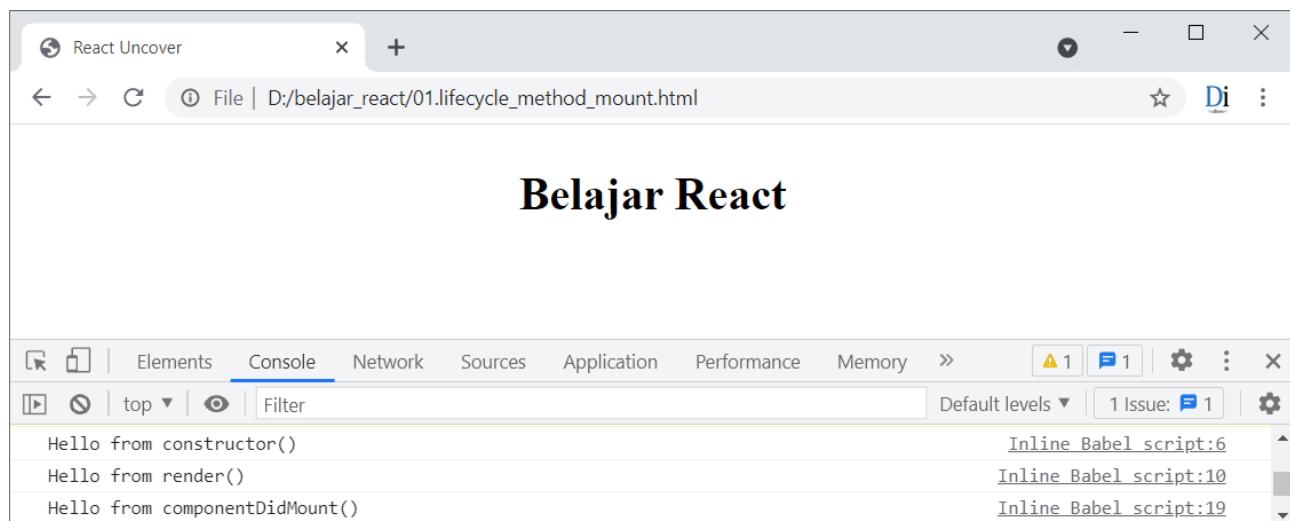
Cukup dengan teori, sekarang kita masuk ke praktek kode program untuk melihat setiap tahapan dari *react component lifecycle*:

01.lifecycle_method_mount.html

```

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          console.log("Hello from constructor()");
5      }
6
7      render() {
8          console.log("Hello from render()");
9          return (
10             <div>
11                 <h1> Belajar React </h1>
12             </div>
13         )
14     }
15
16     componentDidMount() {
17         console.log("Hello from componentDidMount()");
18     }
19
20     componentDidUpdate() {
21         console.log("Hello from componentDidUpdate()");
22     }
23
24     componentWillUnmount() {
25         console.log("Hello from componentWillUnmount()");
26     }
27 }
28
29 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Menjalankan method React Component Lifecycle

Dalam kode program ini saya membuat ke-5 method *React component lifecycle*. Di setiap method, terdapat perintah `console.log()` sekedar memastikan apakah method tersebut dijalankan React atau tidak.

Pada saat halaman di-load pertama kali, dalam tab Console akan terlihat 3 teks berikut:

```
Hello from constructor()
Hello from render()
Hello from componentDidMount()
```

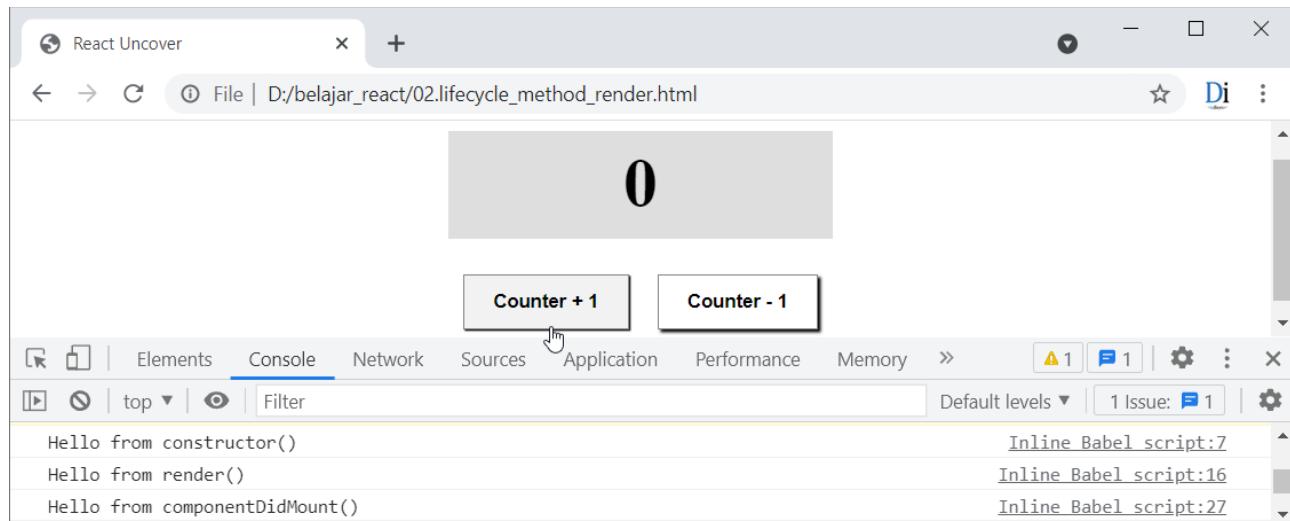
Semua teks ini berasal dari 3 method yang ada di proses *mounting*, yakni `constructor()`, `render()` dan `componentDidMount()`. Isi method lain tidak tampil karena kita tidak menggunakan state sehingga proses *updating* tidak akan berjalan.

Mari tambah state ke dalam kode program:

02.lifecycle_method_render.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14         h1 {
15             background-color: gainsboro;
16             font-size: 3em;
17             text-align: center;
18             padding: 10px 0;
19             margin: 15px 0;
20         }
21
22         button {
23             background-color: white;
24             cursor: pointer;
25             padding: 10px 20px;
26             font-weight: bold;
27             border: 1px solid gray;
28             box-shadow: 2px 2px 2px black;
29             margin: 10px;
30         }
31
32         button:hover {
33             box-shadow: 1px 1px 1px black;
34             background-color: #f1f1f1;
35     }
```

```
36  </style>
37 </head>
38
39 <body>
40   <div id="root"></div>
41
42   <script src="js/react.development.js"></script>
43   <script src="js/react-dom.development.js"></script>
44   <script src="js/babel.js"></script>
45   <script type="text/babel">
46
47     class MyApp extends React.Component {
48       constructor(props) {
49         super(props);
50         this.state = { counter: 0 };
51         console.log("Hello from constructor()");
52     }
53
54     handleButtonClick = (e) => {
55       let val = parseInt(e.target.getAttribute("val"));
56       this.setState((prevState) => ({ counter: prevState.counter + val }));
57     }
58
59     render() {
60       console.log("Hello from render()");
61       return (
62         <div>
63           <h1> {this.state.counter} </h1>
64           <button val="+1" onClick={this.handleButtonClick}>Counter + 1</button>
65           <button val="-1" onClick={this.handleButtonClick}>Counter - 1</button>
66         </div>
67       )
68     }
69
70     componentDidMount() {
71       console.log("Hello from componentDidMount()");
72     }
73
74     componentDidUpdate() {
75       console.log("Hello from componentDidUpdate()");
76     }
77
78     componentWillUnmount() {
79       console.log("Hello from componentWillUnmount()");
80     }
81   }
82
83   ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
84
85 </script>
86 </body>
87
88 </html>
```



Gambar: Praktek method React Component Lifecycle dengan state

Tampilan ini mirip seperti praktek kita di materi state (mini project **counter click**), dimana terdapat 2 buah tombol yang jika di klik akan menambah atau mengurangi state counter.

Sama seperti sebelumnya, pada saat proses loading pertama kali, pesan teks dari method `constructor()`, `render()` dan `componentDidMount()` sudah langsung tampil, yang artinya semua method dari tahap *mounting* sudah berjalan.

Pada saat tombol "Counter + 1" di klik sekali, akan bertambah 2 teks baru di tab Console:

```
Hello from constructor()
Hello from render()
Hello from componentDidMount()
Hello from render()
Hello from componentDidUpdate()
```

Dua teks terakhir berasal dari method `render()` dan `componentDidUpdate()`. Keduanya diproses karena masuk ke tahapan *updating*. Silahkan klik kembali tombol "Counter + 1" atau "Counter - 1", maka kedua teks ini akan tampil berulang.

Satu hal yang bisa disimpulkan adalah, isi method `render()` dan `componentDidUpdate()` akan selalu dijalankan ulang setiap terjadi perubahan nilai state.

Untuk praktek tahap *unmounting*, kita harus buat 2 buah komponen karena *unmounting* hanya bisa terlihat saat komponen dihapus dari struktur DOM:

03.lifecycle_method_unmount.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>React Uncover</title>
```

React Component Lifecycle

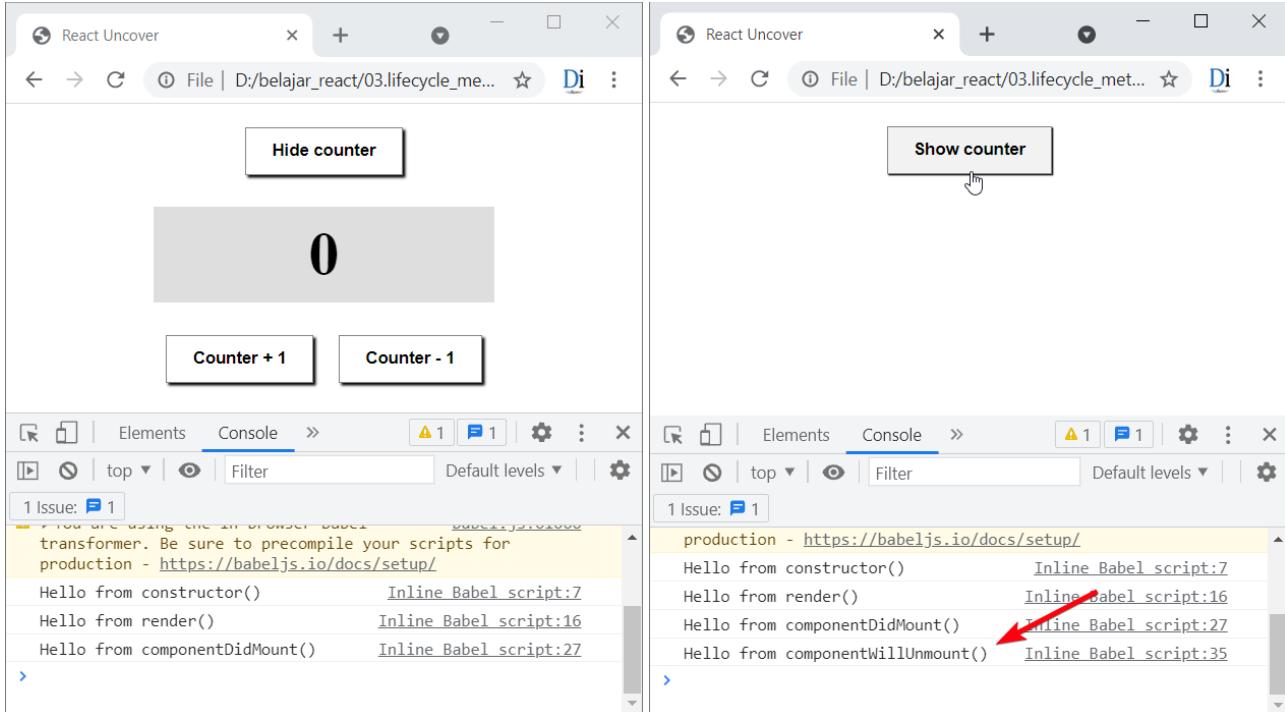
```
8   <style>
9     #root {
10       display: flex;
11       justify-content: center;
12     }
13
14   .container {
15     display: flex;
16     flex-direction: column;
17     align-items: center;
18   }
19
20   h1 {
21     background-color: gainsboro;
22     font-size: 3em;
23     text-align: center;
24     padding: 10px 0;
25     margin: 15px 0;
26   }
27
28   button {
29     background-color: white;
30     cursor: pointer;
31     padding: 10px 20px;
32     font-weight: bold;
33     border: 1px solid gray;
34     box-shadow: 2px 2px 2px black;
35     margin: 10px;
36   }
37
38   button:hover {
39     box-shadow: 1px 1px 1px black;
40     background-color: #f1f1f1;
41   }
42 </style>
43 </head>
44
45 <body>
46   <div id="root"></div>
47
48   <script src="js/react.development.js"></script>
49   <script src="js/react-dom.development.js"></script>
50   <script src="js/babel.js"></script>
51   <script type="text/babel">
52
53     class Counter extends React.Component {
54       constructor(props) {
55         super(props);
56         this.state = { counter: 0 };
57         console.log("Hello from constructor()");
58     }
59
60     handleButtonClick = (e) => {
61       let val = parseInt(e.target.getAttribute("val"));
62       this.setState((prevState) => ({ counter: prevState.counter + val }));
63   
```

```

63     }
64
65     render() {
66       console.log("Hello from render()");
67       return (
68         <div>
69           <h1> {this.state.counter} </h1>
70           <button val="+1" onClick={this.handleButtonClick}>Counter + 1</button>
71           <button val="-1" onClick={this.handleButtonClick}>Counter - 1</button>
72         </div>
73       )
74     }
75
76     componentDidMount() {
77       console.log("Hello from componentDidMount()");
78     }
79
80     componentDidUpdate() {
81       console.log("Hello from componentDidUpdate()");
82     }
83
84     componentWillUnmount() {
85       console.log("Hello from componentWillUnmount()");
86     }
87   }
88
89   class MyApp extends React.Component {
90     constructor(props) {
91       super(props);
92       this.state = { showCounter: false };
93     }
94
95     handleButtonClick = () => {
96       this.setState((prevState) => ({ showCounter: !prevState.showCounter }));
97     }
98
99     render() {
100       return (
101         <div className="container">
102           <button onClick={this.handleButtonClick}>
103             {this.state.showCounter ? "Hide counter" : "Show counter"}
104           </button>
105           {this.state.showCounter && <Counter />}
106         </div>
107       )
108     }
109   }
110
111   ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
112
113   </script>
114 </body>
115
116 </html>

```

React Component Lifecycle



Gambar: Melihat proses unmounting

Kode program counter sebelumnya saya tempatkan ke dalam class terpisah, yakni komponen Counter yang didefinisikan pada baris 53-87. Semua method lifecycle tetap ada di dalam komponen ini.

Komponen Counter kemudian diakses dari dalam komponen MyApp di baris 105. Namun terdapat tambahan logika baru di sini. Komponen Counter baru akan tampil saat tombol "Show counter" di klik, dan akan tersembunyi jika tombol "Hide counter" di klik (toggle).

Idenya adalah dengan membuat sebuah state `showCounter` di baris 92. Di dalam constructor, state `showCounter` pertama kali diisi dengan nilai `false`. Di baris 103 terdapat operator ternary yang akan mengganti isi teks tag `<button>`. Jika state `showCounter` berisi nilai `true`, maka isi tag `<button>` dengan "Hide counter", jika `false` isi dengan "Show counter". Dengan demikian teks tag `<button>` akan berganti-ganti sesuai isi state `showCounter`.

Pada saat tombol tersebut di klik, method `handleButtonClick()` di baris 95-97 akan berjalan. Isinya membalik nilai boolean state `showCounter` dengan perintah `!prevState.showCounter`. Artinya jika saat ini `showCounter` bernilai `false`, setelah tombol di klik akan menjadi `true`, demikian juga sebaliknya.

Nilai state `showCounter` juga dipakai pada baris 105. Tag `<Counter>` hanya tampil jika isi state `showCounter` bernilai `true`.

Dengan kode program ini, komponen Counter akan dibuat dan dihapus setiap kali tombol "Hide/Show counter" di klik. Pada saat terhapus, teks "Hello from componentWillUnmount()" akan tampil sebagai tanda proses unmount sudah berjalan.

13.3. Mini Project: Countdown

Agar fungsi method *lifecycle* ini semakin jelas, kita akan bahas dengan sebuah mini project, yaitu membuat program **countdown** atau hitung mundur.

Program ini menampilkan sebuah angka (misalnya 60) lalu setiap detik akan mundur menjadi 59, 58, dst hingga 0. Supaya efek *unmount* bisa berjalan, countdown saya tempatkan di komponen terpisah dan baru tampil saat tombol di klik, kurang lebih sama seperti praktek komponen Counter sebelum ini.

Berikut kode program awal dari mini project "countdown":

04.countdown.html

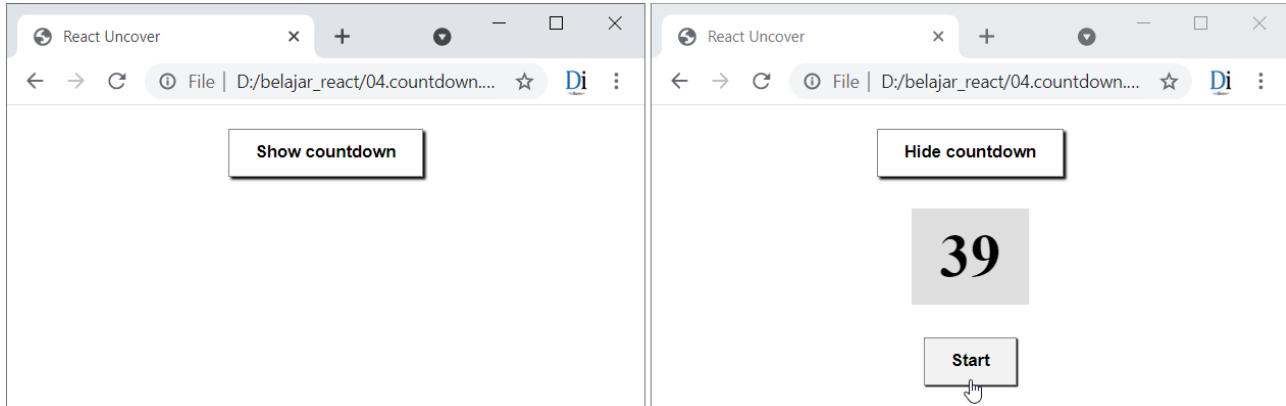
```

1  class Countdown extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { countdown: 60 };
5      }
6
7      handleButtonClick = () => {
8          setInterval(() => {
9              this.setState(
10                  (prevState) => ({ countdown: prevState.countdown - 1 })
11              )
12          }, 1000);
13      }
14
15      render() {
16          console.log(`render: ${this.state.countdown}`);
17          return (
18              <div>
19                  <h1> {this.state.countdown} </h1>
20                  <button onClick={this.handleButtonClick}>Start</button>
21              </div>
22          )
23      }
24  }
25
26  class MyApp extends React.Component {
27      constructor(props) {
28          super(props);
29          this.state = { showcountdown: false };
30      }
31
32      handleButtonClick = () => {
33          this.setState((prevState) => ({ showcountdown: !prevState.showcountdown }));
34      }
35
36      render() {
37          return (
38              <div className="container">
39                  <button onClick={this.handleButtonClick}>
```

```

40         {this.state.showcountdown ? "Hide countdown" : "Show countdown"}
41     </button>
42     {this.state.showcountdown && <Countdown />}
43   </div>
44 )
45 }
46 }
47
48 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```



Gambar: Tampilan mini project "countdown"

Isi kode program komponen `MyApp` di baris 26-46 kurang lebih sama seperti praktik kita sebelum ini.

Komponen `<Countdown/>` baru akan tampil saat tombol "Show countdown" di klik, dan akan terhapus jika tombol "Hide Countdown" di klik. Proses `toggle` di tentukan dari isi state `showcountdown` seperti di baris 42. Karena logika ini sama seperti program Counter sebelumnya, kita langsung saja bahas isi komponen class `Countdown` antara baris 1-24.

Di dalam constructor, terdapat pendefinisian state `countdown` dengan nilai awal 60. State `countdown` inilah yang ditampilkan ke dalam tag `<h1>` di baris 19. Artinya begitu komponen `Countdown` tampil, angka 60 langsung mengisi tag `<h1>`.

Di baris 20, terdapat sebuah tag `<button>` "Start". Ketika tombol ini di klik, method `this.handleButtonClick` akan diproses yang berisi pemanggilan fungsi `setInterval()` bawaan JavaScript.

Sekedar pengingat, `setInterval()` adalah fungsi JavaScript untuk menjalankan kode program setiap interval tertentu. Kode yang ingin dijalankan diinput sebagai argument pertama dalam bentuk `callback`, dan interval diinput sebagai argument kedua dalam satuan milidetik.

Sebagai contoh, jika kita ingin menampilkan teks "Hello" ke tab Console setiap 1 detik sekali, bisa menggunakan perintah berikut:

```
setInterval(() => { console.log("Hello") }, 1000);
```

Di dalam method `handleButtonClick()`, fungsi `setInterval()` berisi kode berikut:

```
setInterval(() => {
  this.setState(
    (prevState) => ({ countdown: prevState.countdown - 1 })
  )
}, 1000);
```

Artinya, isi state `countdown` akan berkurang 1 angka setiap detik. Silahkan klik tombol "Start" untuk memulai proses `countdown`.

Mengatasi Warning "memory leak"

Ketika dijalankan pertama kali, tidak ada masalah dengan kode program kita. Angka `countdown` sukses berkurang 1 angka setiap detik.

Sebagai percobaan, silahkan klik tombol "Hide countdown" saat `countdown` sedang berlangsung. Ternyata, di tab console akan tampil pesan warning berikut:

Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and asynchronous tasks in the componentWillUnmount method.

Pesan ini tampil karena JavaScript masih memiliki jadwal untuk update state `countdown` (berasal dari fungsi `setInterval()`), akan tetapi komponen `Countdown` sudah tidak ada lagi. Seperti yang tertulis di pesan warning, ini bisa menyebabkan "*memory leak*", sebuah bug dimana aplikasi kita memakai ruang memory yang sebenarnya sudah tidak diperlukan lagi.

Jadi, bagaimana mengatasi masalah ini?

Solusinya adalah, kita harus hentikan fungsi `setInterval()` tepat saat komponen `Countdown` dihapus. Ini adalah tugas yang sangat pas untuk method `componentWillUnmount()`.

Di dalam JavaScript, fungsi `setInterval()` sebenarnya mengembalikan suatu angka sebagai identitas atau **id** dari fungsi tersebut. Nilai id ini bisa diinput ke dalam fungsi `clearInterval()` untuk menghentikannya.

Berikut contoh praktek singkat di JavaScript native:

```
// jalankan interval
let intervalID = setInterval(() => { console.log("Hello") }, 1000);

// hentikan interval
clearInterval(intervalID);
```

Inilah yang bisa kita isi ke dalam `componentWillUnmount()`. Berikut modifikasi dari komponen `Countdown`:

```
05.countdown_unmount.html
1 class Countdown extends React.Component {
2   constructor(props) {
```

```

3      super(props);
4      this.state = { countdown: 60 };
5      this.intervalID = null;
6  }
7
8  handleButtonClick = () => {
9      this.intervalID = setInterval(() => {
10         this.setState(
11             (prevState) => ({ countdown: prevState.countdown - 1 })
12         )
13     }, 1000);
14 }
15
16 render() {
17     console.log(`render: ${this.state.countdown}`);
18     return (
19         <div>
20             <h1> {this.state.countdown} </h1>
21             <button onClick={this.handleButtonClick}>Start</button>
22         </div>
23     )
24 }
25
26 componentWillUnmount() {
27     clearInterval(this.intervalID);
28     console.log("componentWillUnmount diproses");
29 }
30
31 }
32 ...

```

Di dalam constructor, saya membuat satu property `this.intervalID` dan mengisinya dengan nilai `null`. Property `this.intervalID` akan menampung nilai interval id tepat saat tombol "Start" di klik, yakni di baris 9.

Property `this.intervalID` selanjutnya dipakai sebagai nilai input untuk method `clearInterval()` yang ada di baris 27, tepatnya di dalam method `componentWillUnmount()`. Selain itu terdapat tambahan perintah `console.log` sekedar penanda bahwa method `componentWillUnmount()` sudah berjalan.

Di baris 17 terdapat tambahan perintah `console.log()` sekedar untuk melihat isi state `countdown` setiap proses re-render terjadi.

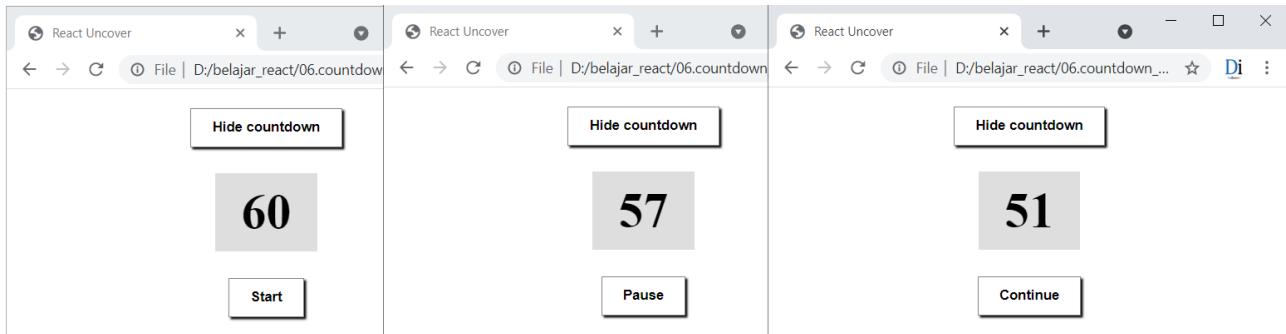
Sekarang silahkan ulangi percobaan sebelumnya. Jalankan countdown dan klik tombol "Hide countdown" saat hitung mundur sedang berlangsung. Hasilnya, tidak ada lagi pesan warning dan teks "componentWillUnmount diproses" tampil di tab Console.

Menambah Tombol "Stop"

Agar lebih menarik, saya ingin tambah tombol "Stop" dan "Continue" ke dalam aplikasi Countdown. Tombol ini akan berbagi dengan tombol "Start".

Pada saat tampil pertama kali, tombol berisi teks "Start" yang ketika di klik akan memulai proses hitung mundur. Pada saat yang bersamaan, teks tombol akan berganti menjadi "Stop" yang ketika di klik akan menghentikan hitung mundur untuk sementara. Teks tombol selanjutnya akan berganti menjadi "Continue" yang jika di klik akan melanjutkan proses countdown.

Tombol "Continue" dan "Stop" berganti-ganti sesuai posisi apakah proses countdown sedang berjalan atau sedang berhenti. Berikut tampilan akhir yang diinginkan:



Gambar: Tombol "Start", "Pause", dan "Continue"

Untuk membuat fitur ini, tentunya perlu sebuah kondisi logika di dalam method `handleButtonClick()`. Logika inilah yang akan menentukan kapan tombol tampil sebagai "Start", "Pause", atau "Continue". Berikut kode program yang saya gunakan:

06.countdown_stop.html

```

1  class Countdown extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { countdown: 60, run: false, buttonName: "Start" };
5          this.intervalID = null;
6      }
7
8      handleButtonClick = () => {
9          if (this.state.run) {
10              clearInterval(this.intervalID);
11              this.setState({ run: false, buttonName: "Continue" })
12          }
13          else {
14              this.intervalID = setInterval(() => {
15                  this.setState(
16                      (prevState) => ({ countdown: prevState.countdown - 1 })
17                  )
18              }, 1000);
19              this.setState({ run: true, buttonName: "Pause" })
20          }
21      }
22
23      render() {
24          console.log(`render: ${this.state.countdown}`);
25          return (

```

```

26     <div className="container">
27         <h1> {this.state.countdown} </h1>
28         <button onClick={this.handleButtonClick}>
29             {this.state.buttonName}
30         </button>
31     </div>
32 )
33 }
34
35 componentWillMount() {
36     clearInterval(this.intervalID);
37     console.log("componentWillMount diproses");
38 }
39 }
40 ...

```

Di baris 4, terdapat 2 state tambahan selain `countdown`, yakni `run` dan `buttonName`.

State `run` akan bertindak sebagai *flag* atau penanda apakah saat ini `countdown` sedang berjalan atau tidak. Nilai awal state diisi dengan `false` yang artinya `countdown` sedang berhenti. Sedangkan state `buttonName` dipakai untuk menyimpan teks yang nantinya dipakai sebagai nama tombol. Untuk nilai awal, state `buttonName` diisi string "Start".

Di dalam method `render()`, satu-satunya perubahan ada di baris 30, dimana teks untuk tag `<button>` sekarang berasal dari `this.state.buttonName`. Dengan demikian pada saat tampil pertama kali, `countdown` belum berjalan dan tombol akan berisi teks "Start" sesuai nilai awal state `buttonName`.

Begini tombol "Start" di klik, method `handleButtonClick()` akan berjalan. Percabangan kondisi di dalam method ini ditentukan oleh isi state `run`. Karena awalnya berisi boolean `false`, maka perintah yang dijalankan ada di blok `else` (baris 14-20).

Di dalam blok `else`, fungsi `setInterval()` akan memulai proses hitung mundur. Selain itu di baris 20 state `run` di update menjadi `true` dan state `buttonName` diisi string "Stop". Dengan perubahan ini, nama tombol juga akan berganti dari "Start" menjadi "Stop".

Sekarang, jika tombol "Stop" di klik, method `handleButtonClick()` akan masuk ke percabangan baris 10-11, sebab state `run` sudah berisi boolean `true`. Di sini, fungsi `clearInterval()` akan menghentikan proses hitung mundur serta mengupdate state `run` kembali menjadi `false` dan state `buttonName` diisi string "Continue".

Lanjut, saat tombol "Continue" di klik, method `handleButtonClick()` akan kembali masuk ke blok `else`, sebab state `run` sekarang berisi boolean `false`. Isinya kembali menjalankan fungsi `setInterval()` untuk memulai lagi proses hitung mundur. Angka hitung mundur berlanjut dari posisi terakhir karena state `countdown` masih berisi nilai sebelumnya. Proses ini akan terus berulang dan kita sudah berhasil membuat tombol "Stop" dan "Continue".

Auto start Countdown

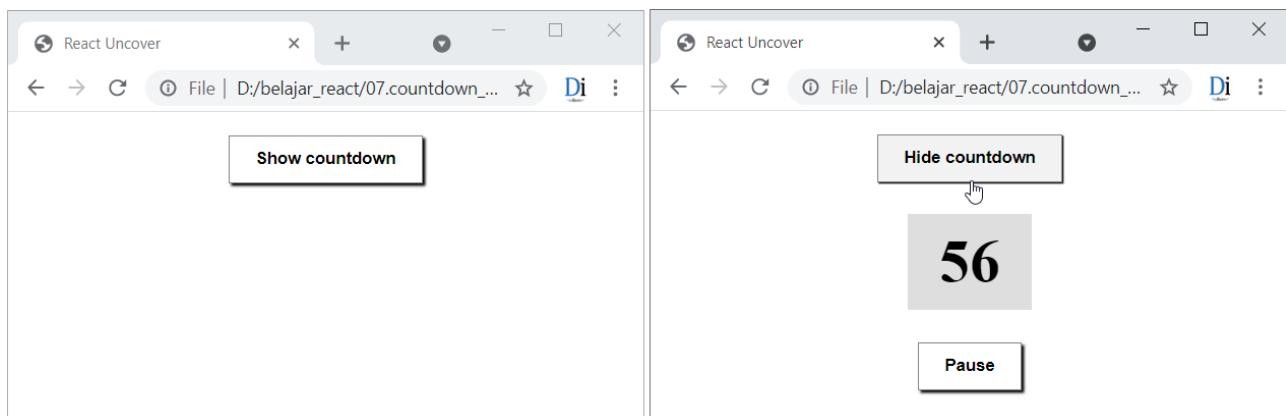
Fitur lain yang ingin saya tambah adalah langsung menjalankan hitung mundur ketika komponen <Countdown/> tampil pertama kali. Saat ini, countdown baru dimulai setelah tombol "Start" di klik.

Salah satu caranya, bisa dengan menambah fungsi `setInterval()` baru ke dalam constructor:

07.countdown_auto_start.html

```

1  class Countdown extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = { countdown: 60, run: true, buttonName: "Pause" };
5
6          this.intervalID = setInterval(() => {
7              this.setState(
8                  prevState => ({ countdown: prevState.countdown - 1 })
9              )
10         }, 1000);
11     }
12 ...
13 ...
14 }
```



Gambar: Menjalankan "auto start countdown"

Dengan tambahan fungsi `setInterval()` di baris 6-10, proses hitung mundur langsung berjalan ketika tombol "Show Countdown" di klik. Selain itu nilai awal state `run` dan `buttonName` di baris 4 juga perlu penyesuaian agar tetap berfungsi sebagaimana mestinya (langsung di set `run:true` dan `buttonName:"Pause"`).

Kode program di atas sudah sukses berjalan, namun ada konsep "*single responsibility principle*" yang dilanggar. Konsep ini sebenarnya lebih ke saran dalam pembuatan kode program, yang salah satu syaratnya: "sebuah fungsi atau method harus menjalankan satu tugas saja".

Dalam kode program kita, constructor memiliki tugas tambahan untuk menjalankan `setInterval()`, seharusnya constructor hanya berisi kode untuk proses inisialisasi saja seperti

pembuatan state.

Jadi, kemana fungsi `setInterval()` ini harus dipindah? Yup, ke method `componentDidMount()`. Method `componentDidMount()` menjadi kandidat yang paling pas karena langsung berjalan pada saat *mounting*.

Berikut modifikasi kode untuk komponen `Countdown`:

08.countdown_didmount.html

```

1  class Countdown extends React.Component {
2
3      constructor(props) {
4          super(props);
5          this.state = { countdown: 60, run: false, buttonName: "Start" };
6      }
7
8      componentDidMount() {
9          this.intervalID = setInterval(() => {
10              this.setState(
11                  {prevState} => ({countdown: prevState.countdown - 1})
12              )
13          }, 1000);
14
15          this.setState({ run: true, buttonName: "Pause" })
16          console.log("componentDidMount diproses");
17      }
18  ...

```

Method `componentDidMount()` ada di baris 8-17, isinya menjalankan fungsi `setInterval()` serta mengupdate state `run` dan `buttonName`.

Di dalam constructor, nilai state `run` dan `buttonName` kembali saya isi dengan nilai awal `false` dan "Start", meskipun nantinya akan langsung berganti menjadi `true` dan "Pause" di dalam method `componentDidMount()`.

Mengatasi Masalah Countdown Negatif

Countdown kita ternyata masih memiliki masalah lain. Jika hitung mundur ini terus dibiarkan, angkanya bisa menjadi negatif. Ini karena tidak ada pemeriksaan kondisi untuk menghentikan `countdown`.

Untuk mencegah hal ini, harus ada kode yang memantau nilai state `countdown` secara berkala. Jika nilainya sudah sampai ke angka 0, hentikan fungsi `setInterval()`.

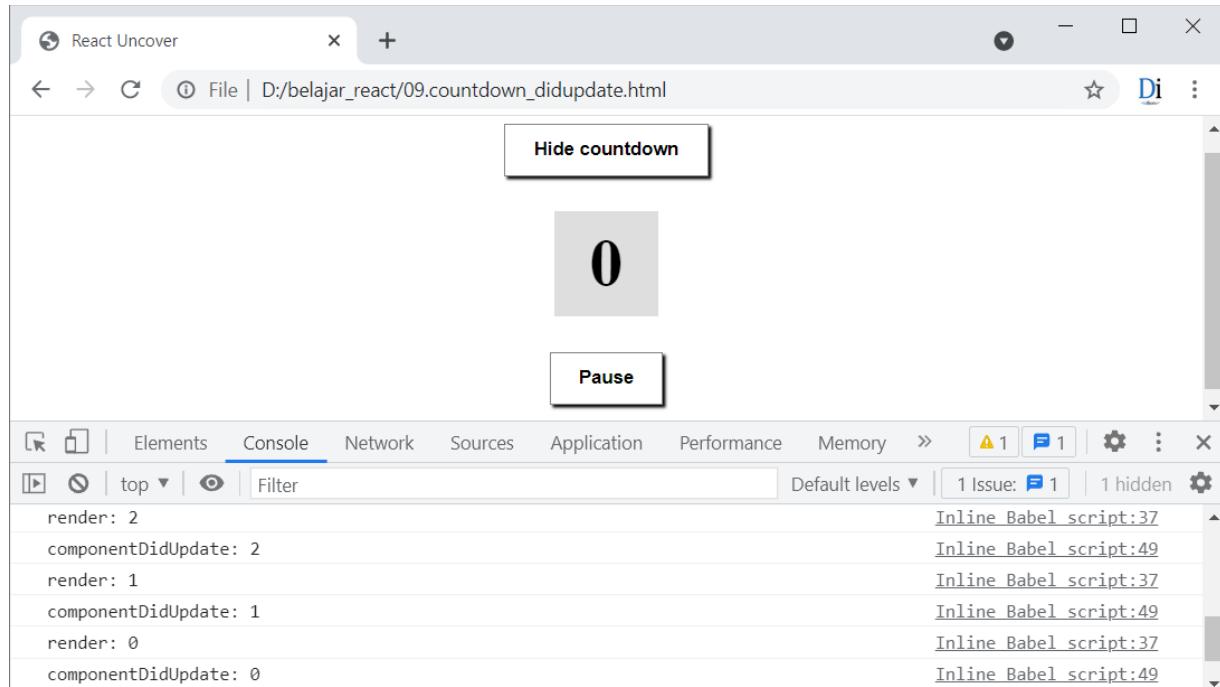
Proses pemeriksaan ini harus berjalan terus-menerus (selama `countdown` berlangsung), sehingga method `componentDidUpdate()` menjadi kandidat yang paling pas. Berikut kode tambahan yang diperlukan:

09.countdown_didupdate.html

```
1 ...
2     componentDidUpdate() {
3         if (this.state.countdown === 0) {
4             clearInterval(this.intervalID);
5         }
6         console.log(`componentDidUpdate: ${this.state.countdown}`);
7     }
8 ...
```

Di baris 3 terdapat pemeriksaan kondisi apakah `this.state.countdown === 0`, jika iya maka hentikan hitung mundur dengan memanggil fungsi `clearInterval(this.intervalID)`. Untuk membantu proses debugging, saya menambah satu perintah `console.log()` di baris 6.

Agar tidak perlu menunggu, modifikasi juga nilai awal state `countdown` menjadi 5, lalu lihat apakah hitung mundur ini berhenti jika sudah sampai ke angka 0. Buka juga tab Console web browser untuk melihat nilai state `countdown` pada setiap tahapan lifecycle:



Gambar: Melihat isi state countdown pada setiap method React component lifecycle

Multiple Countdown

Mini project "Countdown" kita sudah selesai. Sebagai bonus, saya ingin menempatkan banyak countdown ke sebuah halaman. Caranya cukup dengan menulis beberapa kali tag `<Countdown>`.

Berikut kode program lengkap dari mini project "Countdown":

10.countdown_multiple.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14          .container {
15              display: flex;
16              flex-direction: column;
17              align-items: center;
18          }
19
20          h1 {
21              background-color: gainsboro;
22              font-size: 3em;
23              text-align: center;
24              padding: 10px 25px;
25              margin: 15px 0;
26          }
27
28          button {
29              background-color: white;
30              cursor: pointer;
31              padding: 10px 20px;
32              font-weight: bold;
33              border: 1px solid gray;
34              box-shadow: 2px 2px 2px black;
35              margin: 10px;
36          }
37
38          button:hover {
39              box-shadow: 1px 1px 1px black;
40              background-color: #f1f1f1;
41          }
42      </style>
43  </head>
44
45  <body>
46      <div id="root"></div>
47
48      <script src="js/react.development.js"></script>
49      <script src="js/react-dom.development.js"></script>
50      <script src="js/babel.js"></script>
51      <script type="text/babel">
52
53          class Countdown extends React.Component {
```

React Component Lifecycle

```
54  constructor(props) {
55    super(props);
56    this.state = { countdown: 5, run: false, buttonName: "Start" };
57  }
58
59  componentDidMount() {
60    this.intervalID = setInterval(() => {
61      this.setState(
62        (prevState) => ({ countdown: prevState.countdown - 1 })
63      )
64    }, 1000);
65
66    this.setState({ run: true, buttonName: "Pause" })
67    console.log("componentDidMount diproses");
68  }
69
70  handleButtonClick = () => {
71    if (this.state.run) {
72      clearInterval(this.intervalID);
73      this.setState({ run: false, buttonName: "Continue" })
74    }
75    else {
76      this.intervalID = setInterval(() => {
77        this.setState(
78          (prevState) => ({ countdown: prevState.countdown - 1 })
79        )
80      }, 1000);
81
82      this.setState({ run: true, buttonName: "Pause" })
83    }
84  }
85
86  render() {
87    console.log(`render: ${this.state.countdown}`);
88    return (
89      <div className="container">
90        <h1> {this.state.countdown} </h1>
91        <button onClick={this.handleButtonClick}>
92          {this.state.buttonName}
93        </button>
94      </div>
95    )
96  }
97
98  componentDidUpdate() {
99    if (this.state.countdown === 0) {
100      clearInterval(this.intervalID);
101    }
102    console.log(`componentDidUpdate: ${this.state.countdown}`);
103  }
104
105  componentWillUnmount() {
106    clearInterval(this.intervalID);
107    console.log("componentWillUnmount diproses");
108  }
```

React Component Lifecycle

```
109      }
110  }
111
112  class MyApp extends React.Component {
113    constructor(props) {
114      super(props);
115      this.state = { showcountdown: false };
116    }
117
118    handleButtonClick = () => {
119      this.setState((prevState) =>
120        ({ showcountdown: !prevState.showcountdown }));
121    }
122
123    render() {
124      return (
125        <div className="container">
126          <button onClick={this.handleButtonClick}>
127            {this.state.showcountdown ? "Hide countdown" : "Show countdown"}
128          </button>
129          {this.state.showcountdown &&
130            <div style={{ display: "flex" }}>
131              <Countdown />
132              <Countdown />
133              <Countdown />
134            </div>
135          }
136        </div>
137      )
138    }
139  }
140
141  ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
142
143  </script>
144 </body>
145
146 </html>
```



Gambar: Membuat 3 buah countdown

Untuk membuat 3 buah countdown, cukup tulis 3 kali tag <Countdown /> seperti di baris 131-133. Setiap countdown sudah langsung berjalan dan secara internal terpisah satu sama lain (tidak ada bentrok variabel/state). Ini menjadi salah satu keunggulan *React component*, dimana setiap komponen punya "lingkungan terpisah" satu sama lain.

13.4. useEffect Hook

Sekarang kita masuk ke **React component lifecycle** di functional component. Berbeda dengan class component, functional component tidak memiliki method atau fungsi *lifecycle* tersendiri. Semua tahapan *lifecycle* akan di proses oleh satu hook khusus bernama **useEffect**.

Di functional component, terdapat istilah "**effect**" atau "**side effect**", yakni kode program yang menjalankan fungsi tambahan di luar proses render JSX. Tugas utama komponen sebenarnya hanya menampilkan user interface, tugas lain di luar itu dianggap sebagai *side effect*. Inilah alasan penamaan **useEffect** hook.

Karena kita sudah membahas berbagai method *lifecycle* di class component, materi tentang **useEffect** ini menjadi sedikit lebih mudah.

Salah satu alasan saya kenapa buku React Uncover ini masih tetap membahas class component dan tidak langsung ke functional component adalah karena materi **useEffect** hampir selalu dijelaskan dengan method-method *lifecycle* milik class component.

Jadi jika kita tidak bisa membaca diagram *lifecycle* di class component (*mounting*, *updating* dan *unmounting*), materi **useEffect** akan terasa lebih susah.

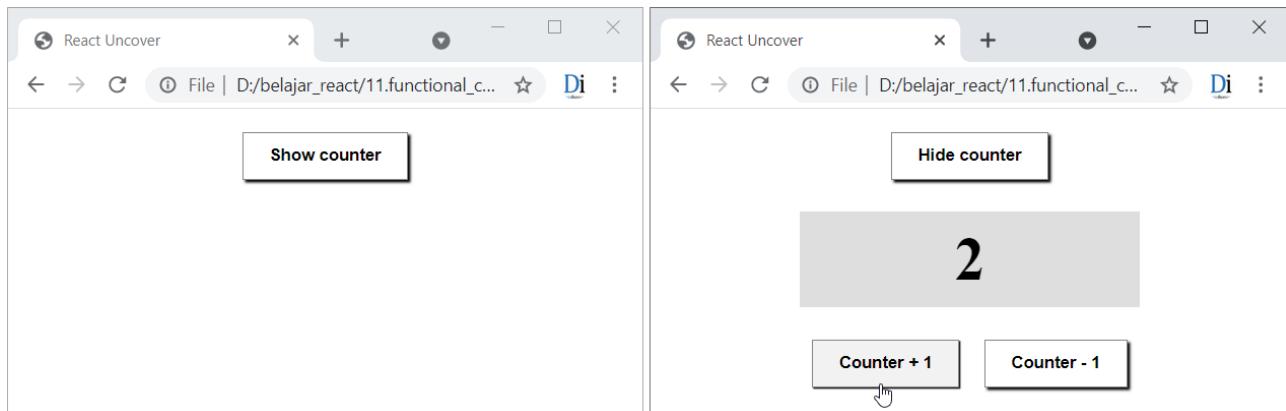
Sebagai bahan praktik, saya kembali memakai komponen **Counter**, tapi kali ini dalam versi functional component:

11.functional_component.html

```

1  const Counter = () => {
2      const [counter, setCounter] = React.useState(0);
3
4      const handleButtonClick = (e) => {
5          let val = parseInt(e.target.getAttribute("val"));
6          setCounter((prevState) => prevState + val)
7      }
8
9      return (
10         <div>
11             <h1> {counter} </h1>
12             <button val="+1" onClick={handleButtonClick}>Counter + 1</button>
13             <button val="-1" onClick={handleButtonClick}>Counter - 1</button>
14         </div>
15     )
16 }
```

```
17 const MyApp = () => {
18   const [showCounter, setShowCounter] = React.useState(false);
19
20   const handleButtonClick = () => {
21     setShowCounter((prevState) => (!prevState))
22   }
23
24   return (
25     <div className="container">
26       <button onClick={handleButtonClick}>
27         {showCounter ? "Hide counter" : "Show counter"}
28       </button>
29       {showCounter && <Counter />}
30     </div>
31   )
32 }
33 }
34
35 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```



Gambar: Menampilkan counter dalam functional component

Kode CSS untuk tampilan ini sama seperti yang dipakai untuk versi class component.

Tidak ada yang baru dalam kode program ini. Komponen **MyApp** dipakai untuk membuat tombol "Show counter" yang ketika di klik akan menampilkan komponen **Counter**. Nilai state counter naik atau turun 1 angka setiap kali tombol "Counter +1" atau "Counter -1" di klik.

useEffect Setelah Proses Render (Updating)

useEffect hook ditulis dengan perintah `React.useEffect()`. Method ini bisa menerima 2 argument, yakni sebuah fungsi *callback* sebagai argument pertama dan array sebagai argument kedua. Array inilah yang nantinya menjadi penentu kapan *callback* akan berjalan.

Jika array di argument kedua tidak ditulis, maka fungsi *callback* milik `useEffect` akan berjalan setiap kali proses render selesai. Berikut contohnya:

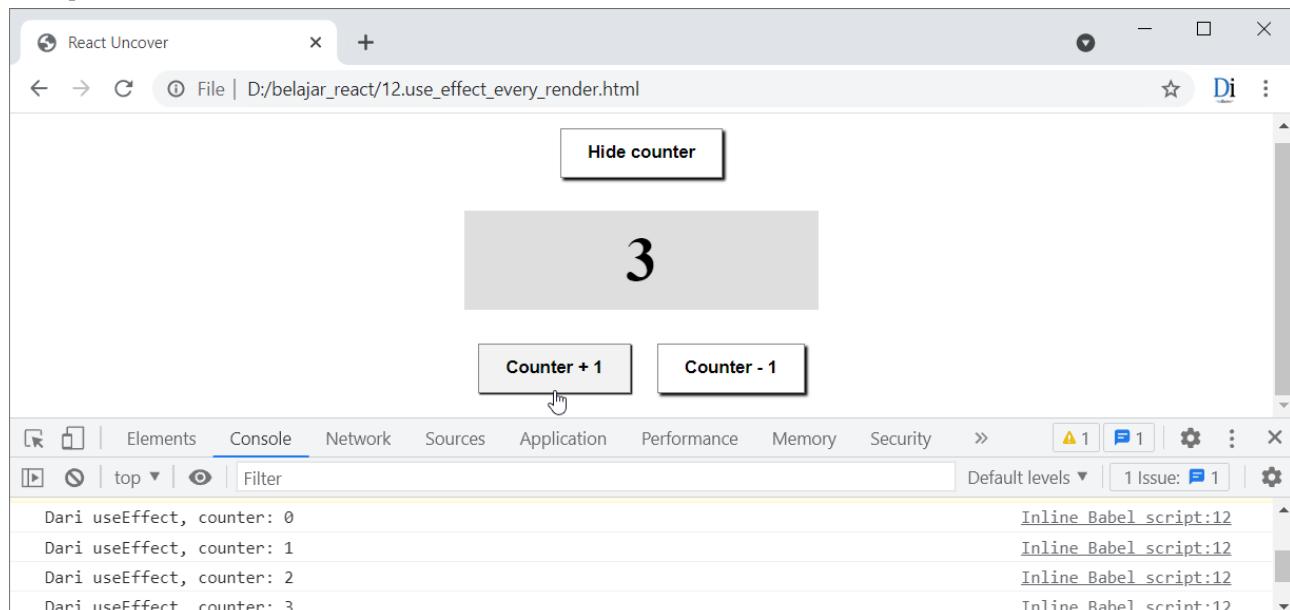
12.use_effect_every_render.html

```
1 const Counter = () => {
2   ...
3   React.useEffect(() => {
4     console.log(`Dari useEffect, counter: ${counter}`);
5   });
6   ...
7 }
```

Di baris 3, method `React.useEffect()` saya isi dengan 1 argument saja, yakni sebuah function dalam format *arrow notation*. Agar lebih jelas, berikut isi function tersebut:

```
() => { console.log(`Dari useEffect, counter: ${counter}`); }
```

Isi function hanya menjalankan 1 perintah `console.log()`. Silahkan tambah ke dalam komponen Counter, dan lihat ke dalam tab Console:



Gambar: Pesan teks tampil setiap kali komponen di render ulang

Teks "Dari useEffect, counter: x" akan tampil setiap kali tombol di klik. Ini membuktikan kalau function callback milik `useEffect` berjalan setiap terjadi proses re-render.

Sekilas ini mirip seperti `componentDidUpdate()`, akan tetapi ada tambahan lain. Pada saat komponen Counter pertama kali tampil (dengan men-klik tombol "Show counter"), teks "Dari useEffect, counter: 0" sudah langsung terlihat.

Jika kita menganggap `useEffect` ini sebagai pengganti method `componentDidUpdate()`, harusnya teks awal ini tidak tampil. Method `componentDidUpdate()` baru berjalan pada proses render kedua, karena yang pertama menjadi jatah method `componentDidMount()`. Silahkan lihat kembali diagram *lifecycle* pada awal bab jika agak ragu dengan penjelasan ini.

Bukti bahwa teks "Dari useEffect, counter: 0" tampil menjelaskan bahwa `useEffect` yang kita

tulis adalah pengganti dari `componentDidMount()` dan juga `componentDidUpdate()` sekaligus. Dengan kata lain, `useEffect` tersebut akan berjalan setiap kali terjadi proses render.

Setelah penelusuran lebih lanjut, saya menemukan bahwa tim React menggabungkan 2 tahap ini karena melihat pada umumnya terdapat duplikasi kode antara method `componentDidUpdate()` dan `componentDidMount()`. Biasanya, kode yang dijalankan `componentDidUpdate()` juga perlu kita tulis ke dalam `componentDidMount()`, jadi sekalian disatukan ke dalam satu fungsi.

useEffect untuk Tahap Mounting

Sebelumnya kita sempat bahas kalau argument kedua dari method `React.useEffect()` bertugas untuk menentukan kapan effect tersebut berjalan. Jika argument kedua ini tidak ditulis, maka side effect akan berjalan setiap kali proses render selesai.

Sekarang jika argument kedua diisi dengan array kosong " [] ", maka side effect tersebut akan berjalan satu kali di awal saja, yang dalam hal ini mirip seperti method `componentDidMount()`.

Silahkan modifikasi method `React.useEffect()` sebelumnya dengan tambahan tanda " [] " sebagai argument kedua:

13.use_effect_once.html

```
1 const Counter = () => {
2   ...
3   React.useEffect(() => {
4     console.log(`Dari useEffect, counter: ${counter}`);
5   }, []);
6   ...
7 }
```

Dengan tambahan di baris 5, teks di tab Console hanya muncul 1 kali di awal saja, tepat pada saat komponen Counter ditampilkan pertama kali. Inilah cara menjalankan side effect yang mirip seperti method `componentDidMount()`.

useEffect untuk Tahap Unmouting

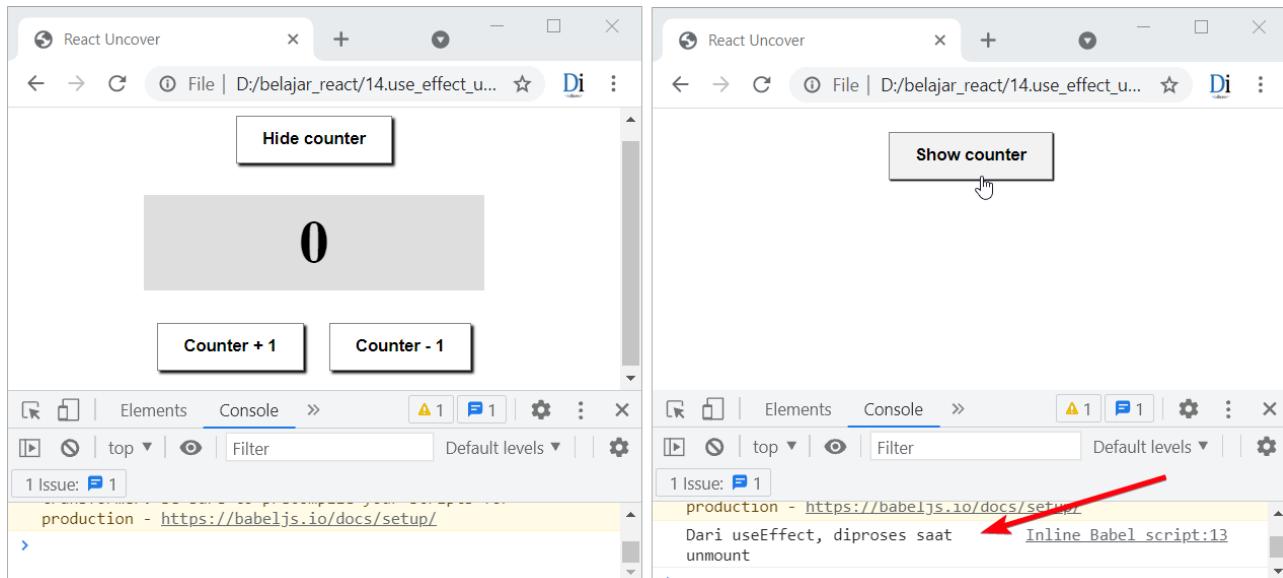
Jika kita ingin menjalankan side effect pada tahap *unmounting* (saat komponen dihapus dari struktur DOM), tulis kode tersebut sebagai nilai kembalian dari function *callback*, yakni di dalam perintah `return`. Berikut contoh penulisannya:

14.use_effect_unmount.html

```
1 const Counter = () => {
2   ...
3   React.useEffect(() => {
4     return () => {
5       console.log(`Dari useEffect, diproses saat unmount`);
6     }
7 }
```

```
7     }, []);
8 ...
9 }
```

Dengan cara ini, perintah `console.log()` di baris 5 akan tampil hanya saat komponen Counter di hapus dari struktur DOM. Dalam praktik kita, ini terjadi saat tombol "Hide counter" di klik:



Gambar: Side effect berjalan saat proses unmounting

useEffect untuk Tahap Mouting dan Unmouting

Menggabungkan cara menjalankan side effect pada saat mounting dan unmouting, keduanya bisa ditulis dalam satu pemanggilan method `React.useEffect()`:

15.use_effect_mount_unmount.html

```
1 const Counter = () => {
2 ...
3   React.useEffect(() => {
4     console.log(`Dari useEffect, counter: ${counter}`);
5     return () => {
6       console.log(`Dari useEffect, diproses saat unmount`);
7     }
8   }, []);
9 ...
10 }
```

Nantinya kita bisa ganti kode program di baris 4 dan 6 sesuai kebutuhan.

Menjalankan Beberapa useEffect

Untuk aplikasi yang kompleks, React mengizinkan kita menjalankan beberapa `useEffect()` pada komponen yang sama. Misalnya ingin menjalankan kode A setiap kali re-render, dan menjalankan kode B saat proses *mounting* dan *unmounting*.

Berikut contoh penulisannya:

16.use_effect_mount_multiple.html

```

1 const Counter = () => {
2   ...
3   // Ini dijalankan untuk setiap render, kecuali unmount
4   React.useEffect(() => {
5     console.log(`Dari useEffect 1, counter: ${counter}`);
6   });
7
8   // Ini dijalankan di awal, dan pada saat unmount
9   React.useEffect(() => {
10    console.log(`Dari useEffect 2, counter: ${counter}`);
11    return () => {
12      console.log(`Dari useEffect 2, diproses saat unmount`);
13    }
14  }, []);
15  ...
16 }
```

Pemisahan ini membuat kode program lebih mudah di kelola. Daripada menyatukan semua side effect, lebih baik baik dipecah sesuai tugasnya masing-masing.

Menjalankan useEffect untuk State Tertentu

Fitur lain dari useEffect yang akan sering kita pakai adalah, memberi batasan sesuai perubahan nilai state.

Maksudnya, kita bisa membatasi agar side effect berjalan hanya jika terjadi perubahan nilai state tertentu, bukan setiap proses re-render. Caranya, tulis nama state ke dalam argument kedua method `React.useEffect()` seperti contoh berikut:

17.use_effect_dependency_state_1.html

```

1 const Counter = () => {
2   const [counter, setCounter] = React.useState(0);
3
4   React.useEffect(() => {
5     console.log(`Dari useEffect, counter: ${counter}`);
6   }, [counter]);
7   ...
8 }
```

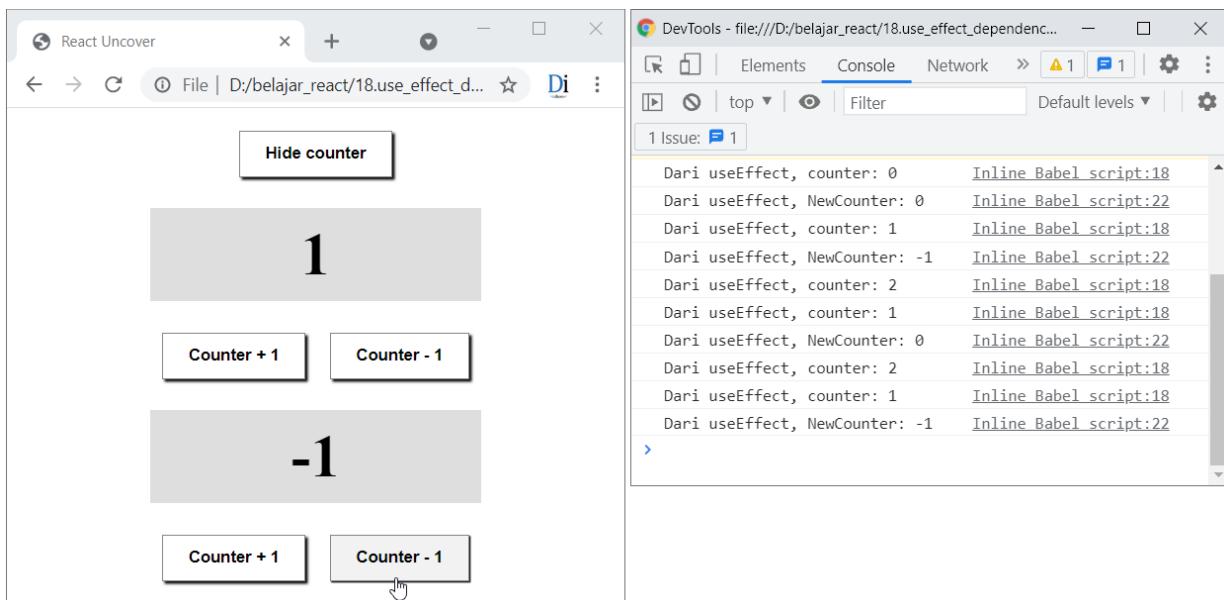
Di dalam komponen Counter, state `counter` dipakai untuk menyimpan angka counter. Agar side effect hanya berjalan ketika nilai state `counter` berubah saja, tambahkan `counter` ke dalam element array argument kedua seperti di baris 6. Ini membuat kode kita lebih efisien, terutama jika terdapat banyak state di dalam komponen tersebut.

Sebagai contoh lain dari praktek ini, saya akan tambah state `newCounter`:

18.use_effect_dependency_state_2.html

```

1  const Counter = () => {
2      const [counter, setCounter] = React.useState(0);
3      const [newCounter, setNewCounter] = React.useState(0);
4
5      const handleButtonClick = (e) => {
6          let val = parseInt(e.target.getAttribute("val"));
7          setCounter((prevState) => prevState + val)
8      }
9
10     const handleNewButtonClick = (e) => {
11         let val = parseInt(e.target.getAttribute("val"));
12         setNewCounter((prevState) => prevState + val)
13     }
14
15     React.useEffect(() => {
16         console.log(`Dari useEffect, counter: ${counter}`);
17     }, [counter]);
18
19     React.useEffect(() => {
20         console.log(`Dari useEffect, NewCounter: ${newCounter}`);
21     }, [newCounter]);
22
23     return (
24         <div>
25             <h1> {counter} </h1>
26             <button val="+1" onClick={handleButtonClick}>Counter + 1</button>
27             <button val="-1" onClick={handleButtonClick}>Counter - 1</button>
28             <h1> {newCounter} </h1>
29             <button val="+1" onClick={handleNewButtonClick}>Counter + 1</button>
30             <button val="-1" onClick={handleNewButtonClick}>Counter - 1</button>
31         </div>
32     )
33 }
```



Gambar: Menjalankan side effect berdasarkan state

Sekarang ada 2 buah counter di dalam komponen Counter. Counter pertama disimpan ke dalam state counter, dan counter kedua disimpan ke dalam state newCounter. Di baris 15-21, terdapat 2 buah method `useEffect()` yang masing-masingnya hanya akan berjalan sesuai nilai state yang ditulis pada array argument kedua.

Dengan cara ini, perubahan di state counter, tidak akan menjalankan side effect milik state newCounter.

13.5. Mini Project: Countdown (Functional Component)

Setelah mempelajari penggunaan `useEffect`, kita akan konversi mini project Countdown dari class component menjadi functional component. Sekilas ini tidak terlalu sulit, yang dibutuhkan hanya mengubah penulisan method lifecycle ke dalam `useEffect` hook.

Berikut kode lengkap mini project Countdown versi functional component:

```

1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14          .container {
15              display: flex;
16              flex-direction: column;
17              align-items: center;
18          }
19
20          h1 {
21              background-color: gainsboro;
22              font-size: 3em;
23              text-align: center;
24              padding: 10px 25px;
25              margin: 15px 0;
26          }
27
28          button {
29              background-color: white;
30              cursor: pointer;
31              padding: 10px 20px;
32              font-weight: bold;
33              border: 1px solid gray;
34              box-shadow: 2px 2px 2px black;
35              margin: 10px;

```

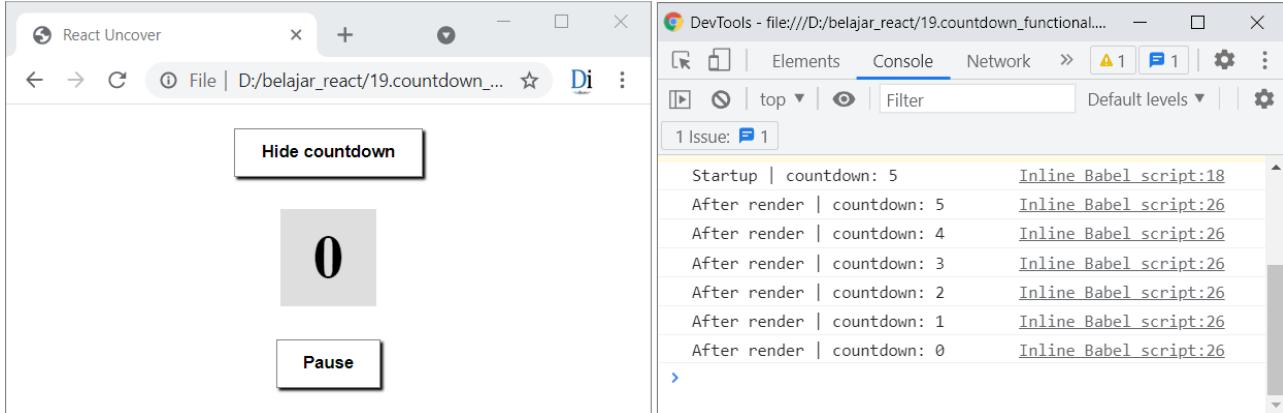
```

36      }
37
38      button:hover {
39          box-shadow: 1px 1px 1px black;
40          background-color: #f1f1f1;
41      }
42  </style>
43 </head>
44
45 <body>
46   <div id="root"></div>
47
48   <script src="js/react.development.js"></script>
49   <script src="js/react-dom.development.js"></script>
50   <script src="js/babel.js"></script>
51   <script type="text/babel">
52
53     const Countdown = () => {
54       const [countdown, setCountdown] = React.useState(5);
55       const [run, setRun] = React.useState(false);
56       const [buttonName, setButtonName] = React.useState("Start");
57
58       const intervalID = React.useRef(null);
59
60       // componentDidMount (startup)
61       React.useEffect(() => {
62         intervalID.current = setInterval(() => {
63           setCountdown((prevState) => (prevState - 1))
64         }, 1000);
65
66         setRun(true);
67         setButtonName("Pause");
68         console.log(`Startup | countdown: ${countdown}`);
69     }, []);
70
71       // componentDidUpdate (after render)
72       React.useEffect(() => {
73         if (countdown === 0) {
74           clearInterval(intervalID.current);
75         }
76         console.log(`After render | countdown: ${countdown}`);
77     });
78
79       // componentWillUnmount
80       React.useEffect(() => {
81         return () => {
82           console.log(`Cleanup | countdown: ${countdown}`);
83
84           clearInterval(intervalID.current);
85         }
86     }, []);
87
88     const handleButtonClick = () => {
89       if (run) {
90         clearInterval(intervalID.current);

```

React Component Lifecycle

```
91         setRun(false);
92         setButtonName("Continue")
93     }
94     else {
95         intervalID.current = setInterval(() => {
96             setCountdown((prevState) => (prevState - 1))
97         }, 1000);
98
99         setRun(true);
100        setButtonName("Pause");
101    }
102}
103
104    return (
105        <div className="container">
106            <h1> {countdown} </h1>
107            <button onClick={handleButtonClick}>
108                {buttonName}
109            </button>
110        </div>
111    )
112
113}
114
115 const MyApp = () => {
116     const [showcountdown, setShowcountdown] = React.useState(false);
117
118     const handleButtonClick = () => {
119         setShowcountdown((prevState) => (!prevState))
120     }
121
122     return (
123         <div className="container">
124             <button onClick={handleButtonClick}>
125                 {showcountdown ? "Hide countdown" : "Show countdown"}
126             </button>
127             {showcountdown && <Countdown />}
128         </div>
129     )
130 }
131
132 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
133
134 </script>
135 </body>
136
137 </html>
```



Gambar: Tampilan Countdown dan tab Console

Di baris awal komponen Countdown, terdapat pendefinisian 3 buah state, yakni `countdown`, `run`, dan `buttonName` (baris 54-56). Ketiganya dipakai untuk menampung nilai hitung mundur dan proses tombol toggle "Start"/"Pause"/"Continue". Cara kerjanya kurang lebih sama seperti di versi class component.

Yang sedikit berbeda ada di baris 58, dimana saya memakai `ref` untuk menyimpan `intervalID`. Di class component, `intervalID` bisa langsung kita simpan sebagai property class, akan tetapi di functional component tidak bisa menggunakan variabel biasa karena nilainya akan ter-reset dalam setiap proses re-render.

Di baris 61-69, terdapat `useEffect()` yang dirancang sebagai pengganti method `componentDidMount()`. Isinya menjalankan fungsi `setInterval()` serta mengupdate state `run` dan `buttonName`. Agar side effect berjalan satu kali di awal saja, array kosong "[]" diinput sebagai argument kedua di baris 69.

`UseEffect` di baris 72-77 berfungsi untuk melakukan pemeriksaan apakah nilai state `countdown` sudah sampai 0 atau belum. Jika hasilnya `true`, fungsi `clearInterval()` akan menghentikan proses hitung mundur agar `countdown` tidak sampai negatif. Side effect ini dirancang sebagai pengganti `componentDidUpdate()` dengan cara tidak mengisi argument kedua dari method `React.useEffect()`.

`UseEffect` ketiga di baris 80-86 berfungsi untuk menghentikan hitung mundur saat komponen `Countdown` terhapus dari struktur DOM. Side effect ini dipakai pengganti `componentWillUnmount()`, sehingga fungsi `clearInterval()` ditulis dalam perintah `return` dari `useEffect` hook.

Sebagai tambahan, proses pengisian dan pengaksesan nilai `intervalID` dilakukan dari `intervalID.current` seperti di baris 62, 74 dan 84. Ini karena di dalam `ref`, nilai variabel tersimpan di dalam property `current`.

Untuk sisa kode program lain saya rasa tidak perlu pembahasan lagi karena hanya hasil konversi dari class component.

Itulah pembahasan kita tentang **React component lifecycle** dan juga **useEffect hook**. Materi ini banyak dipakai untuk project React skala menengah hingga besar. Sebagai contoh, proses pemanggilan API biasanya dilakukan dari dalam method `componentDidMount()`.

Dalam bab berikutnya kita akan masuk ke studi kasus pembuatan aplikasi **Todo**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

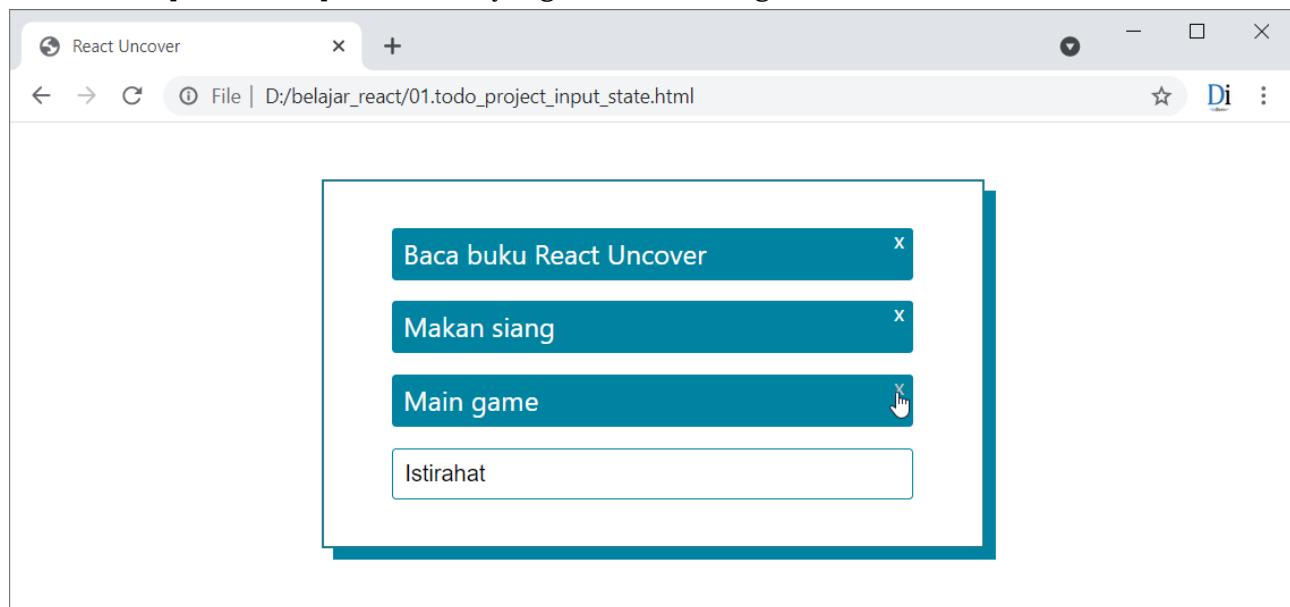
Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

14. Mini Project: Todo

Dalam bab ini kita akan masuk ke studi kasus pembuatan aplikasi **Todo**. Todo adalah sebutan untuk aplikasi yang berisi daftar kegiatan atau hal yang ingin dikerjakan. Setelah kegiatan itu selesai, kita bisa menghapusnya dari daftar todo, kurang lebih sebagai catatan pengingat.

Berikut tampilan dari aplikasi Todo yang akan dirancang:



Gambar: Tampilan aplikasi Todo

Untuk menambah daftar todo, input teks lewat text box di bagian bawah. Dan jika ingin menghapus todo, klik tanda silang (x) di sudut kanan atas setiap list todo.

Seperti biasa, kita akan bahas dengan class component terlebih dahulu, kemudian akan dikonversi menjadi functional component di akhir bab nanti.

14.1. Todo App: Tampilan dan State Awal

Sebagai langkah pertama, saya akan rancang tampilan awal aplikasi. Ini sebenarnya lebih ke materi HTML dan CSS:

01.todo_project_input_state.html

```
1 <!DOCTYPE html>
```

```
2 <html lang="id">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>React Uncover</title>
8   <style>
9     #root {
10       display: flex;
11       justify-content: center;
12       font-size: 1.2rem;
13       font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
14     }
15
16   .container {
17     margin-top: 2rem;
18     padding: 2rem 3rem;
19     box-shadow: 8px 8px 0px #007c9c;
20     border: 2px solid #19748c;
21   }
22
23   input[type="text"] {
24     width: 350px;
25     font-size: 1rem;
26     padding: 8px;
27     outline: 0;
28     border: 1px solid #007c9c;
29     border-radius: 3px 3px;
30   }
31
32   input[type="text"]:focus {
33     border-color: #007c9c;
34     box-shadow: 0 0 3px 4px rgb(0 124 156 / 16%);
35   }
36
37   .todo {
38     background-color: #007c9c;
39     padding: 0.3rem 0.5rem 0.4rem;
40     border-radius: 3px 3px;
41     margin-bottom: 15px;
42     color: white;
43     position: relative;
44   }
45
46   .todo span {
47     background-color: #007c9c;
48     color: #ffffff;
49     position: absolute;
50     right: 0;
51     top: 0;
52     width: 20px;
53     height: 20px;
54     text-align: center;
55     font-size: 16px;
56     cursor: pointer;
```

```
57         border-top-right-radius: 3px 3px;
58         line-height: 16px;
59     }
60
61     .todo span:hover {
62         color: silver;
63     }
64 </style>
65 </head>
66
67 <body>
68     <div id="root"></div>
69
70     <script src="js/react.development.js"></script>
71     <script src="js/react-dom.development.js"></script>
72     <script src="js/babel.js"></script>
73     <script type="text/babel">
74
75     class MyApp extends React.Component {
76
77         constructor(props) {
78             super(props);
79             this.state = { inputTodo: "" }
80         }
81
82         handleInputChange = (e) => {
83             this.setState({ inputTodo: e.target.value })
84         }
85
86         render() {
87             console.log(this.state.inputTodo);
88             return (
89                 <div className="container">
90
91                     <div className="todo">Baca buku React Uncover<span>x</span></div>
92                     <div className="todo">Makan siang<span>x</span></div>
93                     <div className="todo">Main game<span>x</span></div>
94
95                     <form>
96                         <div>
97                             <input type="text" placeholder="Add todo..." 
98                                 value={this.state.inputTodo}
99                                 onChange={this.handleInputChange}
100                                />
101                         </div>
102                     </form>
103
104                 </div >
105             )
106         }
107
108     }
109
110     ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
111
```

```
112  </script>
113 </body>
114
115 </html>
```

Kode CSS yang ada sebenarnya cukup standar, saya menggunakan flexbox untuk mengatur posisi list todo, mengubah warna teks dan warna background, serta memposisikan huruf (x) sebagai tombol untuk menghapus todo.

Masuk ke kode React, di dalam method render() terdapat tag `<div className="container">` sebagai container terluar. Di dalamnya ada 3 tag `<div className="todo">` yang akan menjadi list todo (baris 91-93).

Saat ini list todo masih ditulis manual, nantinya list ini akan kita generate berdasarkan data state. Tambahan tag `x` di akhir setiap todo dipakai untuk membuat tombol hapus, yang saat ini juga belum berfungsi.

Berikutnya terdapat tag `<form>` yang berisi 1 tag `<input type="text">`. Pasangan atribut `value={this.state.inputTodo}` dan `onChange={this.handleInputChange}` dipakai untuk mengakses nilai form (sebagai controller components).

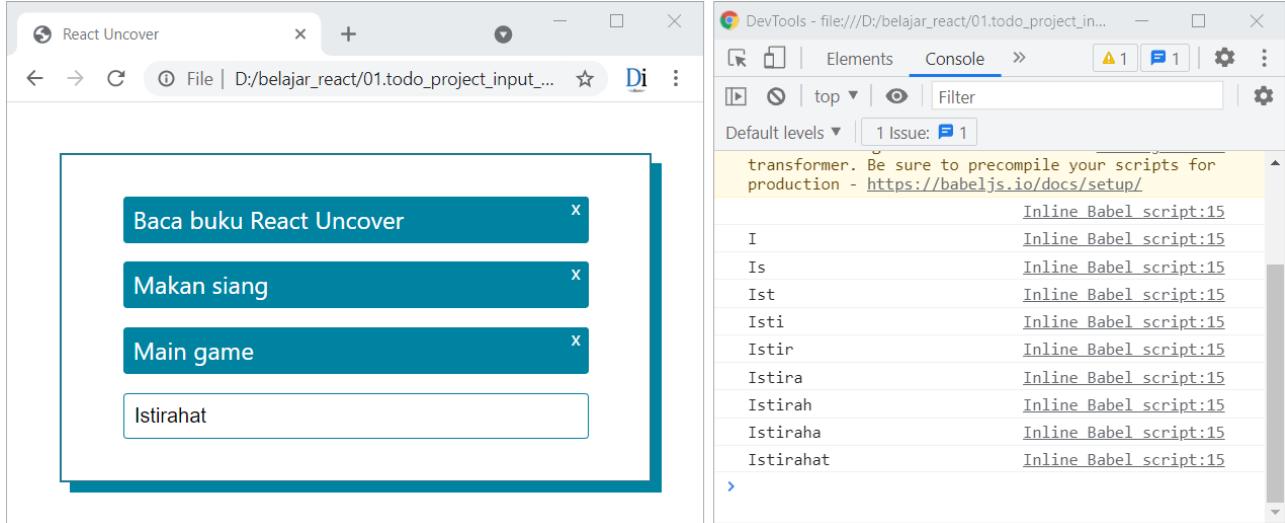
Saya tidak menambah tombol submit ke dalam form karena web browser memiliki perilaku default dimana ketika kita menekan tombol enter saat mengetik inputan, itu secara otomatis akan men-submit form. Ini lebih ke pilihan desain saja, jika anda ingin menambah tombol submit juga tidak masalah.

Masuk ke dalam constructor komponen MyApp, di baris 79 terdapat pendefinisian state `inputTodo`. State ini akan dipakai untuk menampung nilai inputan form.

Method `handleInputChange()` di baris 32-35 berfungsi untuk mengambil nilai form, yakni sebagai event handler dari tag `<input>` sebelumnya.

Untuk memastikan inputan form sudah diterima state, saya menambah `console.log(this.state.inputTodo)` sebelum perintah `return` di baris 87.

Silahkan jalankan kode di atas, dan pastikan isi state `inputTodo` sudah tampil di tab Console web browser:



Gambar: Inputan form sudah berhasil di akses

Begitu inputan form di ketik, akan tampil teks di tab Console sebagai hasil dari perintah `console.log()` dari dalam method `render()`.

14.2. Todo App: Validasi Inputan Todo

Nilai form sudah berhasil di akses, sekarang kita lanjut membuat validasi dan menampilkan pesan error. Syarat validasi untuk inputan todo hanya ada 1, yakni tidak boleh kosong. Jika user menginput todo kosong, tampilkan teks "Todo tidak boleh kosong". Berikut modifikasi dari komponen MyApp:

```
02.todo_project_form_validation.html

1  class MyApp extends React.Component {
2      constructor(props) {
3          super(props);
4          this.state = {
5              inputTodo: "",
6              pesanErrors: null
7          }
8      }
9
10     handleInputChange = (e) => {
11         this.setState({ inputTodo: e.target.value })
12     }
13
14     handleFormSubmit = (e) => {
15         e.preventDefault();
16         if (this.state.inputTodo.trim() === "") {
17             this.setState({ pesanErrors: "Todo tidak boleh kosong" });
18         }
19     }
20
21     render() {
```

```
22     return (
23       <div className="container">
24         <div className="todo">Baca buku React Uncover<span>x</span></div>
25         <div className="todo">Makan siang<span>x</span></div>
26         <div className="todo">Main game<span>x</span></div>
27
28       <form onSubmit={this.handleFormSubmit}>
29         <div>
30           <input type="text" placeholder="Add todo..." value={this.state.inputTodo} onChange={this.handleInputChange}>
31           />
32           {this.state.pesanErrors && <small>{this.state.pesanErrors}</small>}
33         </div>
34       </form>
35     </div >
36   )
37 }
38 )
39 }
40 }
41 }
42 }
43 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Untuk menyimpan pesan error, saya menambah state `pesanErrors` ke dalam constructor.

Proses validasi akan berjalan ketika form di submit, untuk membuatnya perlu tambahan atribut `onSubmit={this.handleFormSubmit}` ke dalam tag `<form>` di baris 28.

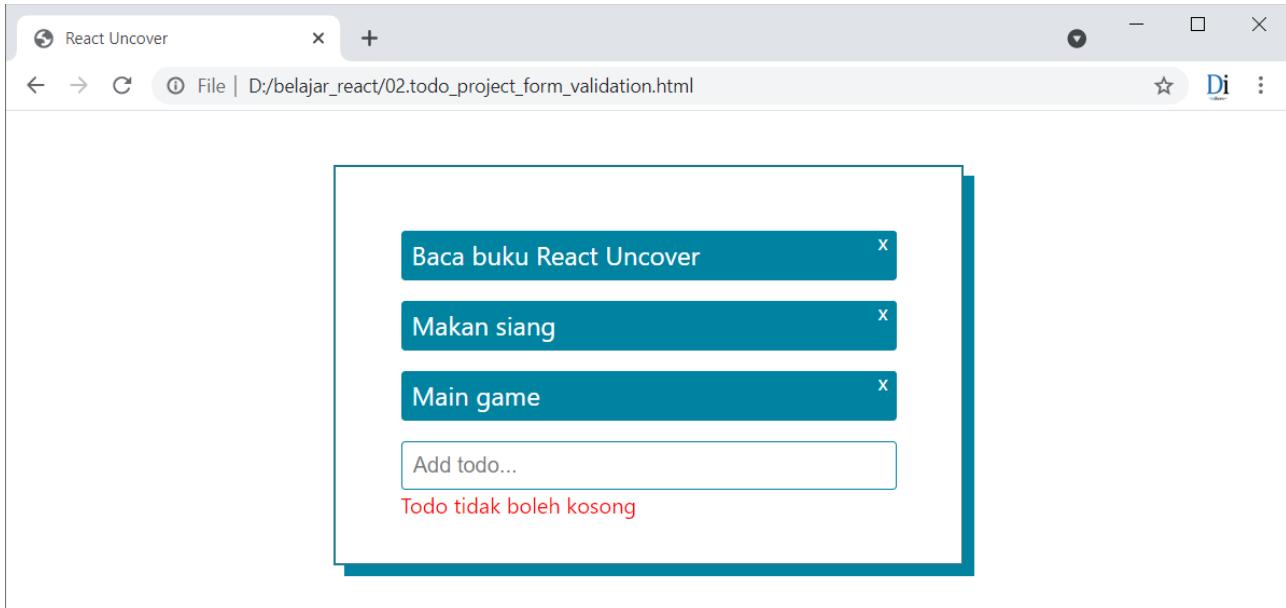
Pendefinisian method `handleFormSubmit()` ada di baris 14-19. Perintah pertama berupa `e.preventDefault()` untuk mencegah form dikirim ke server, kemudian state `inputTodo` akan diperiksa apakah berisi string kosong atau tidak. Jika iya, isi state `pesanErrors` dengan string "Todo tidak boleh kosong".

Pesan error ini selanjutnya ditampilkan di baris 34, yakni di bawah tag `<input>`. Operator `&&` berguna agar pesan error hanya tampil jika state `pesanErrors` berisi sesuatu.

Untuk kode CSS, bisa ditambahkan kode berikut agar teks pesan error tampil dengan warna merah:

```
1 small {
2   display: block;
3   color: red;
4 }
```

Silahkan tes dengan langsung menekan tombol enter tanpa menginput teks apapun ke dalam inputan todo:



Gambar: Pesan error validasi todo

14.3. Todo App: List Awal Todo

Saat ini list todo masih kita tulis manual ke dalam kode JSX. Agar bisa dinamis, data tersebut harus di pindah ke dalam state. Berikut perubahan kode programnya:

```
03.todo_project_initial_todo.html

1 const todos =
2 [
3     { id: "01", text: "Baca buku React Uncover" },
4     { id: "02", text: "Makan siang" },
5     { id: "03", text: "Main game" }
6 ];
7
8 class MyApp extends React.Component {
9     constructor(props) {
10         super(props);
11         this.state = {
12             inputTodo: "",
13             pesanErrors: null,
14             arrayTodo: todos,
15         }
16     }
17
18     handleInputChange = (e) => {
19         this.setState({ inputTodo: e.target.value })
20     }
21
22     handleFormSubmit = (e) => {
23         e.preventDefault();
24         if (this.state.inputTodo.trim() === "") {
```

```

25     this.setState({ pesanErrors: "Todo tidak boleh kosong" });
26   }
27 }
28
29 render() {
30   return (
31     <div className="container">
32       {
33         this.state.arrayTodo.map((todo) =>
34           <div className="todo" key={todo.id}>{todo.text}<span>x</span></div>
35         )
36       }
37     <form onSubmit={this.handleFormSubmit}>
38       <div>
39         <input type="text" placeholder="Add todo..." value={this.state.inputTodo} onChange={this.handleInputChange}>
40         />
41         {this.state.pesanErrors && <small>{this.state.pesanErrors}</small>}
42       </div>
43     </form>
44   </div >
45 )
46 }
47 )
48 }
49
50 }
51
52 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Di baris 1-6 terdapat konstanta todos yang berisi array. Element array ini berbentuk object yang terdiri dari pasangan property id dan text, dimana **id** akan menjadi penanda (nilai unik) untuk setiap todo, sedangkan **text** akan berisi pesan todo. Array todos kemudian diinput sebagai nilai awal dari state arrayTodo di baris 14.

Data yang tersimpan di dalam state todos selanjutnya "dibuka" dengan perulangan `this.state.arrayTodo.map()` di baris 33-35. Inilah cara meng-generate list todo menjadi kode JSX, property `todo.id` dipakai untuk nilai atribut key, dan property `todo.text` berada di antara tag pembuka dan penutup `<div className="todo">`. Tidak lupa tambahan tag `x` untuk tombol hapus.

Hasil akhir kode program ini sama persis seperti contoh sebelumnya, kita hanya memindahkan data todo ke dalam state arrayTodo.

14.4. Todo App: Menambah Todo

Karena data todo sudah ada di dalam state, kita akan bisa buat kode program untuk menambah todo baru. Caranya, tambah 1 object baru ke dalam `arrayTodo` pada saat form di submit. Perintah tersebut perlu dibuat ke dalam method `handleFormSubmit()`:

04.todo_project_add_todo.html

```
1 ...  
2     handleFormSubmit = (e) => {  
3         e.preventDefault();  
4         if (this.state.inputTodo.trim() === "") {  
5             this.setState({ pesanErrors: "Todo tidak boleh kosong" });  
6         }  
7         else {  
8             // Input todo baru ke dalam state  
9             const newTodos = [  
10                 ...this.state.arrayTodo,  
11                 {  
12                     id: new Date().getTime().toString(),  
13                     text: this.state.inputTodo  
14                 }  
15             ];  
16             this.setState({ arrayTodo: newTodos });  
17         }  
18         // kosongkan input text  
19         this.state.inputTodo = "";  
20     }  
21 }  
22 ...
```

Di dalam method `handleFormSubmit()` saat ini sudah ada validasi inputan form dengan kondisi `if`. Penambahan todo hanya bisa kita lakukan jika validasi ini lolos, maka tinggal sambung dengan blok `else` antara baris 7-20.

Di baris 9 saya membuat konstanta `newTodos` untuk menampung semua isi state `arrayTodo` saat ini (diambil dengan bantuan `spread operator`), lalu ditambah dengan object todo baru.

Nilai property `id` untuk object todo baru di-generate berdasarkan tanggal saat ini, atau tepatnya `timestamp` dengan perintah `Date().getTime().toString()` seperti di baris 12.

Di dalam JavaScript, method `Date().getTime()` akan menghasilkan `timestamp` dengan ketelitian hingga milidetik. Ini bisa kita pakai untuk menggenerate sebuah angka unik, meskipun tetap ada kemungkinan nilai ganda jika beberapa todo diinput dengan sangat cepat. Di aplikasi sebenarnya, nilai `id` akan lebih aman jika di generate dari database, misalnya menggunakan kolom `auto_increment`.

Tambahan method `.toString()` berguna untuk mengkonversi angka `timestamp` menjadi string. React sebenarnya tidak mensyaratkan nomor `id` atau nilai atribut `key` harus berbentuk string, tapi boleh juga berbentuk angka selama nilainya unik.

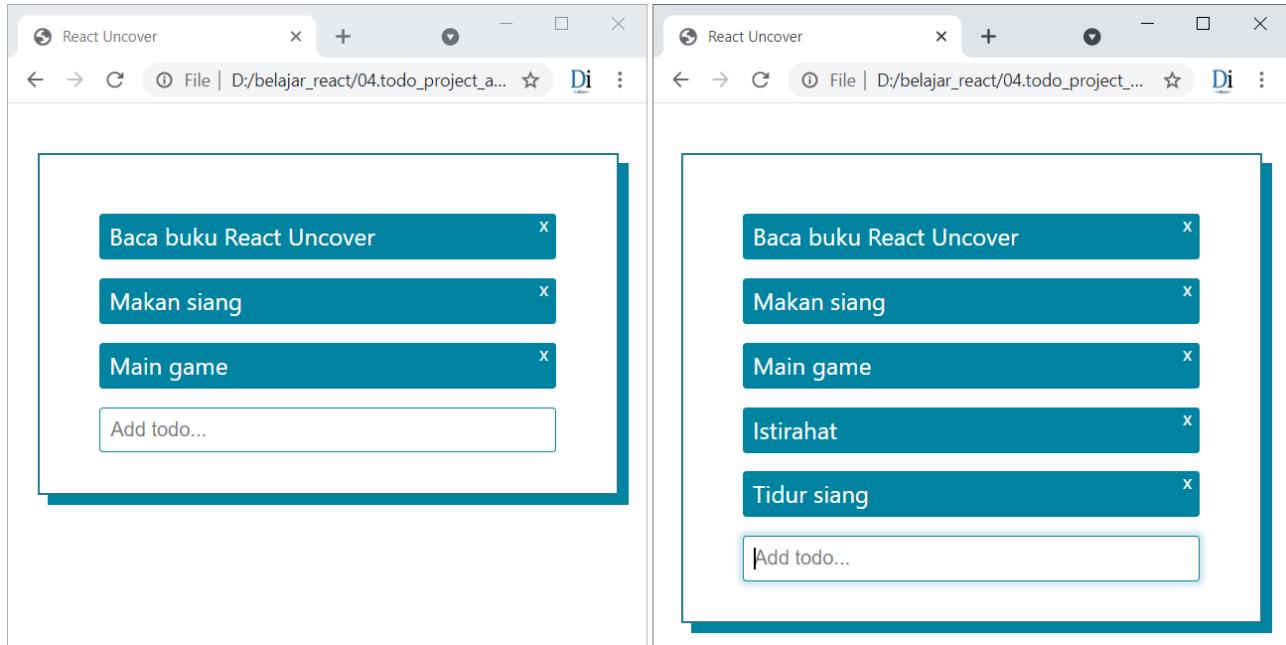
Untuk property `text` object todo baru, nilainya diambil dari hasil inputan form yang sudah tersimpan di dalam state `inputTodo` (baris 13).

Setelah object todo selesai dibuat, nilainya kemudian di update ke dalam state `arrayTodo` dengan perintah `this.setState({ arrayTodo: newTodos })` seperti di baris 16.

Mini Project: Todo

Terakhir di baris 19, inputan form di kosongkan kembali dengan cara mengisi string kosong ke dalam state `inputTodo`. Sebagai pengingat, state `inputTodo` terhubung 2 arah dengan tag `<input>`, sehingga jika kita menghapus nilainya, inputan form juga akan kosong.

Silahkan tes hasil penambahan kode ini. Pada saat tombol enter di tekan, isi inputan teks akan menjadi sebuah todo baru:



Gambar: Penambahan todo baru

Berikut isi `console.log(this.state.arrayTodo)` setelah penambahan 2 buah todo di atas:



Gambar: Isi state arrayTodo

Nilai `id` untuk todo ke-4 dan ke-5 didapat dari `Date().getTime().toString()` dan akan berbeda setiap milidetik.

14.5. Todo App: Menghapus Todo

Di dalam kode JSX, sudah terdapat tanda silang (x) di sudut kanan atas setiap list. Idenya adalah, ketika tanda ini di klik, hapus object todo yang ada di posisi tersebut dari dalam

arrayTodo.

Namun kita perlu memikirkan bagaimana caranya agar tanda silang ini terhubung dengan object todo yang ingin dihapus. Salah satu solusi bisa dengan menyisipkan atribut bantu yang berisi `id` dari object todo tersebut.

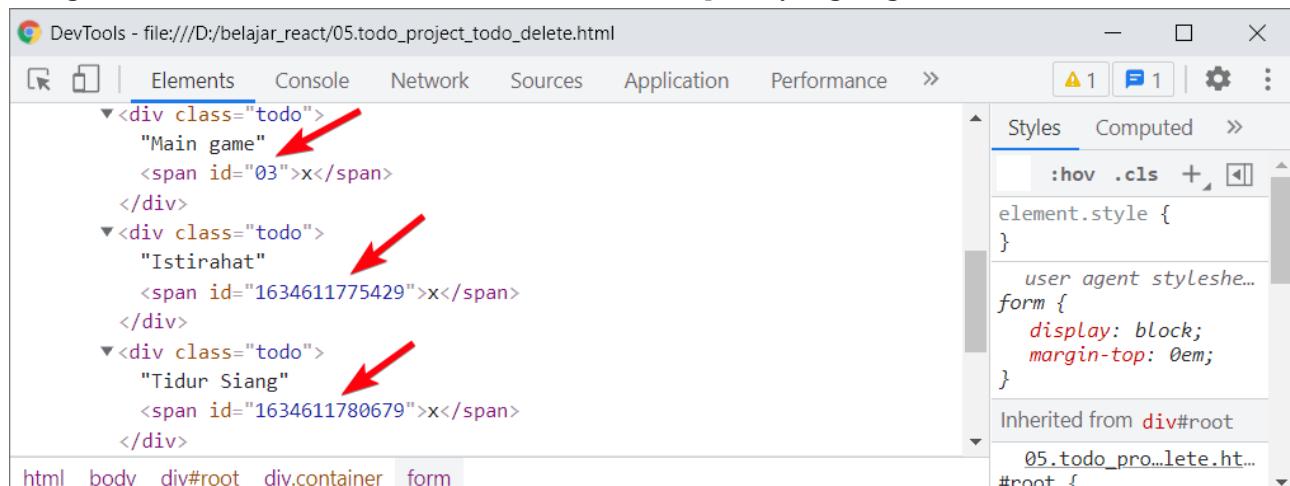
Silahkan modifikasi proses generate list todo menjadi sebagai berikut:

05.todo_project_todo_delete.html

```
1 ...  
2   this.state.arrayTodo.map((todo) =>  
3     <div className="todo" key={todo.id}>{todo.text}  
4       <span id={todo.id} onClick={this.handleDeleteClick}>x</span>  
5     </div>  
6   )  
7 ...
```

Ke dalam tag `` di baris 4 saya menambah atribut `id` yang nilainya diambil dari `todo.id`. Lalu tambahan atribut `onClick` dipakai untuk menjalankan method `handleDeleteClick()`. Sesampainya di dalam method `handleDeleteClick()`, kita akan coba akses nilai atribut `id` ini.

Dengan tambahan kode di atas, berikut contoh tampilan yang di generate web browser:



Gambar: Tambahan atribut id ke dalam tag ``

Dan berikut isi dari method `handleDeleteClick()` yang akan dijalankan saat tanda (x) di klik:

05.todo_project_todo_delete.html

```
1 ...  
2   handleDeleteClick = (e) => {  
3     // console.log(e.target.id);  
4     const newTodos = this.state.arrayTodo.filter(  
5       item => item.id !== e.target.id  
6     );  
7     this.setState({ arrayTodo: newTodos });  
8 }
```

9 ...

Dalam method ini, `e` menjadi parameter yang menampung `event object`. Property `e.target.id` bisa dipakai untuk mengakses atribut `id` yang baru saja kita generate. Untuk memastikan nilainya sesuai, silahkan hapus tanda komentar untuk perintah `console.log(e.target.id)`, pastikan nilainya sama dengan id todo yang ingin dihapus.

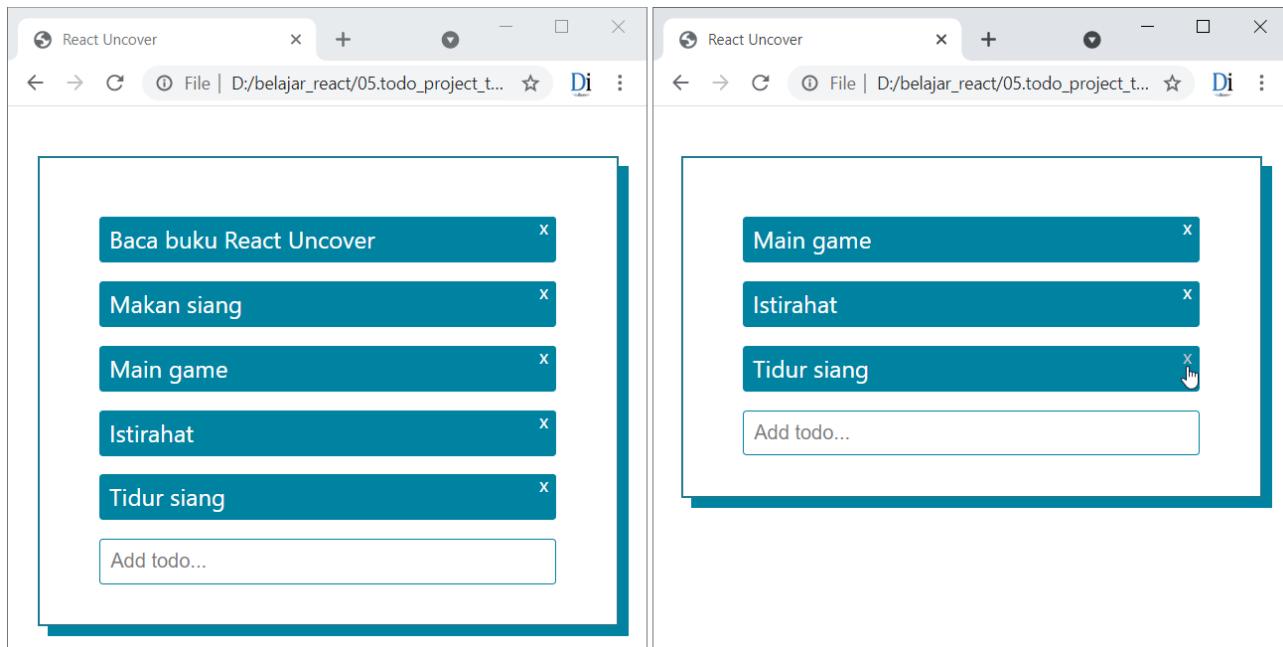
Dengan data dari `e.target.id`, kita mengetahui id dari object todo yang akan dihapus. Untuk menghapus object dengan id tersebut dari dalam `arrayTodo`, bisa minta bantuan method `filter()` bawaan JavaScript.

Method `filter()` ini mirip seperti `map()`, dalam artian akan memproses semua isi array. Bedanya, `filter()` mengembalikan array baru selama fungsi `callback` yang diinput sebagai argument menghasilkan nilai `true`.

Di baris 4, konstanta `newTodos` akan berisi array baru dari semua object yang tersimpan di `arrayTodo`, selama id object tersebut tidak sama dengan `e.target.id`. Terakhir, kita tinggal mengupdate nilai state `arrayTodo` dengan `newTodos` seperti di baris 7.

Jika butuh penjelasan lebih dalam tentang cara penggunaan method `filter()`, bisa ke developer.mozilla.org²¹, atau juga tersedia di buku **JavaScript Uncover**.

Dengan penambahan ini, tombol (x) sudah bisa dipakai untuk menghapus todo:



Gambar: Menghapus list todo

21. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

14.6. Todo App: Membuat Todo Component

Aplikasi todo sebenarnya sudah selesai, namun saya ingin memecah kode yang ada menjadi komponen-komponen yang lebih kecil. Ini bisa menjadi latihan bagaimana membuat aplikasi dengan beberapa komponen yang saling berinteraksi satu sama lain.

Komponen pertama yang akan saya pisahkan adalah **Todo component**. Komponen ini dipakai untuk menampilkan kode JSX dari sebuah list todo. Berikut modifikasi yang diperlukan:

06.todo_project_todo_component.html

```

1  const todos =
2    [
3      { id: "01", text: "Baca buku React Uncover" },
4      { id: "02", text: "Makan siang" },
5      { id: "03", text: "Main game" }
6    ];
7
8  class Todo extends React.Component {
9    render() {
10      return (
11        <div className="todo">{this.props.text}
12          <span id={this.props.id} onClick={this.props.onTodoClick}>x</span>
13        </div>
14      )
15    }
16  }
17
18 class MyApp extends React.Component {
19   constructor(props) {
20     super(props);
21     this.state = {
22       inputTodo: "",
23       pesanErrors: null,
24       arrayTodo: todos,
25     }
26   }
27
28   handleInputChange = (e) => {
29     this.setState({ inputTodo: e.target.value })
30   }
31   handleDeleteClick = (e) => {
32     const newTodos = this.state.arrayTodo.filter(
33       item => item.id !== e.target.id
34     );
35     this.setState({ arrayTodo: newTodos });
36   }
37
38   handleFormSubmit = (e) => {
39     e.preventDefault();
40     if (this.state.inputTodo.trim() === "") {
41       this.setState({ pesanErrors: "Todo tidak boleh kosong" });
42     }

```

```

43     else {
44         // Input todo baru ke dalam state
45         const newTodos = [
46             ...this.state.arrayTodo,
47             {
48                 id: new Date().getTime().toString(),
49                 text: this.state.inputTodo
50             }
51         ];
52         this.setState({ arrayTodo: newTodos });
53
54         // kosongkan input text
55         this.state.inputTodo = "";
56     }
57 }
58
59 render() {
60     return (
61         <div className="container">
62             {
63                 this.state.arrayTodo.map((todo) => (
64                     <Todo
65                         key={todo.id} // key ditulis disini, bukan di <Todo> component
66                         id={todo.id}
67                         text={todo.text}
68                         onTodoClick={this.handleDeleteClick}
69                     />
70                 ))
71             }
72         <form onSubmit={this.handleFormSubmit}>
73             <div>
74                 <input type="text" placeholder="Add todo..." value={this.state.inputTodo} onChange={this.handleInputChange}>
75                 />
76                 {this.state.pesanErrors && <small>{this.state.pesanErrors}</small>}
77             </div>
78         </form>
79     </div >
80 )
81 )
82 )
83 }
84
85 }
86
87 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);

```

Di baris 8-16 terdapat pendefinisian komponen Todo yang untuk sementara akan kita lompati terlebih dahulu.

Masuk ke baris 64-68, sebelumnya di bagian ini terdapat kode JSX untuk menampilkan list todo, akan tetapi sekarang saya ganti dengan komponen <Todo>. Dengan perubahan ini, nilai setiap object di dalam state arrayTodo akan dikirim ke dalam komponen Todo sebagai props, yang terdiri dari `id={todo.id}`, `text={todo.text}` dan `onTodoClick={this.handleDelete}`

Click}. Di dalam komponen Todo-lah data-data ini akan diproses.

Sebagai tambahan, atribut key tetap ditulis di sini, tidak perlu ikut dikirim ke dalam komponen Todo. Tipsnya, dimana ada perulangan `map()`, disitulah atribut key perlu ditulis.

Masuk ke pendefinisian komponen Todo di baris 8-16, isinya langsung mengembalikan kode JSX dari dalam method `render()`. Setiap `props` ditempatkan ke dalam tag `<div>` sesuai fungsinya masing-masing, yang kurang lebih sama seperti struktur JSX di contoh sebelumnya.

Yang butuh sedikit pembahasan ada di atribut `onClick={this.props.onTodoClick}` di dalam tag ``. Ini adalah penerapan dari teknik "state up" yang pernah kita bahas di bab tentang React state. Jika tanda tag (x) di dalam tag `` di klik, itu perlu mengakses state `arrayTodo` untuk menghapus satu object di dalamnya, akan tetapi state `arrayTodo` berada di dalam komponen `MyApp`, bukan di dalam komponen Todo.

Maka yang menjadi nilai dari atribut `onClick` adalah event handler yang dikirim sebagai `props`, yakni `this.props.onTodoClick`. Pada saat event ini dikirim di baris 68, atribut `onTodoClick` merujuk ke method `this.handleDeleteClick()`, maka inilah yang akan dijalankan ketika tombol (x) di klik.

14.7. Todo App: Membuat TodoForm Component

Kode lain yang ingin saya pisah adalah form input todo ke dalam **TodoForm component**.

Untuk kode JSX tidak ada masalah, kita cukup memindahkan semua isi tag `<form>`. Yang menjadi kendala ada di isi method `handleFormSubmit()`. Terdapat 2 alternatif, apakah meninggalkan method ini di dalam komponen `MyApp`, atau membawanya ke dalam komponen `TodoForm`.

Saya memilih alternatif kedua, yakni ikut memindahkan isi method `handleFormSubmit()` ke dalam komponen `TodoForm`. Namun di dalam method ini ada kode yang butuh mengakses state `arrayTodo`, tepatnya saat penambahan todo baru.

Karena kita ingin menerapkan "single source of truth", state `TodoForm` tidak bisa di pindah. Yang bisa dilakukan adalah membuat method baru yang nantinya dikirim sebagai `props`.

Sebelumnya, isi method `handleFormSubmit()` saya tulis sebagai berikut:

06.todo_project_todo_component.html

```
1 handleFormSubmit = (e) => {
2   e.preventDefault();
3   if (this.state.inputTodo.trim() === "") {
4     this.setState({ pesanErrors: "Todo tidak boleh kosong" });
5   }
6   else {
7     // Input todo baru ke dalam state
8     const newTodos = [
```

Mini Project: Todo

```
9      ...this.state.arrayTodo,
10     {
11       id: new Date().getTime().toString(),
12       text: this.state.inputTodo
13     }
14   ];
15   this.setState({ arrayTodo: newTodos });
16
17 // kosongkan input text
18 this.state.inputTodo = "";
19 }
20 }
```

Sekarang akan dipecah menjadi:

07.todo_project_todoform_component_1.html

```
1 addTodo(text) {
2   const newTodos = [
3     ...this.state.arrayTodo,
4     {
5       id: new Date().getTime().toString(),
6       text: text
7     }
8   ];
9   this.setState({ arrayTodo: newTodos });
10 }
11
12 handleFormSubmit = (e) => {
13   e.preventDefault();
14   if (this.state.inputTodo.trim() === "") {
15     this.setState({ pesanErrors: "Todo tidak boleh kosong" });
16   }
17   else {
18     // Input todo baru ke dalam state
19     this.addTodo(this.state.inputTodo);
20     // kosongkan input text
21     this.state.inputTodo = "";
22   }
23 }
```

Bedanya, dalam kode ini proses input ke dalam `arrayTodo` dilakukan oleh method `addTodo()`. Method `addTodo()` menerima 1 argument berupa teks todo baru. Di dalam method `handleFormSubmit()`, pemanggilan method ini dilakukan di baris 19 dengan perintah `this.addTodo(this.state.inputTodo)`.

Dengan cara ini, nantinya method `addTodo()` bisa dikirim sebagai *props* dan diakses dari dalam method `handleFormSubmit()` yang ada di dalam komponen `TodoForm`. Berikut kode program lengkap dengan perubahan ini:

08.todo_project_todoform_component_2.html

```
1 const todos =
2   [
3     { id: "01", text: "Baca buku React Uncover" },
4     { id: "02", text: "Makan siang" },
5     { id: "03", text: "Main game" }
6   ];
7
8 class Todo extends React.Component {
9   render() {
10     return (
11       <div className="todo">{this.props.text}
12         <span id={this.props.id} onClick={this.props.onTodoClick}>x</span>
13       </div>
14     )
15   }
16 }
17
18 class TodoForm extends React.Component {
19   constructor(props) {
20     super(props);
21     this.state = {
22       inputTodo: "",
23       pesanErrors: null,
24     }
25   }
26
27   handleInputChange = (e) => {
28     this.setState({ inputTodo: e.target.value })
29   }
30
31   handleFormSubmit = (e) => {
32     e.preventDefault();
33     if (this.state.inputTodo.trim() === "") {
34       this.setState({ pesanErrors: "Todo tidak boleh kosong" });
35     }
36     else {
37       // Input todo baru ke dalam state
38       this.props.onAddTodo(this.state.inputTodo);
39       // kosongkan input text
40       this.state.inputTodo = "";
41     }
42   }
43
44   render() {
45     return (
46       <form onSubmit={this.handleFormSubmit}>
47         <div>
48           <input type="text" placeholder="Add todo..." value={this.state.inputTodo} onChange={this.handleInputChange}>
49           />
50           {this.state.pesanErrors && <small>{this.state.pesanErrors}</small>}
51         </div>
52       </form>
53     )
54   }
55 }
```

Mini Project: Todo

```
54      </form>
55    )
56  }
57 }
58
59 class MyApp extends React.Component {
60   constructor(props) {
61     super(props);
62     this.state = {
63       arrayTodo: todos,
64     }
65   }
66
67   handleDeleteClick = (e) => {
68     const newTodos = this.state.arrayTodo.filter(
69       item => item.id !== e.target.id
70     );
71     this.setState({ arrayTodo: newTodos });
72   }
73
74   handleAddTodo = (text) => {
75     const newTodos = [
76       ...this.state.arrayTodo,
77       {
78         id: new Date().getTime().toString(),
79         text: text
80       }
81     ];
82     this.setState({ arrayTodo: newTodos });
83   }
84
85   render() {
86     return (
87       <div className="container">
88         {
89           this.state.arrayTodo.map((todo) => (
90             <Todo
91               key={todo.id}
92               id={todo.id}
93               text={todo.text}
94               onClick={this.handleDeleteClick}
95             />
96           ))
97         }
98         <TodoForm onAddTodo={this.handleAddTodo} />
99       </div >
100     )
101   }
102
103 }
104
105 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Di baris 98, terdapat perintah JSX untuk mengakses tag `<TodoForm>` dengan tambahan atribut

onAddTodo={this.handleAddTodo}. Nilai atribut ini tidak lain merujuk ke method handleAddTodo() di baris 74-83. Ini merupakan method penambah todo yang kita pisah sebelumnya. Apapun teks yang dikirim sebagai parameter, itu akan menjadi sebuah todo baru.

Masuk ke komponen TodoForm di baris 18-57, mayoritas kode berasal dari hasil "cut" tag <form> yang tadinya ada di komponen MyApp, termasuk juga memindahkan state inputTodo dan pesanErrors karena keduanya dipakai untuk form saja, ini berbeda dengan state arrayTodo yang harus tetap tinggal di komponen MyApp.

Ketika form di submit, perintah untuk menambahkan todo baru ada di baris 38, yakni this.props.onAddTodo(this.state.inputTodo). Di sini kita mengakses method handleAddTodo() milik MyApp melalui props.onAddTodo yang dikirim sebagai atribut.

Inilah aplikasi todo yang terdiri dari 3 komponen: **MyApp** sebagai komponen utama, **Todo** untuk menampilkan list todo, dan **TodoForm** untuk pemrosesan inputan form. Silahkan pelajari sejenak bagaimana ketiga komponen inti saling berinteraksi satu sama lain.

14.8. Todo App: Versi Functional Component

Bagian ini lebih cocok sebagai latihan, yakni mengubah kode program kita dari bentuk class component menjadi functional component. Silahkan coba sebentar.

Baik, berikut aplikasi Todo dalam bentuk functional component:

```
1 const todos =
2   [
3     { id: "01", text: "Baca buku React Uncover" },
4     { id: "02", text: "Makan siang" },
5     { id: "03", text: "Main game" }
6   ];
7
8 const Todo = (props) => {
9   return (
10     <div className="todo">{props.text}
11       <span id={props.id} onClick={props.onTodoClick}>x</span>
12     </div>
13   )
14 }
15
16 const TodoForm = (props) => {
17   const [inputTodo, setInputTodo] = React.useState("");
18   const [pesanErrors, setpesanErrors] = React.useState(null);
19
20   const handleInputChange = (e) => {
21     setInputTodo(e.target.value)
22   }
23
24   const handleFormSubmit = (e) => {
25     e.preventDefault();
26     if (inputTodo.trim() === "") {
```

```
27     setpesanErrors("Todo tidak boleh kosong");
28 }
29 else {
30     // Input todo baru ke dalam state
31     props.onAddTodo(inputTodo);
32     // kosongkan input text
33     setInputTodo("");
34 }
35 }
36
37 return (
38     <form onSubmit={handleFormSubmit}>
39         <div>
40             <input type="text" placeholder="Add todo..." 
41                 value={inputTodo}
42                 onChange={handleInputChange}
43             />
44             {pesanErrors && <small>{pesanErrors}</small>}
45         </div>
46     </form>
47 )
48 }
49
50 const MyApp = () => {
51     const [arrayTodo, setArrayTodo] = React.useState(todos);
52
53     const handleDeleteClick = (e) => {
54         const newTodos = arrayTodo.filter(
55             item => item.id !== e.target.id
56         );
57         setArrayTodo(newTodos);
58     }
59
60     const handleAddTodo = (text) => {
61         const newTodos = [
62             ...arrayTodo,
63             {
64                 id: new Date().getTime().toString(),
65                 text: text
66             }
67         ];
68         setArrayTodo(newTodos);
69     }
70
71     return (
72         <div className="container">
73             {
74                 arrayTodo.map((todo) => (
75                     <Todo
76                         key={todo.id}
77                         id={todo.id}
78                         text={todo.text}
79                         onClick={handleDeleteClick}
80                     />
81                 )));
82     );
83 }
```

```
82      }
83      <TodoForm onAddTodo={handleAddTodo} />
84    </div >
85  )
86 }
87
88 ReactDOM.createRoot(document.getElementById('root')).render(<MyApp />);
```

Tidak ada sesuatu yang baru di sini, proses konversi cukup dengan menukar cara pembuatan class menjadi function, menambah useState hook, serta penyesuaian props. Sisanya sama seperti di class component.

Dalam bab ini kita telah membuat aplikasi Todo yang cukup sering jadi ajang latihan pemahaman React. Berikutnya, kita akan lanjut pelajari cara membuat project React menggunakan "Create react app".

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

15. Create React App

Semua kode program React yang kita buat sejak awal buku hanya ditulis ke dalam tag `<script>` di sebuah file HTML. Meskipun ini sangat praktis, tapi bukanlah sebuah cara ideal dalam membuat aplikasi React. Dalam bab ini kita akan bahas cara membuat kode React menggunakan "Create react app".

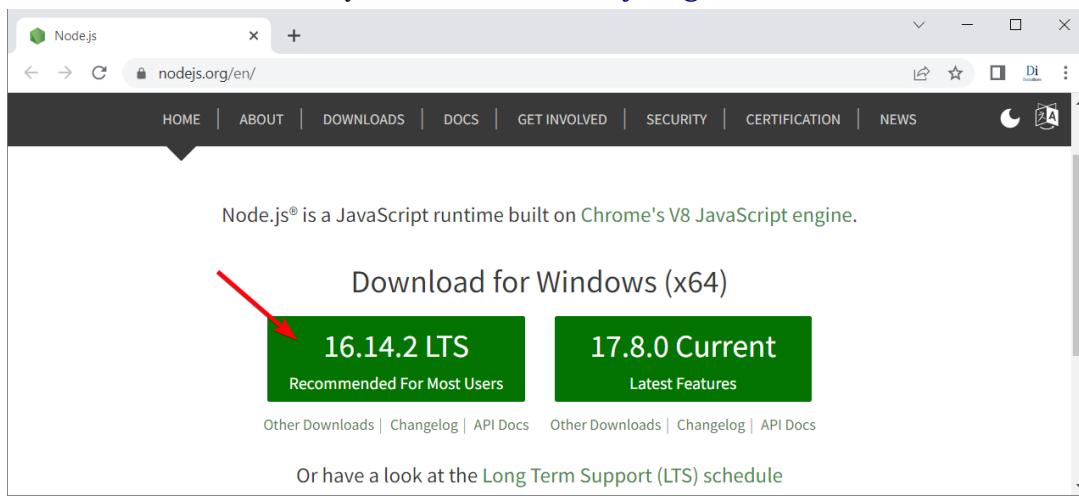
15.1. Menginstall Create React App

Create React App adalah sebuah lingkungan pengembangan (*development environment*) untuk menulis kode React. Ini bukanlah aplikasi terpisah, tapi hanya kumpulan tools dan struktur folder yang sudah disediakan tim React. Tujuannya agar proses pembuatan aplikasi bisa lebih efisien dan mudah dikelola.

Untuk bisa menggunakan *create react app*, kita harus menginstall Node.js terlebih dahulu. Karena pada dasarnya *create react app* merupakan sebuah *package* dari Node.js. Versi yang dibutuhkan minimal Node.js 10 ke atas.

Node.js sendiri merupakan *platform* atau teknologi untuk membuat JavaScript bisa berjalan di web server, kurang lebih bisa menggantikan peran PHP. Namun di sini kita tidak akan membahas Node.js lebih lanjut (itu bisa jadi 1 buku tersendiri).

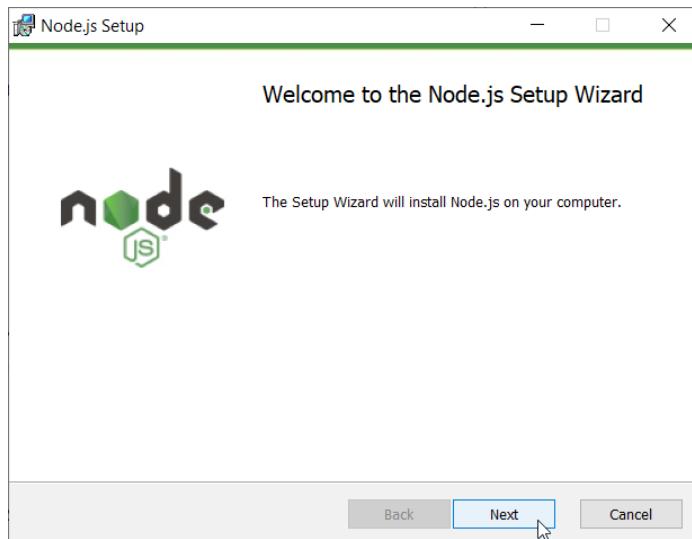
Node.js diperlukan hanya untuk bisa mengakses perintah **npx** (node package execution) yang dipakai saat menginstall *create react app*. Jika di komputer anda belum terinstall Node.js, silahkan download file installernya dari website nodejs.org.



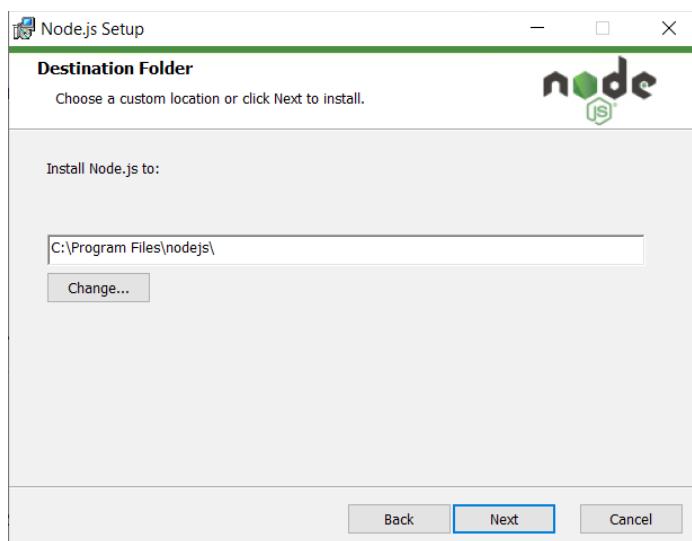
Gambar: Tampilan web nodejs.org

Web nodejs.org bisa mendeteksi sistem operasi yang digunakan, karena saya memakai Windows 10 64-bit, langsung diarahkan untuk memilih salah satu dari versi 16.14.2 atau 17.8.0 untuk OS Windows. Kemungkinan besar versi yang anda dapatkan akan lebih baru dari ini.

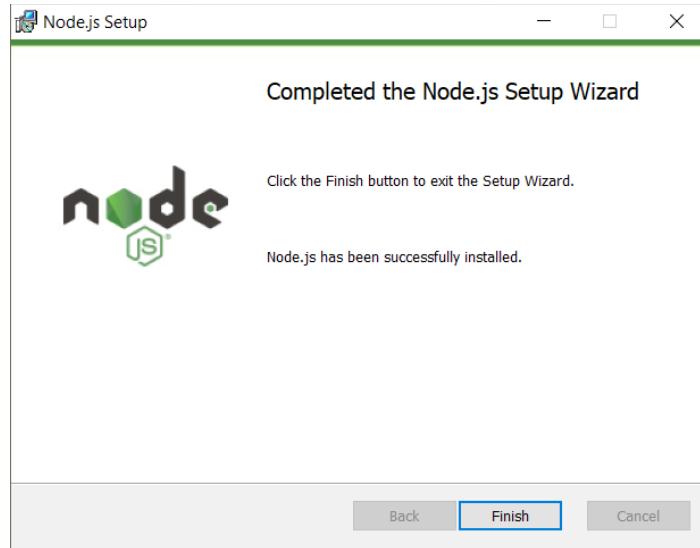
Karena kita hanya butuh perintah dasar saja, saya akan pilih versi 16. Klik tombol warna hijau untuk memulai proses download file **node-v16.14.2-x64.msi** (berukuran sekitar 27MB). Setelah itu, double klik file instalasi dan ikuti prosesnya. Tidak ada pengaturan yang perlu diubah, cukup klik tombol next beberapa kali sampai selesai.



Gambar: Jendela awal proses instalasi Node.js



Gambar: Node.js akan diinstall di C:\Program Files\nodejs



Gambar: Proses instalasi Node.js sudah selesai

Setelah instalasi selesai, mari cek apakah Node.js sudah bisa diakses dari cmd atau belum. Silahkan buka cmd, lalu ketik perintah berikut:

```
node -v
```

Seharusnya akan tampil angka yang berisi versi dari Node.js.

Kemudian ketik lagi

```
npx -v
```

Dan kembali, seharusnya juga tampil angka yang merupakan versi dari npx.

A screenshot of a Windows Command Prompt window titled "Command Prompt". It shows the following output:

```
Microsoft Windows [Version 10.0.19044.1586]
(c) Microsoft Corporation. All rights reserved.

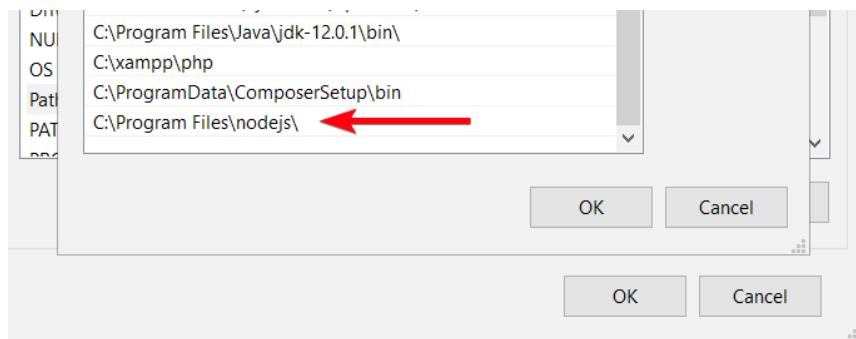
C:\Users\Andre>node -v
v16.14.2

C:\Users\Andre>npx -v
6.14.16

C:\Users\Andre>
```

Gambar: Periksa versi Node.js dan npm yang terinstall

Apabila kedua perintah ini tidak dikenali, cek pengaturan *system environment variable*, pastikan folder C:\Program Files\nodejs\ sudah terdaftar.



Gambar: List folder di system environment variable Windows

Mengenal npm dan npx

Pada saat kita menginstall Node.js, terdapat aplikasi lain yang juga ikut terinstall, yakni **npm** (node package manager) dan **npx** (node package execution). [Npm](#) adalah sebuah package manager JavaScript. Dengan npm, kita bisa menginstall berbagai library JavaScript langsung dari cmd atau terminal. Sedangkan npx selain menginstall package tersebut, juga bisa menjalankannya.

Setelah Nodejs tersedia, saatnya tes jalankan perintah **create react app**.

Masih menggunakan cmd, silahkan masuk ke folder yang dipakai untuk latihan kode React. Dalam praktek ini saya sudah menyediakan folder **belajar_react** di drive D, sehingga pathnya ada di D:\belajar_react. Anda bebas jika ingin menggunakan folder lain.

Setelah berada di dalam folder tersebut, ketik perintah berikut dan tekan enter:

```
npx create-react-app my-app
```

Pastikan koneksi internet tersedia dan punya paket data yang cukup, karena perintah di atas akan mendownload ribuan file dengan ukuran mencapai 150MB. Tergantung kecepatan koneksi, proses download ini bisa memakan waktu hingga beberapa menit.

```
Command Prompt
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>D:
D:>cd belajar_react
D:\belajar_react>npx create-react-app my-app
npx: installed 67 in 47.334s

Creating a new React app in D:\belajar_react\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

Gambar: Proses instalasi create-react-app

Setelah selesai, di dalam folder D:\belajar_react akan muncul folder baru bernama **my-app**. Di dalam folder inilah nantinya kita akan membuat berbagai perintah React, tidak lagi ke file HTML langsung.

Jika terjadi kendala ketika menjalankan kode ini, folder **my-app** hasil perintah `npx create-react-app my-app` juga tersedia di Google Drive. Silahkan download dan unzip ke folder D:\belajar_react.

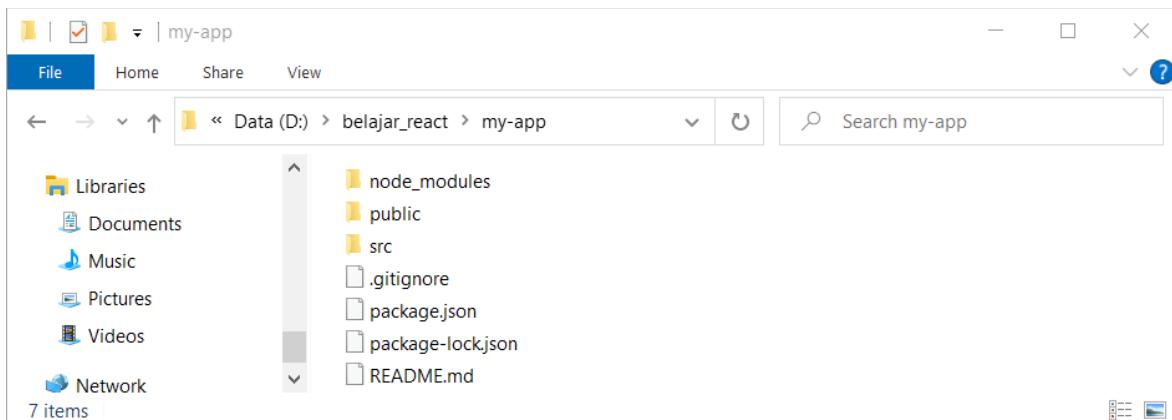
Nama folder **my-app** boleh saja diganti (di-rename) menjadi nama lain sesuai project yang akan kita buat, misalnya **my-todo**, **project-reactku**, atau nama lain. Perintah install yang dipakai sebelumnya juga bisa langsung ditulis dengan nama project ini. Misalnya jika diketik:

```
npx create-react-app project-reactku
```

Maka akan muncul folder **project-reactku** yang berisi semua file `create-react-app`. Nama folder boleh bebas dan tidak berpengaruh ke kode React yang akan kita tulis nantinya.

15.2. File Bawaan Create React App

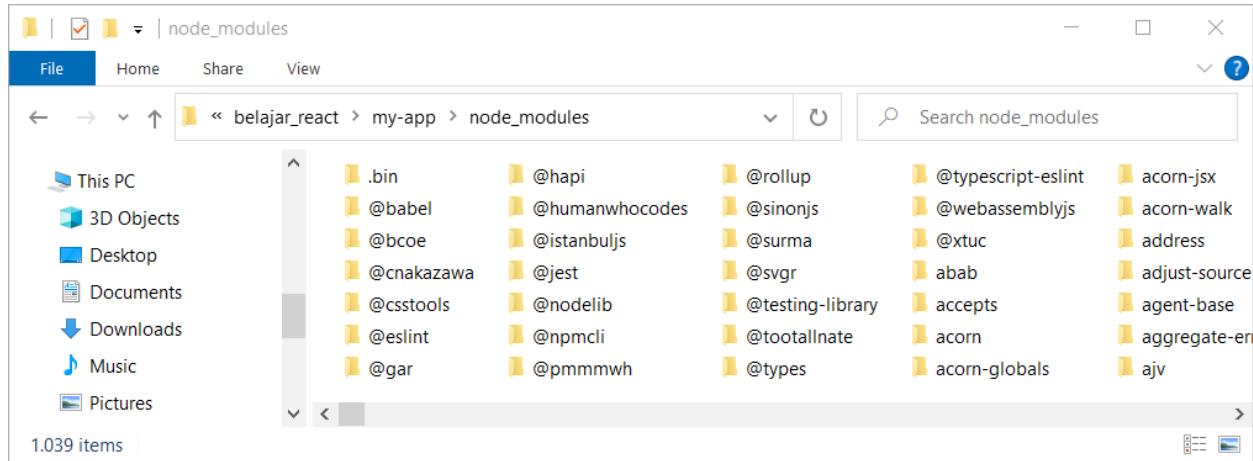
Folder **my-app** berisi 3 folder dan 4 file. Berikut penjelasan dari setiap folder:



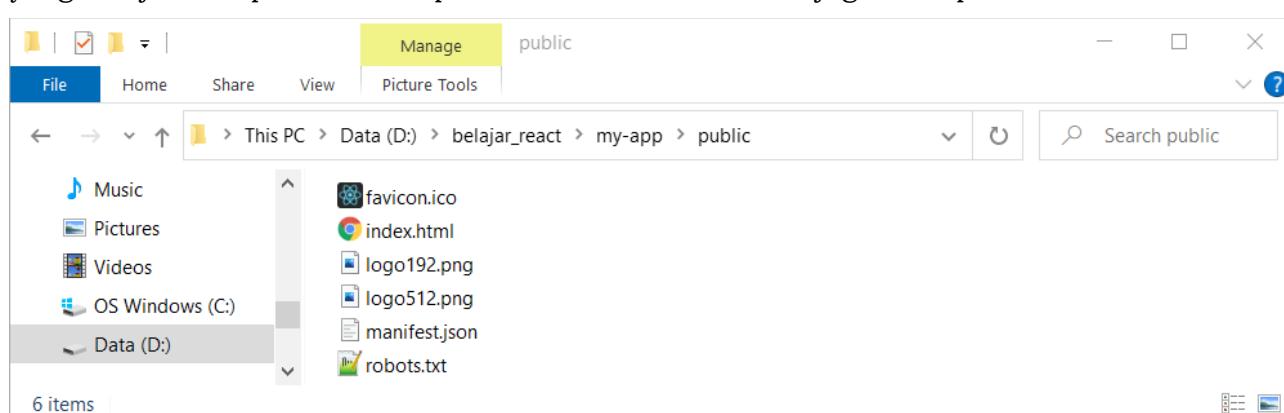
Gambar: Isi folder my-app

Folder **node_modules** berisi file internal dari berbagai *dependency* milik `create-react-app`. Ukurannya mencakup 98% dari total semua folder (terdiri dari 30.000 lebih file). Folder ini bisa kita biarkan apa adanya dan tidak akan diakses secara langsung.

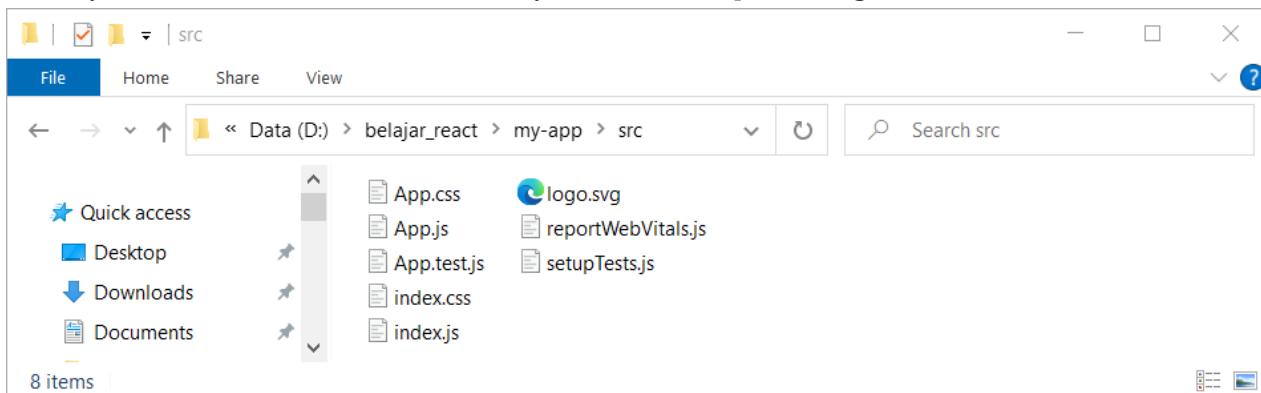
Create React App



Folder **public** berisi file yang bisa diakses public secara langsung. Diantaranya kita bisa menyimpan file gambar, file favicon, dsb. Di dalam folder public juga terdapat file **index.html** yang menjadi template utama aplikasi React. Untuk saat ini juga tidak perlu di modif.



Folder **src** merupakan folder utama dari kode React. Disinilah nantinya kita akan banyak membuat kode program. Bawaan "create react app", sudah ada sekitar 8 file di dalamnya. File ini hanya berisi kode contoh dan nantinya akan kita hapus sebagian.



Di luar 3 folder ini, terdapat 4 file yang langsung ada di dalam folder my-app: `.gitignore`, `package.json`, `package-lock.json` dan `README.md`.

Tiga file pertama dipakai untuk pengelolaan package Nodejs. Untuk penggunaan dasar, ketiganya tidak perlu kita utak-atik. Sedangkan file `README.md` berisi dokumentasi singkat dari semua file yang dihasilkan perintah "create my app". Jika ingin membacanya, bisa buka dari teks editor seperti Notepad++ atau VSCode.

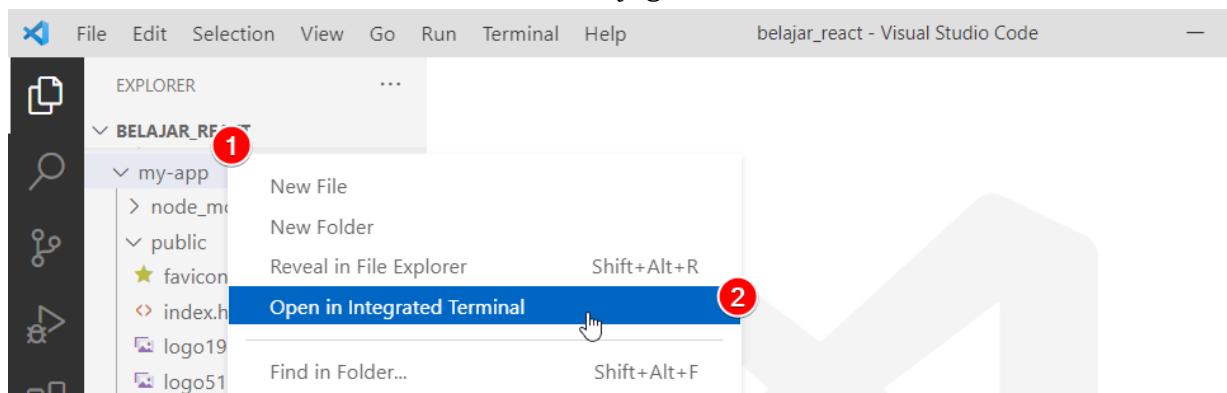
Untuk menghemat kuota data, silahkan copy folder **my-app** ke tempat lain sebagai file master. Jika nanti kita butuh menginstall create react app lagi, cukup copy file ini agar tidak perlu menjalankan ulang perintah `create react app` dari cmd.

15.3. Menjalankan Live Server

Paket create react app memiliki *live development server* yang bisa kita pakai untuk melihat hasil kode program React secara realtime. Ketika file React di save, halaman di web browser akan langsung reload secara otomatis.

Caranya, buka cmd atau terminal dan masuk ke folder tempat file "create react app" di simpan. Dalam praktek sebelumnya, folder yang dimaksud ada di `D:\belajar_react\my-app`. Setelah itu, jalankan perintah "`npm start`" (tanpa tanda kutip).

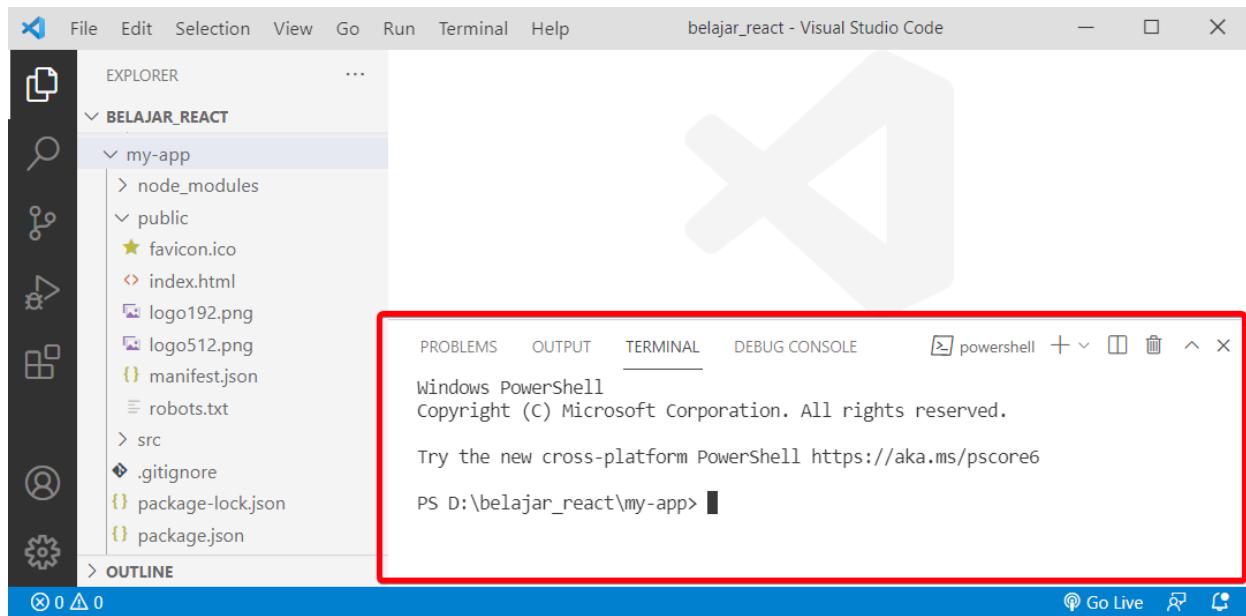
Ini bisa dilakukan dari cmd Windows, atau bisa juga dari menu terminal bawaan VS Code:



Gambar: Cara membuka terminal di VS Code

Untuk di VS Code, silahkan klik kanan folder **my-app** di bagian "Explorer" (1), lalu pilih menu "Open in Integrated Terminal" (2).

Create React App



Gambar: Tampilan terminal di VS Code

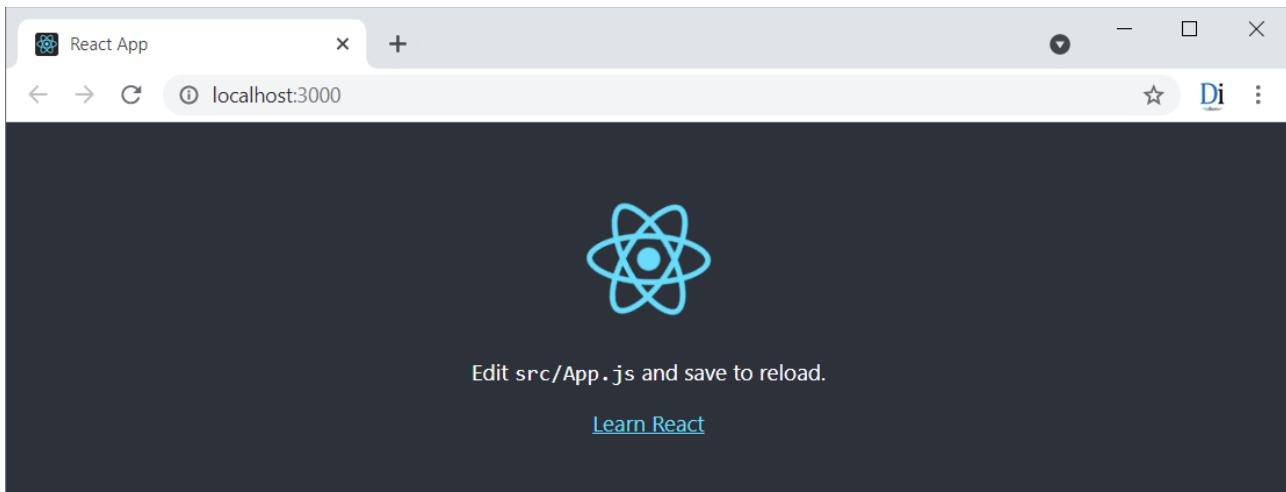
Sesaat kemudian akan tampil tab terminal di bagian bawah VS Code. Terminal ini sudah terbuka di `D:\belajar_react\my-app`, sehingga bisa langsung diketik "`npm start`":

```
PS D:\belajar_react\my-app> npm start
Compiled successfully!
You can now view my-app in the browser.
Local:          http://localhost:3000
On Your Network: http://192.168.1.7:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

Gambar: Menjalankan live server bawaan create react app

Perintah `npm start` akan men-compile semua kode React yang terdapat di folder **my-app**, kemudian menjalankan sebuah *live server* pada alamat `http://localhost:3000`. Perintah ini mungkin butuh waktu beberapa detik. Jika tidak ada masalah, akan terbuka tab baru di web browser yang menampilkan halaman berikut:

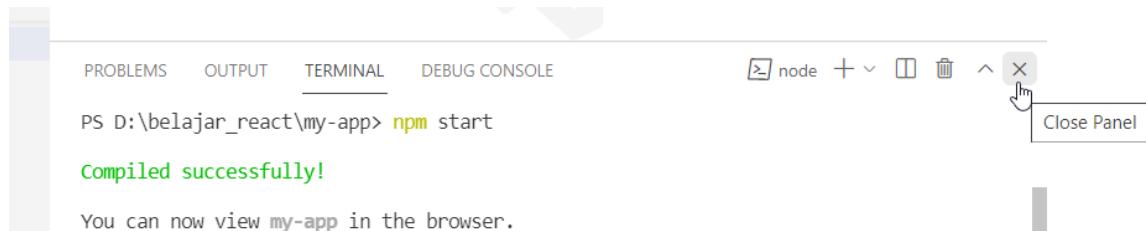
Create React App



Gambar: Tampilan create react app di web browser

Jika halaman tidak langsung terbuka, silahkan ketik manual `http://localhost:3000` di web browser.

Tab terminal VS Code boleh saja ditutup dengan men-klik tombol close (x) di sudut kanan. Meskipun sebenarnya ini tidak benar-benar menghentikan terminal, tapi hanya disembunyikan saja. Jika ingin menampilkan kembali, bisa klik menu **View -> Terminal** atau tekan tombol `CTRL + ``.



Gambar: Menutup terminal dengan klik icon (x) / Close Panel

Jika perintah `npm start` dijalankan dari cmd, maka jendela cmd itu tidak boleh ditutup (biarkan saja berjalan). Karena jika di tutup, live server juga akan berhenti.

15.4. Modifikasi File Create React App

Tampilan di halaman `localhost:3000` berasal dari file yang ada di folder `my-app\public` dan `my-app\src`. Kita akan coba modifikasi beberapa file untuk melihat apa fungsinya.

Pertama, silahkan buka file `my-app\public\index.html` di VS Code. File ini terdiri dari beberapa tag HTML dasar. Dengan menghapus semua baris komentar, berikut isi file tersebut:

```
public\index.html  
1  <!DOCTYPE html>  
2  <html lang="en">
```

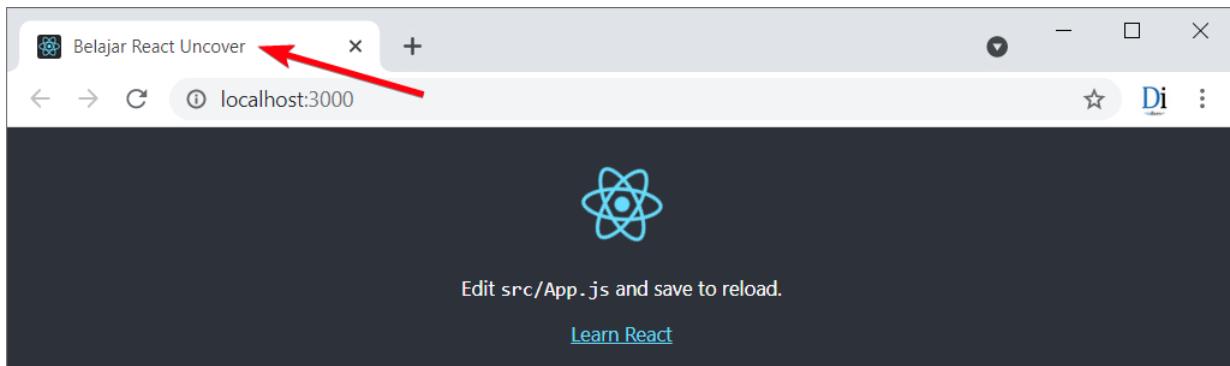
```

3 <head>
4   <meta charset="utf-8" />
5   <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <meta name="theme-color" content="#000000" />
8   <meta
9     name="description"
10    content="Web site created using create-react-app"
11  />
12  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
14  <title>React App</title>
15 </head>
16 <body>
17  <noscript>You need to enable JavaScript to run this app.</noscript>
18  <div id="root"></div>
19 </body>
20 </html>

```

Ini merupakan file template yang menjadi tempat dari kode React. Perhatikan tag `<div id="root">` di baris 18, itulah tag yang nantinya menjadi target dari `ReactDOM.createRoot().render()` di file `src\index.js` (akan kita lihat sesaat lagi).

Semua tag di halaman ini bebas jika ingin dimodifikasi. Sebagai contoh, saya akan ganti tag `<title>React App</title>` di baris 14 menjadi `<title>Belajar React Uncover</title>`. Setelah itu, save file `index.html` dan langsung lihat di web browser:



Gambar: Titel halaman `localhost:3000` sudah berubah menjadi "Belajar React Uncover"

Hasilnya, title halaman `localhost:3000` sudah berubah dari "React App" menjadi "Belajar React Uncover".

Kadang live server bawaan create react app tidak selalu live dan bisa "nyangkut". Oleh karena itu silahkan refresh manual web browser jika tidak ada perubahan.

Contoh lain, tag `<link rel="icon" href="%PUBLIC_URL%/favicon.ico"/>` di baris 5 berfungsi untuk menampilkan gambar favicon. Secara default, keyword `%PUBLIC_URL%` merujuk ke folder `public`, maka icon yang dimaksud adalah file `public\favicon.ico`. Kita bisa ganti dengan gambar lain jika diinginkan.

Kesimpulannya, file yang paling penting di folder **public** hanyalah `index.html`. Sisa file lain seperti `favicon.ico`, `logo192.png`, atau `robots.txt` boleh saja dihapus atau dimodifikasi jika tidak diperlukan.

Sekarang kita masuk ke folder **src** yang berisi beberapa file React. File paling penting adalah `index.js` yang berisi kode berikut:

`src\index.js`

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';

6
7 ReactDOM.render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>,
11   document.getElementById('root')
12 );
13
14 reportWebVitals();
```

Di awal kode program terdapat 5 kali perintah **import**. Fungsi dari perintah import sudah pernah kita bahas di awal buku, yakni pada *JavaScript Module* di bab 2 tentang [JavaScript \(ES6\) untuk React](#). Perintah antara baris 1-5 akan meng-import class, file css serta komponen yang diperlukan ke dalam file saat ini.

Fokus utama ada di baris 7-12. Ini merupakan perintah `ReactDOM.render()` untuk men-inject kode React ke dalam element HTML. Element yang dimaksud tidak lain adalah tag `<div id="root">` yang ada di file `public\index.html`.

Pada saat buku ini saya update di awal April 2022, React 18 baru rilis sekitar 1 minggu lalu. Sepertinya tim *create react app* masih belum mengupdate perintah render dari `ReactDOM.render()` milik React 17 menjadi `ReactDOM.createRoot().render()` yang ada di React 18.

Mungkin saat anda membaca materi ini, baris 7-12 dari file `my-app\index.js` sudah menggunakan perintah `ReactDOM.createRoot().render()`. Jika belum, kita akan ubah secara manual sesaat lagi.

Sebagai argument pertama `ReactDOM.render()`, terdapat tag `<React.StrictMode>` yang menjadi container dari tag `<App />`.

Tag `<React.StrictMode>` dipakai agar kode program React berjalan di "strict mode" atau mode yang ketat aturan. Mode ini akan menampilkan banyak pesan warning jika ada sesuatu yang

kurang pas. Warning sangat berguna untuk mencegah bug yang mungkin terjadi, meskipun kodenya tetap jalan.

Mirip seperti tag `<React.Fragment>`, tag `<React.StrictMode>` tidak berdampak apapun di sisi tampilan, hanya sekedar container saja.

Di dalam `<React.StrictMode>`, terdapat tag `<App>`. Ini merujuk ke sebuah komponen `App` yang di-import pada baris 4. Kode untuk komponen `App` ada di file `src\app.js`, mari buka file ini:

`src\app.js`

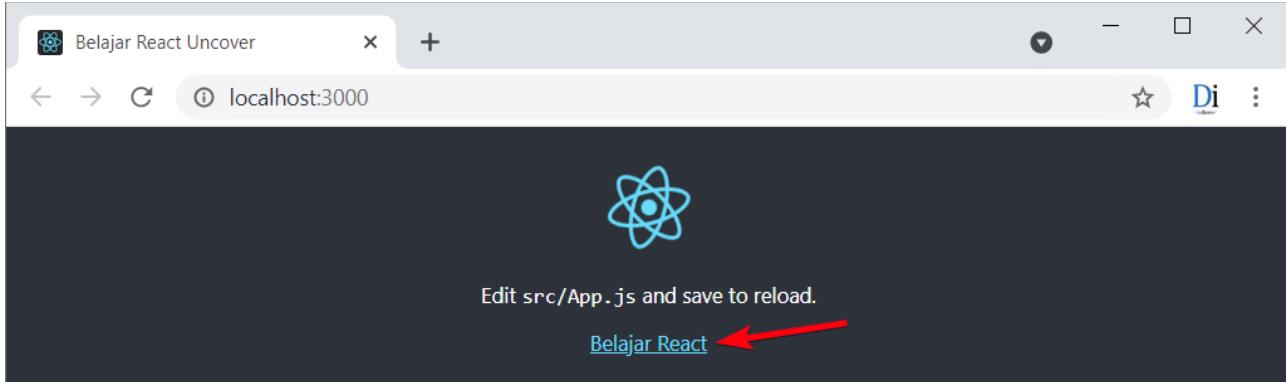
```

1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
22   );
23 }
24
25 export default App;

```

Di baris 1 dan 2 kembali terdapat perintah `import`. Perintah pertama untuk mengimport gambar `logo.svg`, dan perintah kedua untuk mengimport file `App.css`.

Isi komponen `App` berbentuk functional component yang langsung mengembalikan sebuah struktur JSX. Kode JSX di baris 6-21 inilah yang kita lihat di halaman `localhost:3000`. Sebagai contoh, silahkan edit baris 18 dari "Learn React" menjadi "Belajar React", save file dan cek hasilnya di web browser:



Gambar: Perubahan halaman localhost:3000

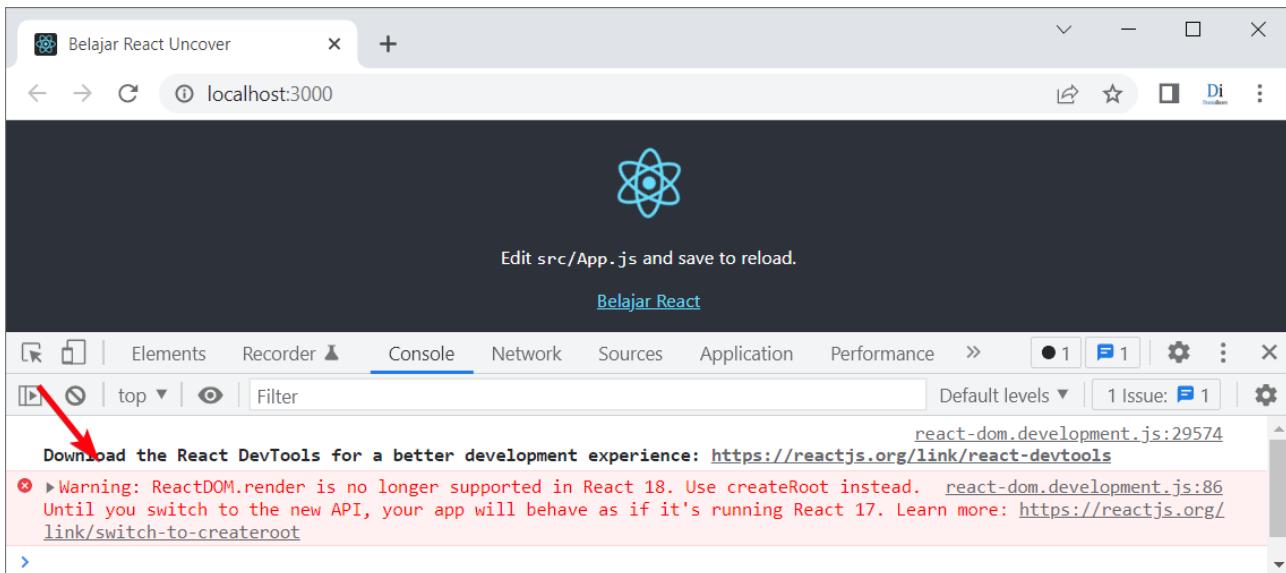
Sisa kode lain di dalam JSX berisi tag HTML untuk menampilkan gambar dengan tag ``, sebuah teks dengan tag `<p>`, serta tag `<a>` untuk membuat link. Semua kode ini bisa di modif sesuai keinginan.

15.5. Mengganti Perintah Render ke React 18

Sepanjang pembahasan kode React sejak awal buku, kita selalu mendapat pesan warning di tab console developer tools terkait perintah `ReactDOM.createRoot().render()`.

React 18 mewajibkan proses import library React DOM menggunakan perintah import "react-dom/client". Sayangnya, perintah ini belum bisa dijalankan dengan tag `<script>`. Sekarang dengan `create react app`, itu sudah tidak masalah.

Akan tetapi pada saat saya coba menginstall `create react app` di awal April 2022, ternyata tetap tampil pesan warning, tapi kali ini karena perintah render di file `my-app\index.js` masih memakai `ReactDOM.render()` bawaan React 17:



Gambar: Pesan warning karena penggunaan method ReactDOM.render() bawaan React 17

Saya yakin ini hanya sementara, menunggu tim pengembang React mengupdate kode bawaan create react app. Namun kita juga bisa mengupdatenya secara manual. Silahkan buka file `my-app\index.js`, lalu modifikasi menjadi berikut:

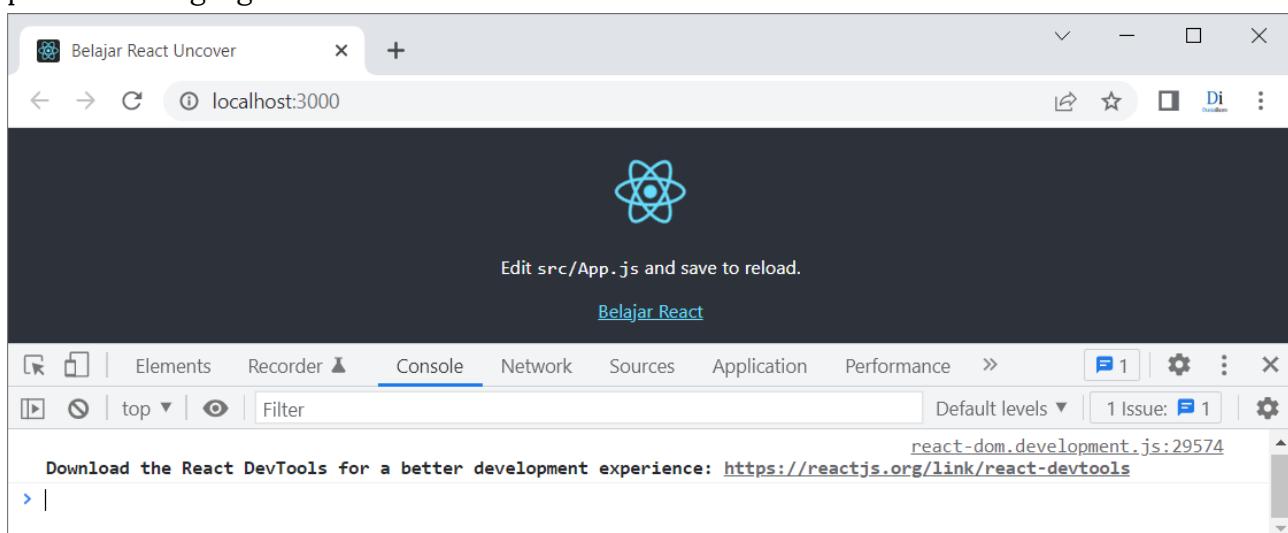
```
src\index.js

1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6
7 ReactDOM.createRoot(document.getElementById('root'))
8 .render(
9   <React.StrictMode>
10    <App />
11  </React.StrictMode>
12 );
13
14 reportWebVitals();
```

Perubahan ada di baris 2, dan 7-12. Di baris 2, sekarang mengimport `ReactDOM from 'react-dom/client'` dari sebelumnya `ReactDOM from 'react-dom'`. Inilah syarat yang selalu diminta React 18 di dalam tab console.

Setelah itu kita perlu mengkonversi perintah `ReactDOM.render()` menjadi `ReactDOM.createRoot().render()` di baris 7-12. Perintah `document.getElementById('root')` sekarang diinput sebagai argument dari method `createRoot()`. Sedangkan tag `<React.StrictMode>` dan `<App />` menjadi argument dari method `render()`.

Save kode di atas dan cek ke dalam tab console di developer tools, sekarang sudah tidak ada pesan warning lagi:



Gambar: Tidak ada lagi warning di dalam tab console

Salah satu alasan kenapa method `ReactDOM.createRoot().render()` harus diimport dari library `react-dom/client`, dan bukan `react-dom` saja adalah rencana tim React untuk mengembangkan kode yang bisa berjalan di server. Untuk perintah tersebut, nantinya akan menjadi jatah library `react-dom/server`.

Perkembangan ini sangat menarik diikuti karena bisa membuka banyak kemungkinan baru. Salah satunya me-render React component di server, atau berkomunikasi langsung dengan database. Saat ini server side React masih dalam versi awal pengembangan.

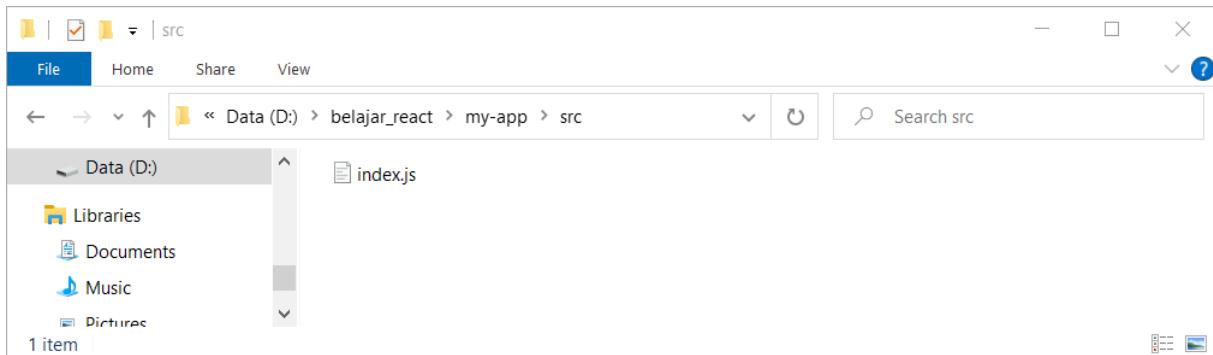
15.6. Membuat React Component

Berdasarkan praktik sebelumnya, kita bisa lihat alur dasar penulisan kode di create react app:

1. File `public\index.html` bertindak sebagai file template utama. Jika butuh mengubah kode untuk tag `<head>`, disinilah tempatnya. Di dalam tag `<body>` terdapat tag `<div id="root">` sebagai lokasi untuk kode React.
2. File `src\index.js` bertindak sebagai file React utama untuk mengisi tag `<div id="root">` yang ada di `public\index.html`.

Kedua file inilah yang menjadi inti dari **create react app**. Sisa file lain di folder `public` dan juga folder `src` merupakan file pelengkap saja.

Agar lebih memahami konsep ini, saya akan hapus semua isi folder `src` selain `index.js` dan membuat komponen React dari nol.

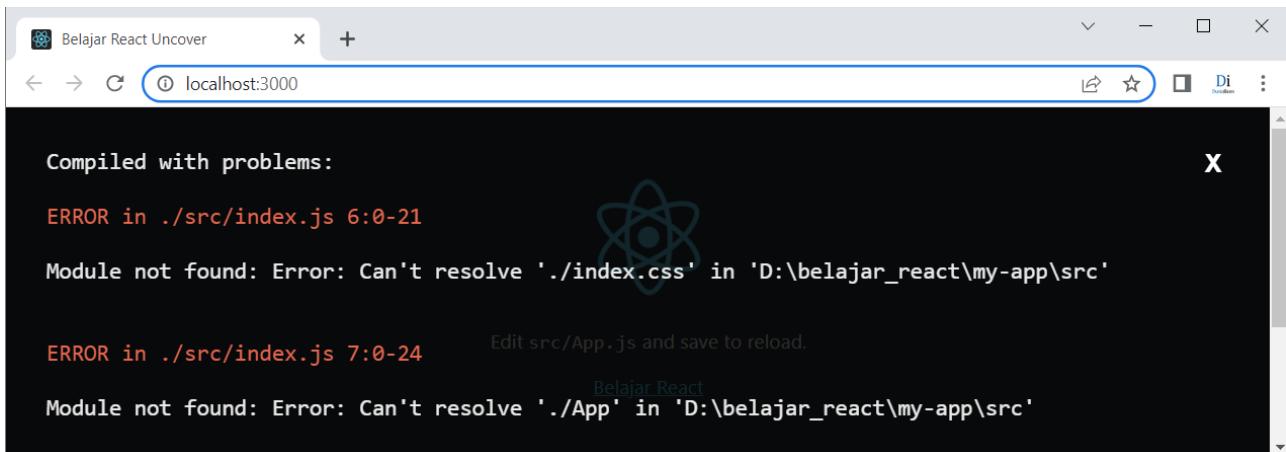


Gambar: Sisakan 1 file index.js di folder src

Menghapus file bawaan create react app sebenarnya hampir selalu dilakukan untuk setiap project React.

Setelah menghapus semua file selain `index.js`, halaman `localhost:3000` akan error dan itu normal.

Create React App



Gambar: Halaman localhost:3000 error karena tidak menemukan komponen App.js

Sekarang buka file `src\index.js` dan modifikasi menjadi berikut:

```
src\index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import Belajar from './Belajar';
4
5 ReactDOM.createRoot(document.getElementById('root'))
6 .render(
7   <React.StrictMode>
8     <Belajar />
9   </React.StrictMode>
10 );
```

Perintah import di baris 1 dan 2 dipakai untuk mengimport class `React` dan `ReactDOM` ke dalam file `index.js`. Ini adalah pengganti tag `<script>` yang selama ini kita pakai untuk mengakses file `react.development.js` dan `react-dom.development.js`, yakni menggantikan 2 tag berikut:

```
<script src="js/react.development.js"></script>
<script src="js/react-dom.development.js"></script>
```

Perintah import di baris 3 berfungsi untuk mengimport komponen `Belajar` ke dalam file `index.js`. Dengan perintah ini, harusnya ada file `Belajar.js` di folder `src` (akan kita buat sesaat lagi). Perhatikan cara penulisan path `"./Belajar"`, yang harus diawali tanda titik, forward slash dan nama file JavaScript tanpa akhiran `.js`.

Kemudian di baris 5 terdapat perintah `ReactDOM.createRoot().render()` untuk memproses kode JSX ke dalam tag `<div id="root">`. Kode JSX yang dimaksud ada di baris 7-9, yakni tag `<Belajar />` yang berada dalam container `<React.StrictMode>`.

Selanjutnya, buatlah file baru bernama `Belajar.js` di dalam folder `src`, lalu isi dengan kode berikut:

Create React App

src\Belajar.js

```
1 const Belajar = () => <h1>Belajar React</h1>;
2
3 export default Belajar;
```

Yup, hanya butuh 2 perintah saja, dan ini adalah sebuah React component!

Di baris 1 saya membuat konstanta `Belajar` yang langsung mengembalikan kode JSX `<h1>Belajar React</h1>`. Ini merupakan penulisan singkat dari *arrow notation* yang sama artinya jika ditulis sebagai berikut:

```
const Belajar = () => {
  return (
    <h1>Belajar React</h1>
  )
};
```

Konstanta `Belajar` kemudian di export dengan perintah `export default Belajar` di baris 3. Ini harus ditulis agar komponen `Belajar` bisa di-import oleh file lain. Lebih lengkap tentang maksud perintah `export` sudah pernah kita bahas di materi **JavaScript Module** pada bab 2.

Save file `Belajar.js` dan cek di web browser:



Gambar: Tampilan halaman localhost:3000

Sip, kita sudah berhasil menampilkan React component di create react app. Tampilan teks "Belajar React" berasal dari komponen `<Belajar/>` yang ada di file `src\Belajar.js`.

Anda bebas ingin memberikan nama file untuk komponen yang di import, tidak harus sama dengan nama komponen. Misalnya boleh saja menyimpan komponen `Belajar` ke dalam file bernama `Foo.js`.

Namun kebiasaan programmer React (best practicenya), gunakan nama file yang sama dengan nama komponen. Ini juga akan memudahkan kita karena bisa menebak nama komponen dari nama filenya saja.

Percobaan berikutnya, saya akan modif isi `Belajar.js` sebagai berikut:

src\Belajar.js

```
1 const Tombol = (props) => {
2   const handleButtonClick = () => {
```

```

3     alert(props.pesan);
4 }
5
6 return (
7   <button onClick={handleButtonClick} style={{ margin: "10px" }}>
8     {props.children}
9   </button>
10 )
11 }
12
13 const Belajar = () => {
14   return (
15     <div>
16       <h1>Belajar React</h1>
17       <Tombol pesan="Belajar React">React</Tombol>
18       <Tombol pesan="Belajar JavaScript">JavaScript</Tombol>
19     </div>
20   )
21 }
22
23 export default Belajar;

```

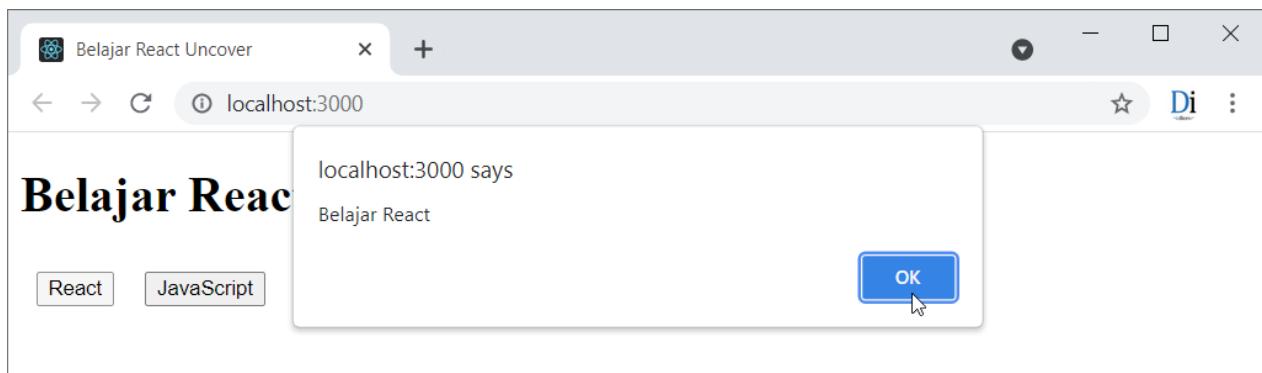
File `Belajar.js` sekarang berisi 2 komponen: `Tombol` di baris 1-11 dan `Belajar` di baris 13-21.

Untuk kode yang agak kompleks seperti ini, disarankan membacanya dari komponen yang akan di export terlebih dahulu. Di baris 23, komponen yang dimaksud adalah `Belajar`.

Komponen `Belajar` mengembalikan kode JSX yang terdiri dari 1 tag `<h1>` dan 2 tag `<Tombol>`. Karena diawali dengan huruf besar, tag `<Tombol>` ini merujuk ke komponen `Tombol` yang ada di baris 1-11. Masing-masing tag `<Tombol>` mengirim nilai `props` dalam bentuk atribut `pesan` serta sebuah teks antara tag pembuka dan penutup.

Di baris 6-10, komponen `Tombol` mengembalikan kode JSX dalam bentuk tag `<button>` dengan event `onClick`. Dengan demikian pada saat tag `<button>` ini di klik, method `handleButtonClick()` di baris 2-4 akan berjalan dan menampilkan pesan teks.

Berikut tampilan dari penambahan kode di atas:



Gambar: Komponen Belajar dengan tambahan komponen Tombol

Praktek dari komponen Tombol ini sebenarnya sudah kita pakai di bab Event. Jika kurang paham dengan alur kerjanya, boleh pelajari kembali bab tersebut.

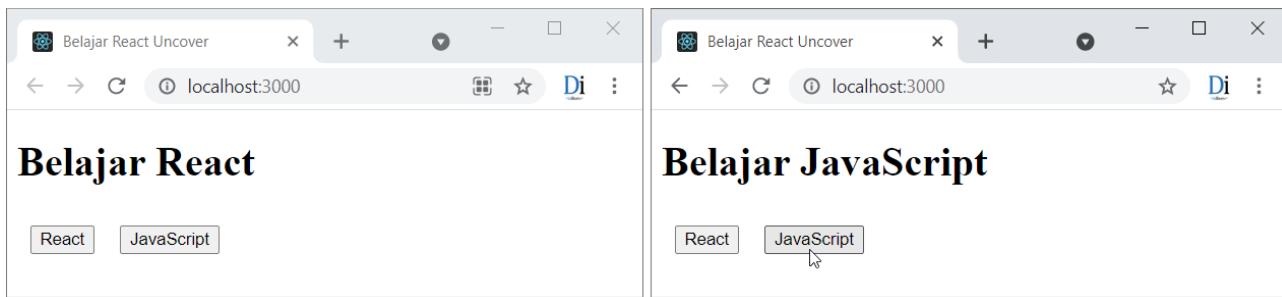
File Belajar.js tidak perlu meng-export komponen Tombol, tapi cukup komponen Belajar saja. Ini karena komponen Tombol hanya dipakai secara internal oleh komponen Belajar.

15.7. Membuat State

Komponen Belajar saat ini sudah berisi event, sekarang akan coba kita tambah dengan state. Berikut perubahannya:

```
src\Belajar.js

1 import React from 'react';
2
3 const Tombol = (props) => {
4
5     const handleButtonClick = () => {
6         props.onTombolClick(props.children);
7     }
8
9     return (
10        <button onClick={handleButtonClick} style={{ margin: "10px" }}>
11            {props.children}
12        </button>
13    )
14 }
15
16 const Belajar = () => {
17     const [judul, setJudul] = React.useState("React");
18
19     const handleTombolclick = (judul) => {
20         setJudul(judul);
21     }
22     return (
23         <div>
24             <h1>Belajar {judul}</h1>
25             <Tombol onClick={handleTombolclick}>React</Tombol>
26             <Tombol onClick={handleTombolclick}>JavaScript</Tombol>
27         </div>
28     )
29 }
30
31 export default Belajar;
```



Gambar: Tampilan komponen dengan state

Karena kita menggunakan functional component, maka pembuatan state harus dengan `useState` hook seperti di baris 17. Tambahan lain, di baris 1 perlu perintah `import React from 'react'` karena `useState` adalah sebuah method milik **React** class.

Dengan tambahan state ini, maka ketika tombol di klik, judul di tag `<h1>` pada baris 24 akan berganti-ganti dari "Belajar React" menjadi "Belajar JavaScript" dan sebaliknya. Event yang mengubah state judul ada di komponen Tombol, yakni dengan prinsip "state up" (method `handleTombolclick` dikirim sebagai `props`).

Perhatikan kembali perintah untuk pembuatan state, yakni:

```
const [judul, setJudul] = React.useState("React");
```

Tidak ada yang salah dari cara ini. Tapi sebagai alternatif penulisan, method `React.useState()` bisa dipersingkat menjadi `useState()` saja dengan cara menambah alias pada saat proses import file React.

Jika perintah import di baris 1 diubah menjadi sebagai berikut:

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

Maka kita bisa membuat state `judul` dengan perintah:

```
const [judul, setJudul] = useState("React");
```

Teknik ini juga berlaku untuk method hook lain. Misalnya jika proses import ditulis dengan:

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

Maka pembuatan `ref` juga bisa dilakukan dengan perintah `const foo = useRef("Hei")`, tidak perlu menulisnya sebagai `const foo = React.useRef("Hei")`.

Penulisan antara `React.useState()` atau `useState()` ini hanya sekedar alternatif. Anda boleh pakai yang mana saja. Yang jelas jika kita ingin menulis hook tanpa awal adalah class "React", maka harus sertakan nama alias pada perintah `import`.

15.8. Pemisahan Komponen

Best practice lain dari create react app adalah, setiap komponen sebaiknya disimpan ke dalam file terpisah. Dalam praktik kita saat ini, di dalam file `Belajar.js` terdapat komponen `Belajar` dan juga komponen `Tombol`. Kedua komponen sebaiknya dipisah ke dalam file tersendiri.

Caranya, silahkan buat file baru bernama `Tombol.js` di dalam folder `my-app\src`, lalu isi dengan kode berikut:

`src\Tombol.js`

```

1  const Tombol = (props) => {
2
3    const handleButtonClick = () => {
4      props.onTombolClick(props.children);
5    }
6
7    return (
8      <button onClick={handleButtonClick} style={{ margin: "10px" }}>
9        {props.children}
10       </button>
11    )
12  }
13
14 export default Tombol;
```

Kode di atas hanya hasil cut-paste dari komponen `Tombol` di file `Belajar.js`. Tambahan ada di baris 14, yakni perintah `export` agar komponen ini bisa diakses dari file `Belajar.js`.

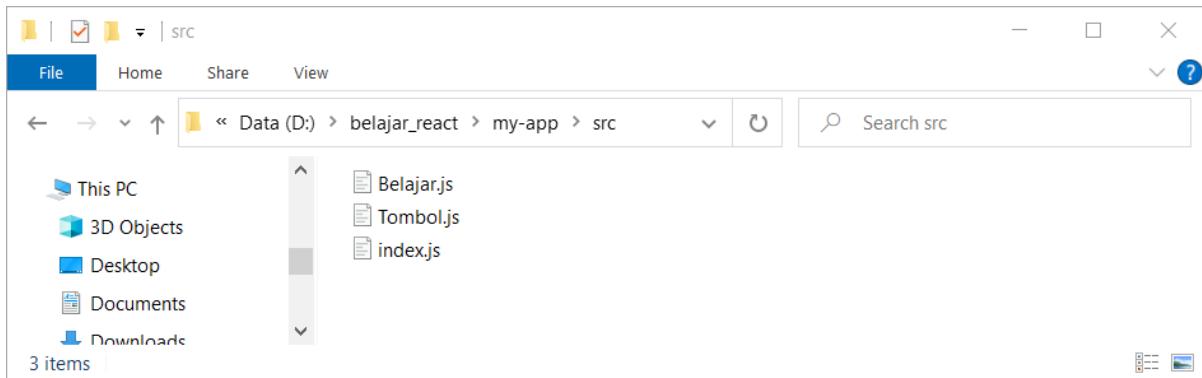
Kemudian modifikasi juga isi file `Belajar.js` sebagai berikut:

`src\Belajar.js`

```

1  import React, { useState } from 'react';
2  import Tombol from './Tombol';
3
4  const Belajar = () => {
5    const [judul, setJudul] = useState("React");
6
7    const handleTombolclick = (judul) => {
8      setJudul(judul);
9    }
10   return (
11     <div>
12       <h1>Belajar {judul}</h1>
13       <Tombol onClick={handleTombolclick}>React</Tombol>
14       <Tombol onClick={handleTombolclick}>JavaScript</Tombol>
15     </div>
16   )
17 }
18
19 export default Belajar;
```

Di dalam file `Belajar.js` sekarang hanya ada komponen `Belajar` saja. Untuk mengakses komponen `Tombol`, dilakukan dengan cara mengimport komponen tersebut di baris 2.



Gambar: Di dalam folder `my-app\src` sekarang ada 3 buah file

Untuk praktek membuat aplikasi yang lebih kompleks, bisa saja kita butuh mengimport 3, 4 atau 10 komponen. Itu tidak masalah karena pemisahan komponen membuat kode program menjadi lebih mudah dikelola.

Agar lebih teratur lagi, semua komponen bisa di tempatkan ke dalam folder khusus. Sebagai contoh praktek, saya akan membuat folder baru bernama "**components**" di `my-app\src`, lalu memindahkan file `Belajar.js` dan `Tombol.js` ke dalamnya. Dengan perubahan ini, struktur folder `my-app\src` menjadi sebagai berikut:

```
|index.js
|--components
|  Belajar.js
|  Tombol.js
```

File `index.js` tidak ikut masuk ke folder `components` karena menjadi file utama React. Namun kita tetap perlu memodifikasi `index.js` sebab komponen `Belajar` sudah pindah folder:

`src\index.js`

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import Belajar from './components/Belajar';
4
5 ReactDOM.createRoot(document.getElementById('root'))
6 .render(
7   <React.StrictMode>
8     <Belajar />
9   </React.StrictMode>
10 );
```

Perubahannya ada di baris 3, dimana path file komponen `Belajar` sekarang ada di `'./components/Belajar'` dari sebelumnya di `'./Belajar'`.

Tidak ada aturan baku dari struktur file dan folder ini, lebih ke suka-suka kita saja. Tapi

menyimpan semua file komponen di folder `components` cukup sering diterapkan oleh programmer React.

15.9. Import File CSS

Selain mengimport komponen dalam bentuk file JS, kita juga bisa mengimport file CSS ke dalam file React. Salah satu teknik yang biasa dilakukan adalah, menyiapkan file CSS untuk setiap komponen React, tujuannya agar style itu "melekat" ke masing-masing komponen.

Sebagai contoh, saat ini kita memiliki 2 komponen di 2 file JS: `Belajar.js` dan `Tombol.js`, maka untuk setiap komponen, bisa dibuat file `Belajar.css` dan `Tombol.css`. Setiap file CSS ini akan berisi kode CSS untuk men-style komponen itu saja.

Nama file CSS boleh bebas, tapi kebiasaan programmer React (dan juga untuk memudahkan kita), buatlah nama file CSS yang sama dengan nama komponen.

Sebagai bahan praktek, silahkan buat file `Belajar.css` dan `Tombol.css` ke dalam folder `my-app\src\components`, lalu isi dengan kode berikut:

`src\components\Belajar.css`

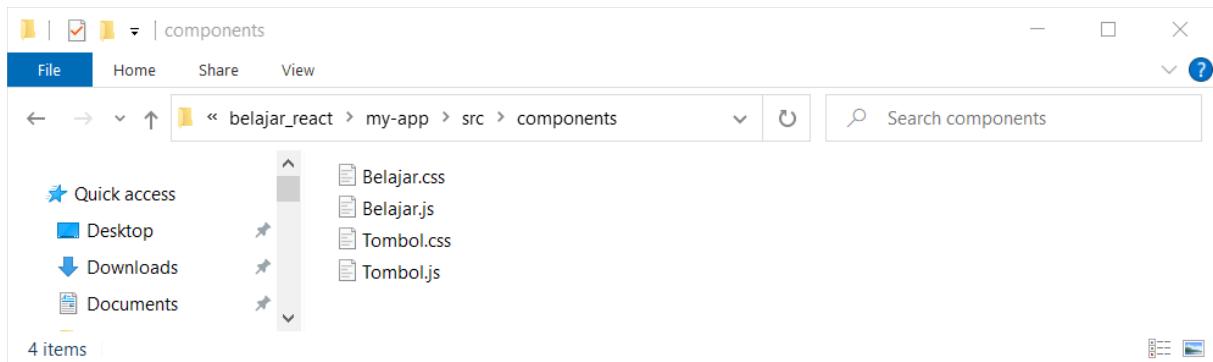
```
1  h1 {
2    background-color: gainsboro;
3    font-size: 3em;
4    text-align: center;
5    padding: 10px 20px;
6    margin: 15px 0;
7 }
```

`src\components\Tombol.css`

```
1  button {
2    background-color: white;
3    cursor: pointer;
4    padding: 10px 20px;
5    font-weight: bold;
6    border: 1px solid gray;
7    box-shadow: 2px 2px 2px black;
8    margin: 10px;
9  }
10
11 button:hover {
12   box-shadow: 1px 1px 1px black;
13   background-color: #f1f1f1;
14 }
```

Dengan tambahan 2 file ini, di dalam folder `src\components\` sudah terdapat 4 file:

Create React App



Gambar: Folder src\components\ berisi 4 file

Setiap file CSS harus kita import ke dalam komponen yang sesuai. Silahkan tambah perintah import ke bagian atas file Belajar.js dan juga Tombol.js:

src\components\Belajar.js

```
1 import React, { useState } from 'react';
2 import Tombol from './Tombol';
3 import "./Belajar.css";
4
5 const Belajar = () => {
6   ...
7 }
8
9 export default Belajar;
```

src\components\Tombol.js

```
1 import "./Tombol.css";
2
3 const Tombol = (props) => {
4   ...
5 }
6
7 export default Tombol;
```

Berbeda dengan import file JS, perintah import file CSS harus ditulis lengkap dengan nama extension-nya, yakni `import "./Belajar.css"`, dan bukan `import "./Belajar"`. Berikut hasil tampilan di web browser:



Gambar: Hasil penambahan kode CSS

Dengan perubahan tampilan ini, bisa dipastikan kalau file `Belajar.css` dan `Tombol.css` sudah berhasil di akses. Namun sebenarnya saya masih butuh 1 file lagi, yakni `index.css` sebagai pasangan dari file `index.js`. Maka silahkan buat file `index.css` di folder `my-app\src`, dan isi dengan kode berikut:

`src\index.css`

```
1 #root {  
2   display: flex;  
3   justify-content: center;  
4 }  
5  
6 div {  
7   text-align: center;  
8 }
```

Tidak lupa, di bagian atas file `index.js` juga harus mengimport file `index.css`:

`src\index.js`

```
1 import React from 'react';  
2 import ReactDOM from 'react-dom/client';  
3 import Belajar from './components/Belajar';  
4 import './index.css';  
5  
6 ReactDOM.createRoot(document.getElementById('root'))  
7 .render(  
8   ...  
9 );
```

Dan berikut hasilnya:



Gambar: Hasil penambahan file `index.css`

Membuat file CSS untuk setiap React component bukanlah hal wajib, karena tidak semua komponen perlu di-style.

15.10. CSS Modules

Proses import file CSS yang kita praktekkan sebelum ini sebenarnya bukan benar-benar per-

komponen, tapi tetap bersifat global. Maksudnya, kode CSS yang ada di `index.css` bisa saja menimpa kode CSS yang ada di `Belajar.css`.

Sebagai bukti, silahkan modifikasi file `src\index.css` dengan tambahan kode berikut:

`src\index.css`

```
1 #root {  
2     display: flex;  
3     justify-content: center;  
4 }  
5  
6 div {  
7     text-align: center;  
8 }  
9  
10 h1 {  
11     background-color: red;  
12 }
```

Kode di baris 10-12 mencoba untuk mengubah warna background tag `<h1>` menjadi merah. Dan berikut hasilnya:



Gambar: Warna background tag `<h1>` menjadi merah

Background tag `<h1>` berubah dari abu-abu menjadi merah, padahal kode yang sama ada di `Belajar.css`. Inilah yang di maksud bahwa kode CSS di setiap file yang di import bisa saling mempengaruhi. Saat proses compile oleh `create react app`, ketiga file CSS saat ini: `index.css`, `Belajar.css` dan `Tombol.css` akan disatukan.

Bagaimana agar kita bisa membatasi agar file CSS ini benar-benar untuk komponen itu saja dan tidak bisa terpengaruh file lain? React menyediakan **CSS Module** untuk keperluan ini.

Untuk bisa menerapkan CSS module, kode CSS harus ditulis dalam bentuk class selector, kemudian nama file harus disimpan dengan akhiran `*.module.css`.

Sebagai bahan praktek, silahkan rename nama file `Belajar.css` menjadi `Belajar.module.css`, dan nama file `Tombol.css` menjadi `Tombol.module.css`, kemudian modifikasi isinya dengan kode berikut:

Create React App

src\components\Belajar.module.css

```
1 .judul {  
2   ...  
3 }
```

src\components\Tombol.module.css

```
1 .tombol {  
2   ...  
3 }  
4  
5 .tombol:hover {  
6   ...  
7 }
```

Yang perlu diubah hanya bagian selector saja. Sebelumnya di Belajar.css saya langsung menggunakan selector h1, sekarang diganti menjadi class .judul. Begitu pula untuk di file Tombol.css yang sebelumnya menggunakan selector button dan button:hover, sekarang berganti menjadi class selector .tombol dan .tombol:hover.

Selanjutnya, kita juga perlu memodifikasi isi komponen Belajar.js dan Tombol.js:

src\components\Belajar.js

```
1 import React, { useState } from 'react';  
2 import Tombol from './Tombol';  
3 import styles from './Belajar.module.css';  
4  
5 const Belajar = () => {  
6   const [judul, setJudul] = useState("React");  
7  
8   const handleTombolclick = (judul) => {  
9     setJudul(judul);  
10  }  
11  return (  
12    <div>  
13      <h1 className={styles.judul}>Belajar {judul}</h1>  
14      <Tombol onClick={handleTombolclick}>React</Tombol>  
15      <Tombol onClick={handleTombolclick}>JavaScript</Tombol>  
16    </div>  
17  )  
18}  
19  
20 export default Belajar;
```

src\components\Tombol.js

```
1 import styles from './Tombol.module.css';  
2  
3 const Tombol = (props) => {  
4
```

```

5  const handleButtonClick = () => {
6    props.onTombolClick(props.children);
7  }
8
9  return (
10   <button className={styles.tombol} onClick={handleButtonClick}>
11     {props.children}
12   </button>
13 )
14 }
15
16 export default Tombol;

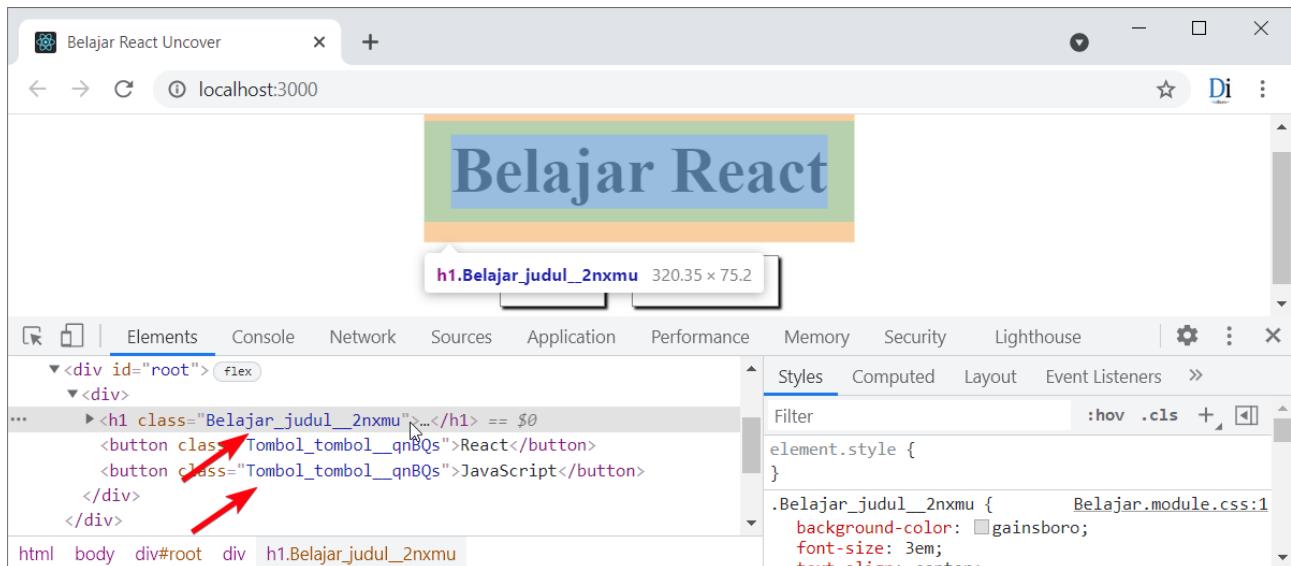
```

Perubahan pertama ada di perintah import file CSS, sebelumnya langsung mengimport file saja, sekarang terdapat nama alias "styles" seperti di baris 3 pada file Belajar.js dan baris 1 pada file Tombol.js. Pastikan juga nama file yang di import sudah diakhiri dengan `*.modules.css`.

Nama alias "styles" akan di pakai sebagai nilai dari atribut `className` dari element HTML yang ingin di style. Misalnya pada baris 13 file Belajar.js, ditulis sebagai `<h1 className={styles.judul}>`, "judul" di sini merujuk ke selector `.judul` yang ada di file Belajar.modules.css. Begitu juga di baris 10 pada file Tombol.js, dimana atribut `className={styles.tombol}` merujuk ke selector `.tombol` yang ada di file Tombol.modules.css.

Nama alias untuk sebenarnya boleh bebas, tidak harus `import styles from '...'`, bisa saja ditulis dengan `import kode from '...'.` Jika ditulis seperti ini, maka penulisan atribut menjadi `className={kode.judul}` dan `className={kode.tombol}`.

Secara teknis, React akan menggenerate class acak sebagai pengganti class selector untuk setiap module. Silahkan periksa source code akhir di web browser:



Gambar: Nama class yang digenerate React

Di sini, perintah `<h1 className={styles.judul}>` yang kita tulis di dalam komponen Belajar di generate menjadi `<h1 class="Belajar_judul__2nxmu">`. Begitu juga tag `<button className={styles.tombol}>` yang di generate menjadi `<button class="Tombol_tombol_qnBQs">`. Inilah teknik yang dipakai React agar nama class yang ditulis tidak saling bentrok satu sama lain.

Jika kita punya komponen Foo yang juga memiliki class `.judul`, itu tidak masalah dan tidak akan bentrok karena React akan men-generate nama class yang berbeda.

CSS module memang dirancang agar class CSS dipakai oleh 1 komponen saja. Akan tetapi tetap ada kemungkinan tertimpa oleh style dari kode lain sebagai akibat dari konsep "specificity".

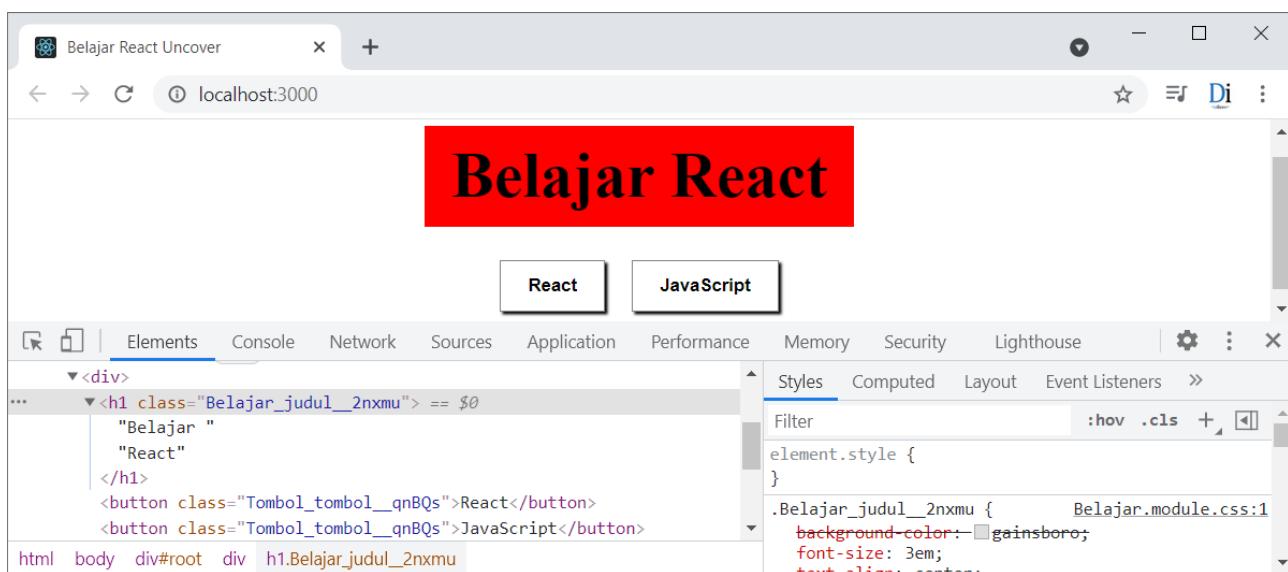
Di dalam CSS, specificity berhubungan dengan prioritas selector. Misalnya property `background-color` yang diset oleh class selector, bisa ditimpas oleh property `background-color` yang di set oleh id selector. Termasuk jika ditambah nilai `!important`.

Sebagai contoh praktik, silahkan buka file `index.css` lalu tambah perintah `!important` ke dalam selector `h1` yang sebelumnya kita pakai untuk menimpas style `h1` milik `Belajar.css`:

`src\index.css`

```

1  #root {
2    display: flex;
3    justify-content: center;
4  }
5
6  div {
7    text-align: center;
8  }
9
10 h1 {
11   background-color: red !important;
12 }
```



Gambar: Warna background tag h1 kembali menjadi merah

Hasilnya, tag `<h1>` kembali menjadi merah karena perintah `!important` memiliki prioritas tertinggi di dalam CSS. Sebelum kita lanjut, silahkan hapus selector `h1` di `index.css` agar background tag `h1` kembali menjadi abu-abu.

CSS module tidak harus dipakai untuk setiap project, lebih ke pilihan saja. Beberapa programmer ada yang lebih nyaman menggunakan 1 file CSS untuk semua aplikasi React, yakni cukup di `index.css`.

15.11. useEffect hook dan Fragment

Praktek selanjutnya saya ingin menambah `useEffect` hook dan `Fragment` ke dalam mini project kita. Silahkan modifikasi file `Belajar.js` menjadi berikut:

```
src\components\Belajar.js

1 import React, { useState, useEffect, Fragment } from 'react';
2 import Tombol from './Tombol';
3 import styles from './Belajar.module.css';
4
5 const Belajar = () => {
6   const [judul, setJudul] = useState("React");
7
8   useEffect(() => {
9     console.log("Judul diubah menjadi:", judul)
10    }, [judul])
11
12  const handleTombolclick = (judul) => {
13    setJudul(judul);
14  }
15  return (
16    <Fragment>
17      <div>
18        <h1 className={styles.judul}>Belajar {judul}</h1>
19        <Tombol onClick={handleTombolclick}>React</Tombol>
20        <Tombol onClick={handleTombolclick}>JavaScript</Tombol>
21      </div>
22    </Fragment>
23  )
24}
25
26 export default Belajar;
```



Gambar: Hasil penggunaan useEffect hook dan Fragments

Perintah di baris 1 mengimport 4 alias dari file 'react', yakni React, useState, useEffect, dan Fragment.

Hook `useEffect()` saya pakai di baris 8-10, sekedar menampilkan teks ke tab console jika isi state judul berubah. Sedangkan Fragment dipakai sebagai container dari kode JSX di baris 16.

Penulisan alias untuk hook sebenarnya tidak wajib, lebih ke mempersingkat pembuatan kode program saja. Misalnya jika kita tidak mengimport alias `{useState}`, maka perintah yang dipakai nanti harus `React.useState()`, tidak bisa langsung `useState()` seperti di baris 6.

Begini juga jika alias `{Fragment}` tidak ikut ditulis, maka tag `<Fragment>` di baris 16 harus ditulis sebagai `<React.Fragment>`.

Karena kita menggunakan create react app, React fragment juga bisa ditulis dengan pasangan tag kosong `<>` dan `</>`. Penulisan singkat ini bahkan tidak perlu import alias `{Fragment}`, dan bisa langsung dipakai.

Sebagai contoh, penulisan JSX dari kode di atas bisa juga ditulis sebagai berikut:

```
54  return (
55    <>
56      <div>
57        <h1 className={styles.judul}>Belajar {judul}</h1>
58        <Tombol onClick={handleTombolclick}>React</Tombol>
59        <Tombol onClick={handleTombolclick}>JavaScript</Tombol>
60      </div>
61    </>
62  )
```

Pasangan `<>` dan `</>` hanya bisa dipakai dalam create react app saja, tidak bisa jika kode React ditulis langsung ke dalam file HTML seperti di bab-bab sebelumnya.

15.12. Class Component

Sejak awal bab, saya langsung menggunakan functional component. Kita juga bisa memakai class component di dalam create react app. Untuk bahan praktek, silahkan modifikasi isi file `Belajar.js` menjadi sebagai berikut:

```
src\components\Belajar.js

1 import React from 'react';
2 import Tombol from './Tombol';
3 import styles from "./Belajar.module.css";
4
5 class Belajar extends React.Component {
6     constructor(props) {
7         super(props);
8         this.state = { judul: "React" };
9     }
10
11    componentDidMount() {
12        console.log("Judul diubah menjadi:", this.state.judul)
13    }
14
15    componentDidUpdate() {
16        console.log("Judul diubah menjadi:", this.state.judul)
17    }
18
19    handleTombolclick = (judul) => {
20        this.setState({ judul: judul });
21    }
22
23    render() {
24        return (
25            <>
26                <div>
27                    <h1 className={styles.judul}>Belajar {this.state.judul}</h1>
28                    <Tombol onClick={this.handleTombolclick}>React</Tombol>
29                    <Tombol onClick={this.handleTombolclick}>JavaScript</Tombol>
30                </div>
31            </>
32        )
33    }
34 }
35
36 export default Belajar;
```

Sebelumnya kita menggunakan `useEffect` hook untuk menampilkan perubahan tag judul ke dalam tab console. Sekarang proses itu dilakukan oleh method `componentDidMount()` dan `componentDidUpdate()`.

Perhatikan dalam praktek ini, komponen `Tombol` yang diimport pada baris 2 masih berbentuk functional component, dan itu tidak masalah. React mengizinkan kita menulis komponen

dalam bentuk class maupun functional yang saling mengimport satu sama lain.

15.13. Todo Project

Dalam bab sebelumnya kita membuat mini project **Todo** yang masih dalam bentuk file HTML. Sekarang saya ingin membuat ulang project tersebut ke dalam create react app.

Kode program akhir dari project Todo dipecah ke dalam 3 komponen, yakni **MyApp**, **Todo**, dan **TodoForm**. Ketiga komponen ini akan menjadi file **MyApp.js**, **Todo.js**, dan **TodoForm.js**.

Langkah pertama, edit file **index.js** sebagai berikut:

```
src\index.js

1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import './index.css';
5
6 ReactDOM.createRoot(document.getElementById('root'))
7 .render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>
11 );
```

Kali ini yang menjadi komponen utama adalah **<App />**, yang di import pada baris 3 dan ditulis ke dalam kode JSX di baris 9. Komponen ini akan kita buat sesaat lagi.

Untuk kode CSS, modifikasi file **index.css** dengan kode berikut:

```
src\index.css

1 #root {
2   display: flex;
3   justify-content: center;
4   font-size: 1.2rem;
5   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
6 }
7
8 .container {
9   margin-top: 2rem;
10  padding: 2rem 3rem;
11  box-shadow: 8px 8px 0px #007c9c;
12  border: 2px solid #19748c;
13 }
14
15 input[type="text"] {
16   width: 350px;
17   font-size: 1rem;
18   padding: 8px;
19   outline: 0;
```

```

20   border: 1px solid #007c9c;
21   border-radius: 3px 3px;
22 }
23
24 input[type="text"]:focus {
25   border-color: #007c9c;
26   box-shadow: 0 0 3px 4px rgb(0 124 156 / 16%);
27 }
28
29 .todo {
30   background-color: #007c9c;
31   padding: 0.3rem 0.5rem 0.4rem;
32   border-radius: 3px 3px;
33   margin: 15px 0;
34   color: white;
35   position: relative;
36
37 }
38
39 .todo span {
40   background-color: #007c9c;
41   color: #ffffff;
42   position: absolute;
43   right: 0;
44   top: 0;
45   width: 20px;
46   height: 20px;
47   text-align: center;
48   font-size: 16px;
49   cursor: pointer;
50   border-top-right-radius: 3px 3px;
51   line-height: 16px;
52 }
53
54 .todo span:hover {
55   color: silver;
56 }
57
58 small {
59   display: block;
60   color: red;
61 }

```

Untuk project ini saya tidak menggunakan CSS module, tapi cukup menempatkan semua kode CSS di file `index.css` saja. Kembali, ini lebih ke kesukaan setiap programmer.

Tahap selanjutnya, buatlah file `App.js` di dalam folder `src` dan isi dengan kode berikut:

`src\App.js`

```

1 import React, { useState } from 'react';
2 import Todo from './components/Todo';
3 import TodoForm from './components/TodoForm';
4

```

```
5  const todos = [
6    {
7      id: "01",
8      text: "Baca buku React Uncover"
9    },
10   {
11     id: "02",
12     text: "Makan siang"
13   },
14   {
15     id: "03",
16     text: "Main game"
17   }
18 ];
19
20 const App = () => {
21   const [arrayTodo, setArrayTodo] = useState(todos);
22
23   const handleDeleteClick = (e) => {
24     const newTodos = arrayTodo.filter(
25       item => item.id !== e.target.id
26     );
27     setArrayTodo(newTodos);
28   }
29
30   const handleaddTodo = (text) => {
31     const newTodos = [
32       ...arrayTodo,
33       {
34         id: new Date().getTime().toString(),
35         text: text
36       }
37     ];
38     setArrayTodo(newTodos);
39   }
40
41   return (
42     <div className="container">
43       {
44         arrayTodo.map((todo) => (
45           <Todo
46             key={todo.id}
47             id={todo.id}
48             text={todo.text}
49             onTodoClick={handleDeleteClick}
50           />
51         ))
52       }
53       <TodoForm onAddTodo={handleaddTodo} />
54     </div >
55   )
56 }
57
58 export default App;
```

Maksud dari kode ini tidak lagi kita bahas detail karena penjelasan lengkap ada di bab sebelumnya.

Di baris 1-3 terdapat perintah untuk import file 'react' serta mengimport components/Todo.js dan components/TodoForm.js. Kedua file ini akan kita buat sesaat lagi. Selain itu perintah export di baris 58 agar komponen MyApp bisa di import oleh file index.js. Sisa kode program lain hanya hasil copy-paste dari komponen MyApp di project Todo.

File Todo.js dan TodoForm.js akan kita tempatkan ke dalam folder components. Oleh karena itu kosongkan isi folder components (hapus file Belajar.js dan Tombol.js) lalu buat file Todo.js dan TodoForm.js dengan kode sebagai berikut:

src\components\Todo.js

```

1  const Todo = (props) => {
2    return (
3      <div className="todo">{props.text}
4        <span id={props.id} onClick={props.onTodoClick}>x</span>
5      </div>
6    )
7  }
8
9  export default Todo;
```

src\components\TodoForm.js

```

1  import React, { useState } from 'react';
2
3  const TodoForm = (props) => {
4    const [inputTodo, setInputTodo] = useState("");
5    const [pesanErrors, setpesanErrors] = useState(null);
6
7    const handleInputChange = (e) => {
8      setInputTodo(e.target.value)
9    }
10
11   const handleFormSubmit = (e) => {
12     e.preventDefault();
13     if (inputTodo.trim() === "") {
14       setpesanErrors("Todo tidak boleh kosong");
15     }
16     else {
17       // Input todo baru ke dalam state
18       props.onAddTodo(inputTodo);
19       // kosongkan input text
20       setInputTodo("");
21     }
22   }
23
24   return (
25     <form onSubmit={handleFormSubmit}>
26       <div>
```

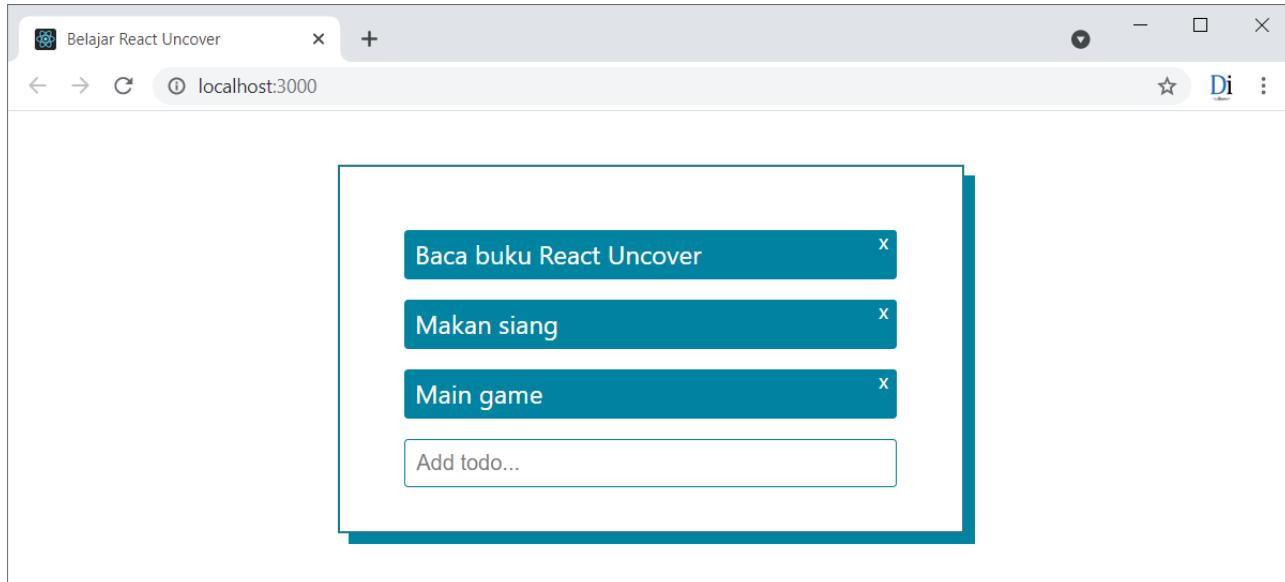
```

27     <input type="text" placeholder="Add todo..." 
28       value={inputTodo}
29       onChange={handleInputChange}
30     />
31     {pesanErrors && <small>{pesanErrors}</small>}
32   </div>
33 </form>
34 )
35 }
36
37 export default TodoForm;

```

Kembali, kode ini hanya copy-paste dari project Todo. Tambahan kode program hanya berupa perintah `import React, { useState }` di baris 1 `TodoForm.js`, serta perintah `export` di akhir setiap file.

Jika tidak ada error, maka tampilan halaman `localhost:3000` akan menampilkan project Todo:



Gambar: Tampilan project Todo di `http://localhost:3000`

Praktek ini memperlihatkan bagaimana cara membuat project menggunakan create react app.

15.14. Publish File React

Create react app berisi kumpulan tools untuk memudahkan pembuatan aplikasi React. Setelah project selesai, kita perlu memindahkan file-file tersebut ke web server agar bisa di akses oleh user. Jika ingin diakses dari internet, maka perlu ditempatkan lagi ke web hosting.

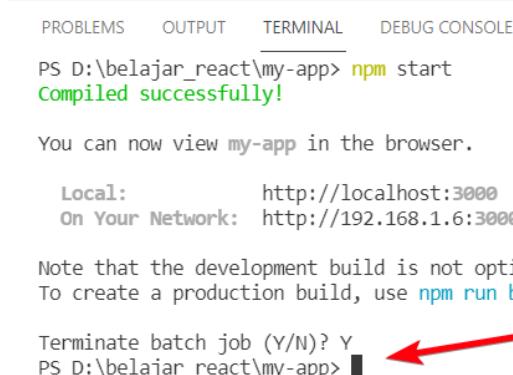
Dengan beberapa pengaturan, file yang ada di dalam folder `my-app` (hasil create react app), bisa saja langsung di upload ke web hosting. Akan tetapi ini bukan cara ideal, sebab project React umumnya terdiri dari berbagai file JS dan CSS yang saling terpisah.

Misalnya jika project memiliki 20 file JS dan 10 file CSS, maka web browser perlu 30 kali

melakukan http request, satu untuk setiap file. Tentunya sangat tidak efisien.

Teknik pembuatan web modern adalah dengan "mencompile" semua file JS dan CSS menjadi 2 file saja (1 file JS, dan 1 file CSS). Setelah di gabung, kode tersebut bisa di-minimize lagi agar lebih efisien. Create react app sudah menyediakan perintah khusus untuk keperluan ini: **npm run build**.

Sebelum praktek, pastikan project Todo sebelumnya sudah sukses berjalan. Setelah itu hentikan web server `localhost:3000` dengan cara menekan tombol CRTL + C di terminal atau cmd, dan ketik Y:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\belajar_react\my-app> npm start
Compiled successfully!

You can now view my-app in the browser.

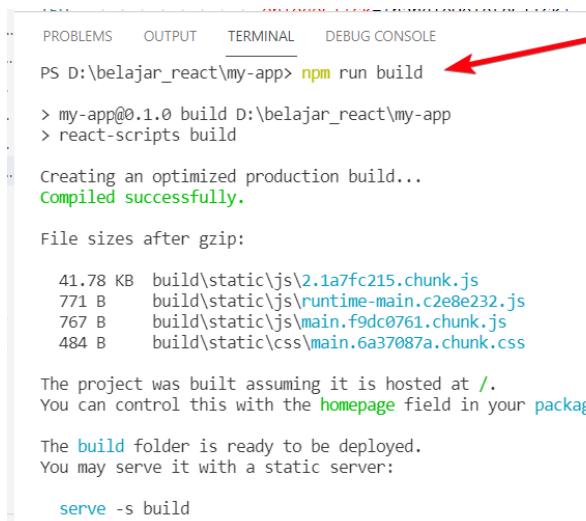
Local:          http://localhost:3000
On Your Network:  http://192.168.1.6:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

Terminate batch job (Y/N)? Y
```

Gambar: Menghentikan server localhost:3000

Cursor terminal akan kembali ke folder `D:\belajar_react\my-app`, lalu ketik perintah: `npm run build`. Terminal akan berhenti sejenak untuk memulai proses compile.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\belajar_react\my-app> npm run build
> my-app@0.1.0 build D:\belajar_react\my-app
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:
41.78 KB  build\static\js\2.1a7fc215.chunk.js
771 B     build\static\js\runtime-main.c2e8e232.js
767 B     build\static\js\main.f9dc0761.chunk.js
484 B     build\static\css\main.6a37087a.chunk.css

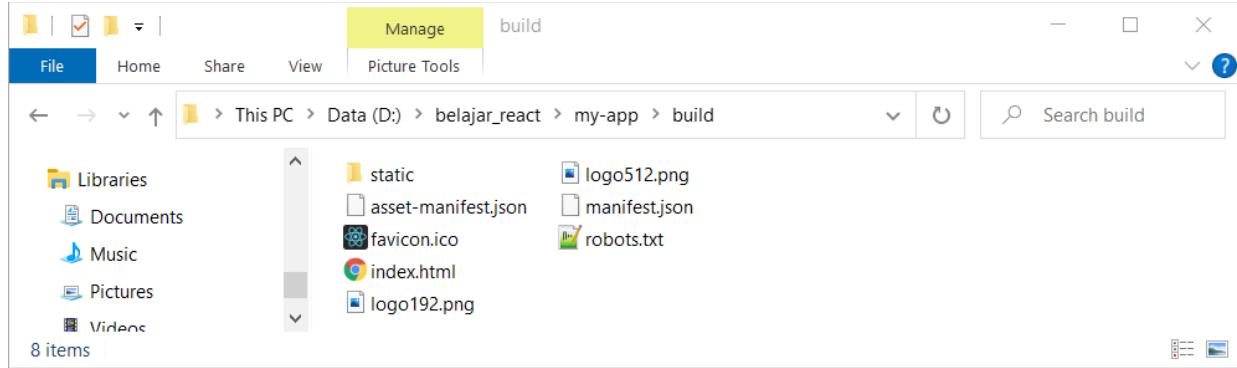
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:
  serve -s build
```

Gambar: Menjalankan perintah npm run build

Setelah selesai, di dalam folder `my-app` akan muncul satu folder baru bernama **build**. Folder ini berisi file akhir yang sudah dicompile:

Create React App



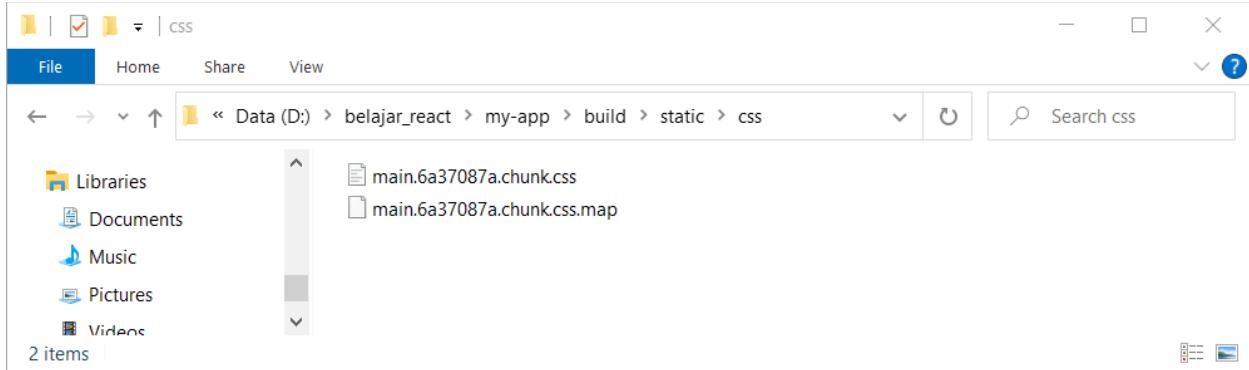
Secara garis besar, file ini berasal dari gabungan folder `my-app\public` dan `my-app\src`. File `index.html` menjadi file utama yang berisi aplikasi React dan itu juga sudah di-minify. Jika file ini dibuka di VS Code, hanya terdiri dari 1 baris panjang:

Gambar: Isi file index.html

Meskipun file `index.html` berisi kode HTML dan JS biasa, tetapi tidak bisa langsung dibuka di web browser. Cara mengaksesnya akan kita bahas sesaat lagi.

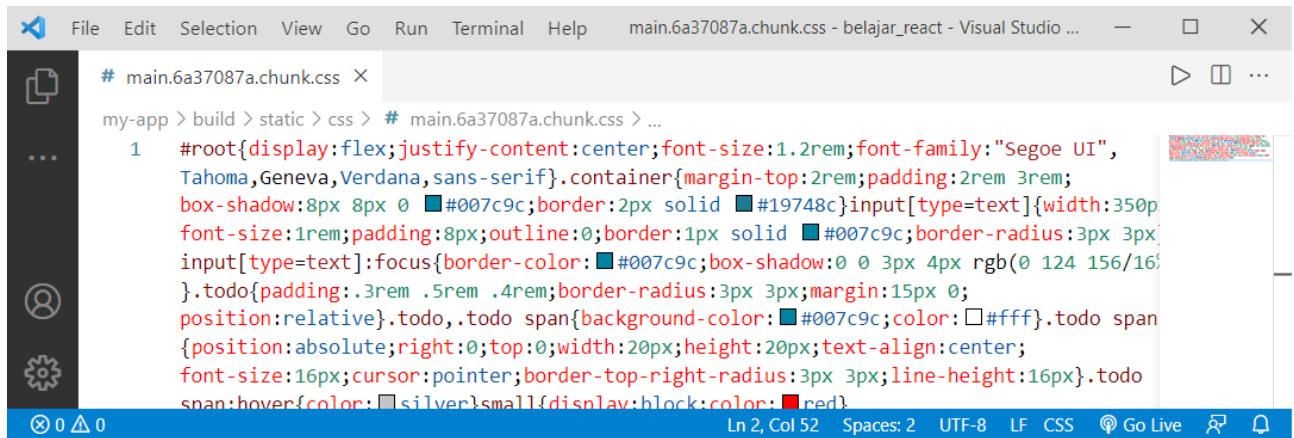
File JS dan CSS dikumpulkan ke dalam folder **static**. Sebagai contoh, silahkan buka folder ini, lalu masuk ke folder **css**. Di dalamnya terdapat 2 file: `main.6a37087a.chunk.css` dan `main.6a37087a.chunk.css.map`. Nama file yang anda temui kemungkinan besar akan berbeda karena terdapat tambahan karakter *hashing* (angka acak) oleh create react app.

Create React App



Gambar: Isi folder my-app\build\static\css

Berikut isi dari file main.6a37087a.chunk.css:



```
# main.6a37087a.chunk.css X
my-app > build > static > css > # main.6a37087a.chunk.css > ...
1 #root{display:flex;justify-content:center;font-size:1.2rem;font-family:"Segoe UI", Tahoma, Geneva, Verdana, sans-serif}.container{margin-top:2rem;padding:2rem 3rem;box-shadow:8px 8px 0 #007c9c;border:2px solid #19748c}input[type=text]{width:350px;font-size:1rem;padding:8px;outline:0;border:1px solid #007c9c;border-radius:3px 3px}input[type=text]:focus{border-color:#007c9c;box-shadow:0 0 3px 4px rgba(0, 124, 156, 16)} .todo{padding:.3rem .5rem .4rem;border-radius:3px 3px;margin:15px 0;position:relative}.todo,.todo span{background-color:#007c9c;color:#fff}.todo span{position:absolute;right:0;top:0;width:20px;height:20px;text-align:center;font-size:16px;cursor:pointer;border-top-right-radius:3px 3px;line-height:16px}.todo span:hover{color:silver}small{display:block;color:red}
```

Gambar: Isi dari file main.6a37087a.chunk.css

Kode CSS yang ada nyaris tidak bisa dibaca karena sudah *minify* oleh perintah `npm run build`. Begitu juga dengan isi folder `my-app\build\js\` yang terdiri dari beberapa file JavaScript yang juga sudah *minify*.

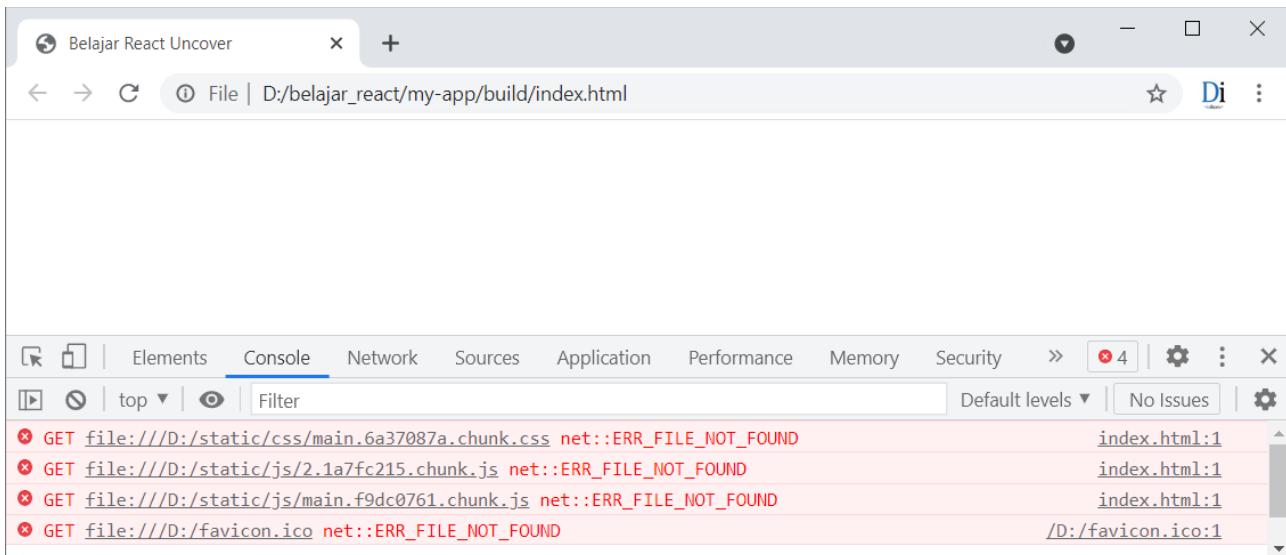
Intinya, semua file yang ada di folder **build** memang bukan untuk dibaca atau di edit, tapi sudah menjadi file akhir yang tinggal dipindahkan ke web server.

Mengatur Path Hasil Build

Apabila kita membuka file `build\index.html` langsung di web browser dengan cara di double click, tidak akan tampak hasil apa-apa (hanya layar putih saja). Ini karena perintah `npm run build` secara default membuat path ke root folder untuk semua file assets.

Hasil pesan error di tab console memperlihatkan masalah ini:

Create React App



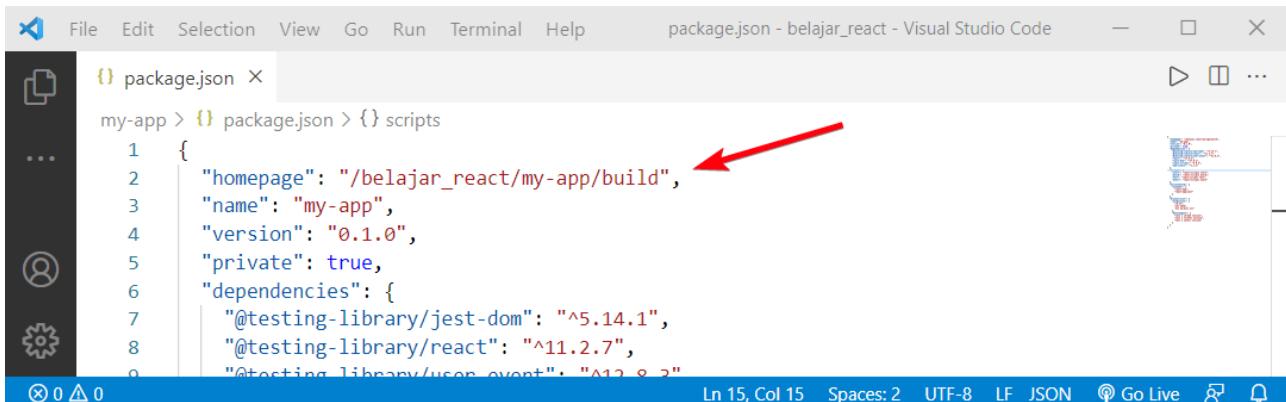
Gambar: Error file not found

Perhatikan pesan error yang tampil. Web browser "komplain" karena tidak bisa menemukan file CSS dan CSS di D:/static/, sebab semua file tersebut ada di D:/belajar_react/my-app/build/.

Solusi dari masalah kita adalah, perintah npm run build perlu instruksi tambahan agar pathnya sesuai. Caranya, buka file my-app\package.json, lalu tambah 1 baris berikut:

```
"homepage": "/belajar_react/my-app/build",
```

Instruksi ini berfungsi untuk mengubah path default aplikasi React:



Gambar: Tambah pengaturan "homepage" di file package.json

Save file package.json dan jalankan ulang perintah npm run build (1). Pastikan ada keterangan *The project was built assuming it is hosted at /belajar_react/my-app/build/* (2).

Create React App

```
PS D:\belajar_react\my-app> npm run build
> my-app@0.1.0 build D:\belajar_react\my-app
> react-scripts build

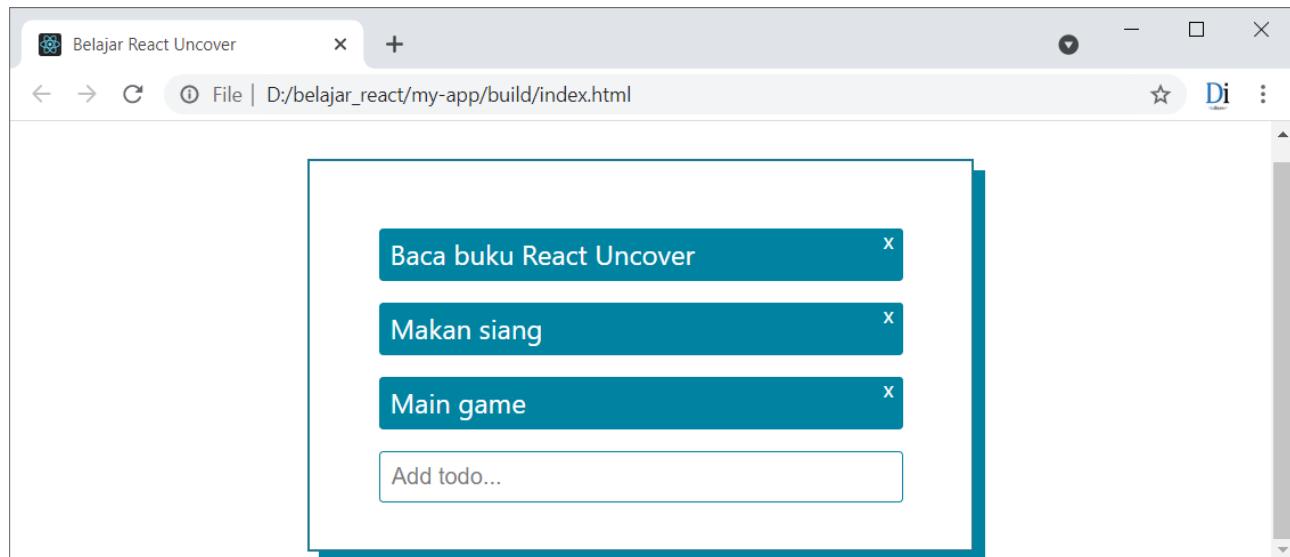
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:
41.78 KB  build\static\js\2.1a7fc215.chunk.js
793 B (+22 B)  build\static\js\runtime-main.8da3cf15.js
767 B  build\static\js\main.f9dc0761.chunk.js
484 B  build\static\css\main.6a37087a.chunk.css

The project was built assuming it is hosted at /belajar_react/my-app/build/.
You can control this with the homepage field in your package.json.
oyed.
```

Gambar: Jalankan ulang perintah npm run build

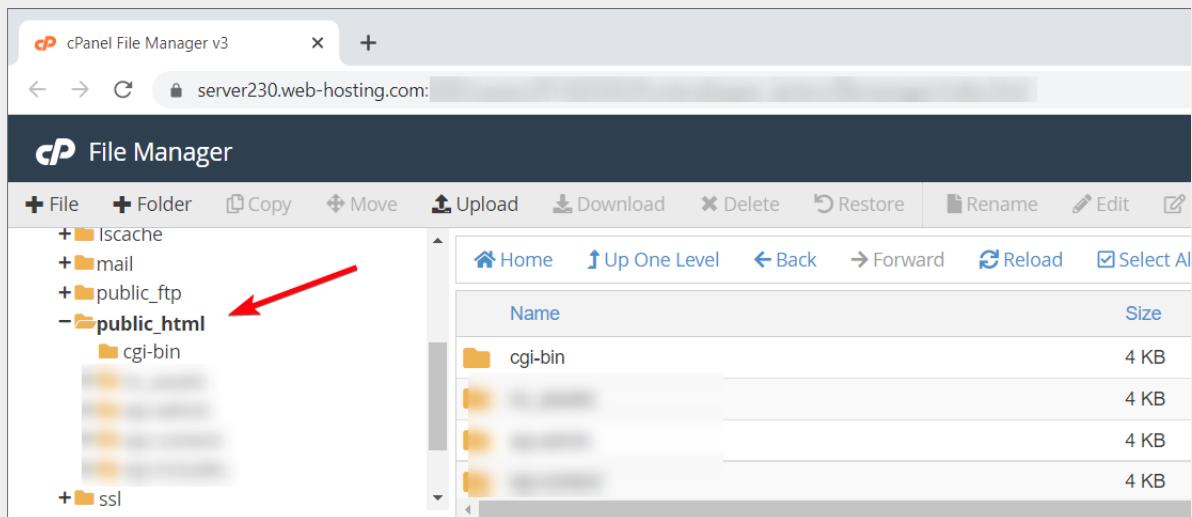
Setelah proses selesai, jalankan file `build\index.html` dengan cara double click, dan aplikasi Todo akan tampil di web browser:



Gambar: Aplikasi Todo hasil build sukses di akses

Dari praktek ini bisa di simpulkan kalau kita perlu mengatur lokasi *path* dari file yang di-compile dengan perintah `npm run build`, terutama jika aplikasi tersebut tidak berada langsung di bawah *root folder*.

Bagi yang sudah pengalaman dengan web hosting, biasanya file untuk website harus diletakkan di dalam folder **www** atau **public_html** yang ada di cPanel:



Gambar: Tampilan file manager di cPanel web hosting

Jika isi folder `build` tempatkan langsung ke dalam folder ini (tidak berada di sub-folder), maka itulah yang dimaksud dengan "langsung di bawah root folder". Ketika user membuka misalnya `www.websiteku.com`, aplikasi Todo akan langsung tampil.

Akan tetapi jika aplikasi Todo ingin di tempatkan ke alamat `www.websiteku.com/todo`, maka perlu tambahan pengaturan "homepage": "/todo" di file `package.json` agar alamat path sesuai dengan sub-folder tersebut.

Menjalankan Hasil Build Dari Static Server

Selain menjalankan hasil `npm run build` dengan cara pengubah path, sebenarnya ada cara yang lebih praktis, yakni dengan menggunakan static server bawaan Nodejs. Cara ini bahkan disarankan saat perintah `npm run build` selesai di proses:

```
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:
  serve -s build
```

The screenshot shows a terminal window with deployment instructions for a Create React App. It includes a note about the project being built for root ('/'), a link to control it via `package.json`, and a command to serve the build folder using the `serve` command with the `-s` option. A red arrow points to the `serve` command. The terminal also shows icons for user and gear, and the command `PS D:\belajar_react\my-app>`.

Gambar: Create react app menyarankan memakai static server

Jika ingin mencoba, silahkan hapus tambahan pengaturan "homepage": "/belajar_react/my-

Create React App

app/build" dari file package.json, lalu jalankan ulang npm run build agar pengaturannya kembali ke kondisi awal.



```
PS D:\belajar_react\my-app> npm run build
> my-app@0.1.0 build D:\belajar_react\my-app
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:
41.78 KB  build\static\js\2.1a7fc215.chunk.js
771 B (-22 B) build\static\js\runtime-main.c2e8e232.js
767 B  build\static\js\main.f9dc0761.chunk.js
484 B  build\static\css\main.6a37087a.chunk.css
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
```

Gambar: Alamat path assets sudah kembali ke pengaturan awal di / (root folder)

Kemudian install static server dengan perintah berikut:

```
npm install -g serve
```

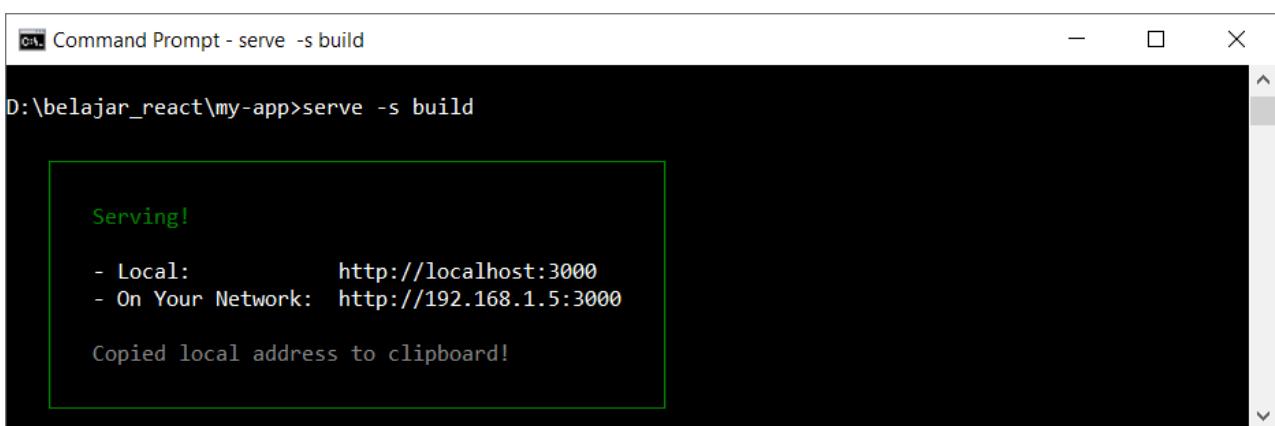


```
PS D:\belajar_react\my-app> npm install -g serve ←
C:\Users\Andre\AppData\Roaming\npm\serve -> C:\Users\Andre\AppData\Roaming\npm\node_modules\serve\bin\serve.js
+ serve@13.0.2
added 90 packages from 40 contributors in 16.234s
PS D:\belajar_react\my-app>
```

Gambar: Install server npm

Setelah itu buka jendela cmd terpisah (jangan pakai terminal bawaan VS Code), lalu masuk ke folder D:\belajar_react\my-app dan jalankan perintah:

```
serve -s build
```

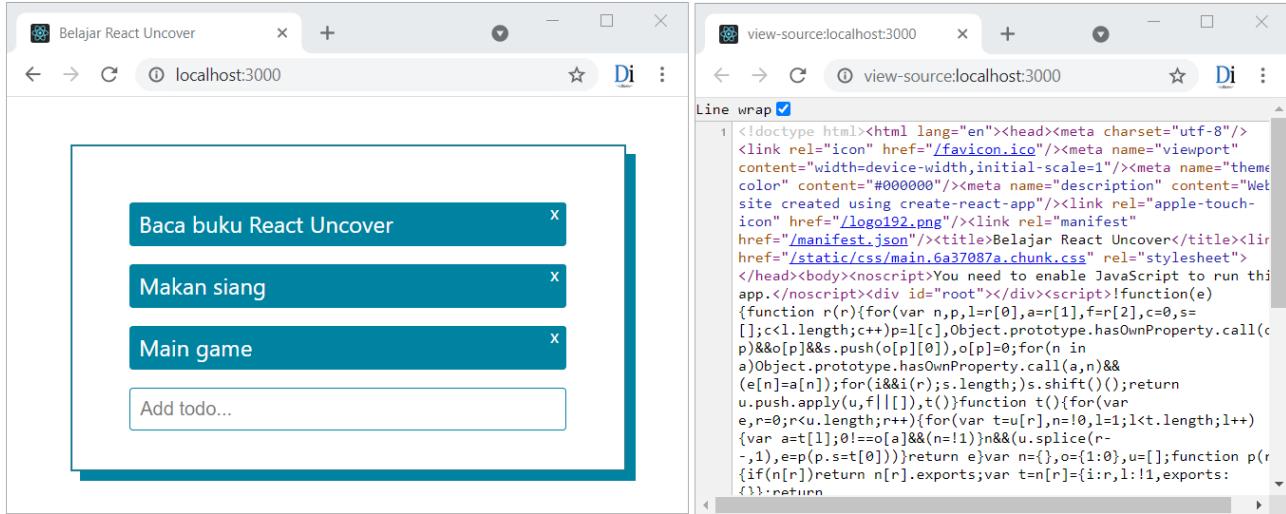


```
Command Prompt - serve -s build
D:\belajar_react\my-app>serve -s build

Serving!
- Local:          http://localhost:3000
- On Your Network: http://192.168.1.5:3000
Copied local address to clipboard!
```

Gambar: Menjalankan server nodejs dengan perintah serve -s build

Sekarang file build sudah bisa diakses dari alamat <http://localhost:3000>:



Gambar: Mengakses file build dari localhost:3000

Salah satu bukti bahwa ini merupakan hasil proses build, bisa di cek dari source code web browser yang hanya tampil 1 baris panjang.

Nomor port yang dipakai oleh static server bisa berbeda-beda terutama jika port 3000 sudah dipakai oleh server normal (hasil `npm start`). Jendela cmd juga harus selalu aktif agar server tetap berjalan. Jika jendela cmd ini di tutup, server juga akan berhenti.

Menjalankan Static Server dari Terminal VS Code

Sebelumnya saya sempat tulis "jangan pakai terminal bawaan VS Code" untuk menjalankan perintah `serve -s build`.

Alasannya karena terdapat mekanisme proteksi dari Windows Powershell yang membatasi perintah tersebut. Windows Powershell sendiri merupakan aplikasi terminal default yang dipilih VS code di OS Windows.

Ketika menjalankan perintah `serve -s build` di terminal VS Code, saya mendapati error:

File C:\Users\Andre\AppData\Roaming\npm\serve.ps1 cannot be loaded because running scripts is disabled on this system. For more information, see about_Execution_Policies at <https://go.microsoft.com/fwlink/?LinkID=135170>.

```

PS D:\belajar_react\my-app> serve -s build
serve : File C:\Users\Andre\AppData\Roaming\npm\serve.ps1 cannot be loaded because running scripts is
disabled on this system. For more information, see about_Execution_Policies at https://go.microsoft.com
/fwlink/?LinkID=135170.
At line:1 char:1
+ serve -s build
+ ~~~~~
    + CategoryInfo          : SecurityError: () [], PSNotSupportedException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS D:\belajar_react\my-app>

```

Gambar: Error ketika menjalankan perintah `serve -s build`

Error ini berkaitan dengan pengaturan ExecutionPolicy bawaan Windows PowerShell. Pengaturan ini membatasi hak akses user untuk mengeksekusi script tertentu. Jika kita tetap ingin menjalankan perintah `serve -s build` dari terminal VS Code, pengaturan Execution Policy ini harus diubah.

Hak akses user bisa dilihat dengan perintah:

`Get-ExecutionPolicy -list`



```
PS D:\belajar_react\my-app> Get-ExecutionPolicy -list
Scope ExecutionPolicy
-----
MachinePolicy      Undefined
UserPolicy         Undefined
Process           Undefined
CurrentUser        Undefined
LocalMachine       Undefined
```

Gambar: Hasil perintah `Get-ExecutionPolicy -list`

Terlihat semua user memiliki hak akses `Undefined`. Untuk mengubahnya, jalankan perintah berikut (1):

`Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`,

Lalu cek kembali hak akses dengan perintah (2):

`Get-ExecutionPolicy -list`



```
PS D:\belajar_react\my-app> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser ①
PS D:\belajar_react\my-app> Get-ExecutionPolicy -list ②
Scope ExecutionPolicy
-----
MachinePolicy      Undefined
UserPolicy         Undefined
Process           Undefined
CurrentUser        RemoteSigned ③
LocalMachine       Undefined
```

Gambar: Mengubah hak akses untuk CurrentUser

Hasilnya, `ExecutionPolicy` untuk `CurrentUser` sudah berubah jadi `RemoteSigned`. Sekarang kita sudah bisa menjalankan perintah `serve -s build`:

Create React App

PS D:\belajar_react\my-app> **serve** -s build

Serving!

- Local: http://localhost:3000
- On Your Network: http://192.168.1.6:3000

Copied local address to clipboard!

Go Live ⌂ ⌂

Gambar: Sukses menjalankan static server

Setelah itu, aplikasi sudah bisa diakses dari alamat `http://localhost:3000`. Untuk menghentikan static server, bisa dengan menekan tombol CTRL + C di terminal:

- Local: http://localhost:3000
- On Your Network: http://192.168.1.6:3000

Copied local address to clipboard!

INFO: Gracefully shutting down. Please wait...
PS D:\belajar_react\my-app>

Go Live ⌂ ⌂

Gambar: Menghentikan static server

Kembali ke ExecutionPolicy yang baru saja kita tukar, jika anda ingin mengembalikannya ke pengaturan awal, bisa jalankan perintah berikut:

`Set-ExecutionPolicy Undefined -Scope CurrentUser`

Lalu cek lagi dengan perintah:

`Get-ExecutionPolicy -list:`

PS D:\belajar_react\my-app> **Set-ExecutionPolicy** Undefined -Scope CurrentUser
PS D:\belajar_react\my-app> **Get-ExecutionPolicy** -list

Scope	ExecutionPolicy
MachinePolicy	Undefined
UserPolicy	Undefined
Process	Undefined
CurrentUser	Undefined
LocalMachine	Undefined

PS D:\belajar_react\my-app>

Go Live ⌂ ⌂

Gambar: Mengembalikan pengaturan ExecutionPolicy untuk CurrentUser

Hasilnya, pengaturan ExecutionPolicy sudah kembali menjadi `Undefined`.

Sepanjang bab ini kita telah lihat cara penggunaan create react app untuk membuat aplikasi React. Ini memang tidak wajib, karena seperti yang sudah kita praktekkan sejak awal buku, aplikasi React tetap bisa dibuat dengan cara biasa (langsung ditulis ke dalam file JS).

Namun untuk project yang lebih kompleks, create react app menyediakan berbagai tools yang akan memudahkan pengelolaan kode program. Termasuk perintah `npm run build` untuk menggabungkan dan me-minify file assets.

Berikutnya, kita akan masuk ke studi kasus yang lebih menantang lagi, yakni membuat mini project **CRUD Mahasiswa**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

16. Mini Project: CRUD Mahasiswa

Sebelumnya kita sudah membuat mini project **Todo**, yang meskipun sederhana tapi sudah berisi fitur interaktif React. Kali ini kita akan coba membuat aplikasi yang lebih menarik lagi, yakni membuat CRUD Mahasiswa.

Dalam programming, **CRUD** merupakan singkatan dari **Create Read Update dan Delete**. Yakni aplikasi yang berisi pengolahan data. Biasanya CRUD butuh database sebagai media penyimpanan. Akan tetapi untuk menyingkat pembahasan dan agar kita bisa lebih fokus ke kode React, CRUD ini hanya disimpan di memory saja.

Berikut tampilan aplikasi yang akan kita buat:

NIM	Nama	Jurusan	Asal Provinsi	
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	<button>Edit</button> <button>Hapus</button>
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	<button>Simpan</button> <button>Batal</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Simpan</button> <button>Batal</button>
20010790	Friska Ramadhan	Ilmu Komputer	Kalimantan Barat	<button>Edit</button> <button>Hapus</button>
20010710	Sari Citra Lestari	Sistem Informasi	Nusa Tenggara Barat	<button>Tambah</button>

Nim sudah dipakai

Jurusan tidak boleh kosong

Tampilan akhir dari mini project "CRUD Mahasiswa"

Pada saat penyusunan buku **React Uncover**, saya memutuskan ingin membagi 2 bahasan, yakni konsep dasar React, dan studi kasus atau mini project. Meskipun masih banyak materi advanced React yang belum sempat di bahas, tapi dengan apa yang sudah kita pelajari, sudah bisa membuat berbagai project sederhana.

Karena itulah mulai dari bab ini hingga akhir buku nanti, kita akan ganti fokus

pembahasan ke pembuatan berbagai mini project. Ini sangat bermanfaat untuk menguji pemahaman serta melihat bagaimana penerapan konsep React dalam aplikasi yang sebenarnya.

16.1. Persiapan Awal

Untuk project ini, saya akan tulis kode React dalam bentuk *functional component* serta menggunakan **create react app**. Idealnya kita bisa mulai dari menginstall *create react app* baru, akan tetapi untuk menghemat bandwith, tidak masalah jika ingin memakai folder **my-app** hasil praktek bab sebelumnya.

Agar tidak bentrok dengan kode yang akan di buat, silahkan hapus folder **build** (hasil perintah `npm run build` dari bab sebelumnya), dan kosongkan isi folder **src** (hapus semua file yang ada).

Setelah itu buka file `public\index.html` dan edit beberapa baris berikut:

`public\index.html`

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="utf-8" />
6      <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
7      <meta name="viewport" content="width=device-width, initial-scale=1" />
8      <meta name="theme-color" content="#000000" />
9      <meta name="description" content="Aplikasi CRUD Mahasiswa - React Uncover" />
10     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
11     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
12     <title>CRUD Mahasiswa</title>
13 </head>
14
15 <body>
16     <noscript>You need to enable JavaScript to run this app.</noscript>
17     <div id="root"></div>
18 </body>
19
20 </html>
```

Perubahan ada di baris 2, 9 dan 12. Ini lebih ke tampilan saja dan tidak berpengaruh ke kode React. Di baris 2 saya menukar atribut `lang="en"` menjadi `lang="id"`, lalu di baris 9 menukar isi atribut `content`, serta di baris 12 mengganti isi tag `<title>`.

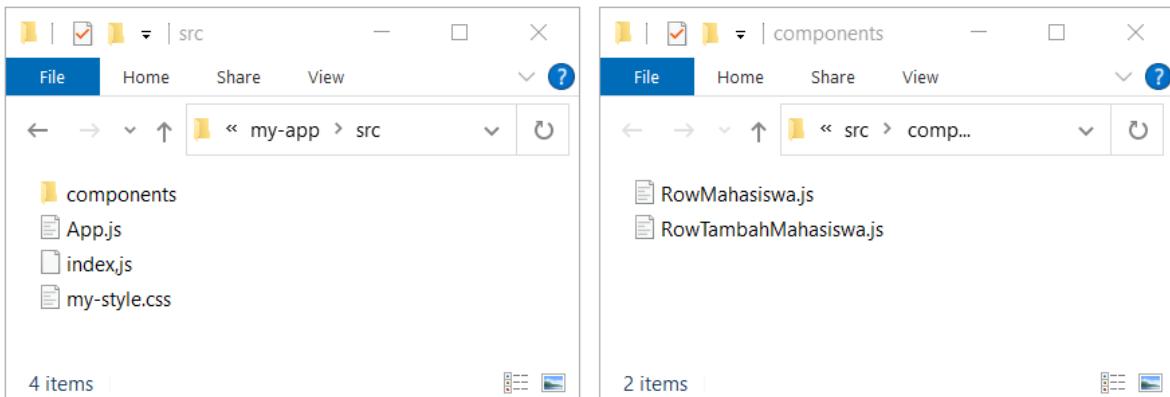
Lalu buat 3 file berikut ke dalam folder **src** (isinya dikosongkan saja dulu):

- `index.js`
- `App.js`
- `my-style.css`

Kemudian buat juga folder **components** di dalam **src** dan isi dengan 2 file berikut:

- RowMahasiswa.js
- RowTambahMahasiswa.js

Kelima file ini belum berisi kode apapun, hanya persiapan saja.



Gambar: Isi folder my-app\src dan my-app\src\components

Berdasarkan praktek dari bab sebelumnya, bisa ditebak bahwa file `src\index.js` akan menjadi file awal React. Disinilah terdapat method `ReactDOM.createRoot().render()` yang akan mengakses tag `<div id="root">` milik `public\index.html`.

Kemudian file `App.js` akan berisi kode utama aplikasi CRUD mahasiswa. Di dalamnya berisi komponen `App` yang akan mengakses komponen `RowMahasiswa` dan `RowTambahMahasiswa` di folder **components**. Terakhir, file `my-style.css` dipakai untuk menampung kode CSS .

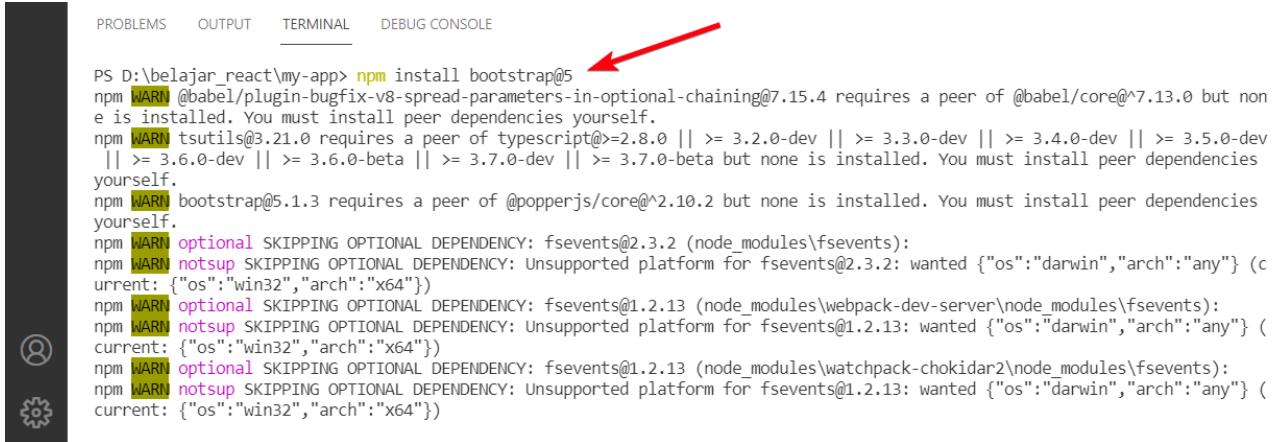
Dalam project ini saya juga ingin menggunakan framework CSS **Bootstrap**, agar tampilan aplikasi kita menjadi lebih cantik.

Terdapat beberapa cara untuk memakai file Bootstrap ke dalam React, cara pertama adalah dengan mendownload manual file `bootstrap.css` dari web resminya di getbootstrap.com, lalu simpan ke dalam folder **src** untuk diimport dari file `index.js`.

Cara kedua yang lebih elegan adalah dengan menginstall Bootstrap sebagai *dependency* menggunakan **npm**. Inilah yang akan kita coba. Silahkan buka terminal, masuk ke folder **my-app**, lalu jalankan perintah berikut:

```
npm install bootstrap@5
```

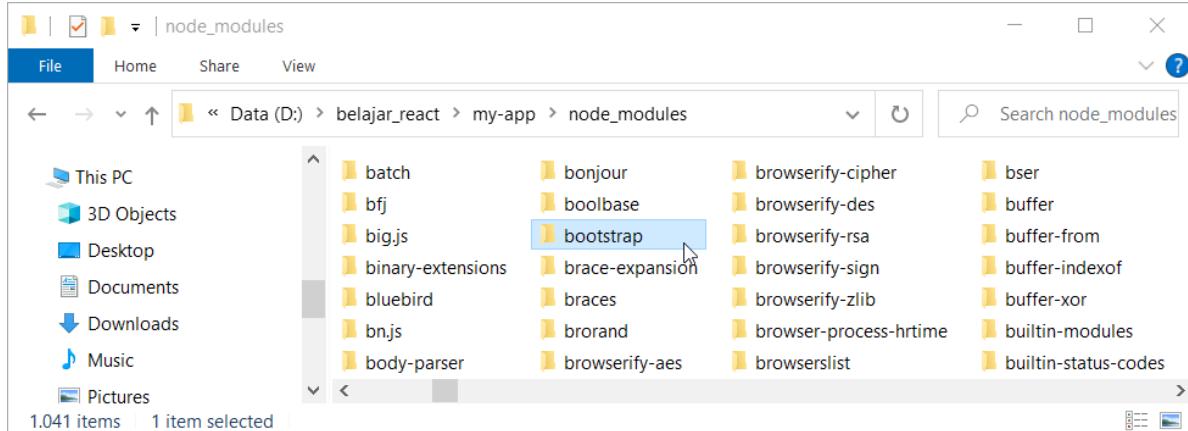
Perintah ini akan mendownload file Bootstrap dan menyimpannya di dalam folder **node_modules**:



```
PS D:\belajar_react\my-app> npm install bootstrap@5
npm WARN @babel/plugin-bugfix-v8-spread-parameters-in-optional-chaining@7.15.4 requires a peer of @babel/core@^7.13.0 but none is installed. You must install peer dependencies yourself.
npm WARN tsutils@3.21.0 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-beta but none is installed. You must install peer dependencies yourself.
npm WARN bootstrap@5.1.3 requires a peer of @popperjs/core@^2.10.2 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules\watchpack-chokidar2\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
```

Gambar: Menginstall Bootstrap dari terminal VS Code

Saat menjalankan perintah ini, kemungkinan akan ada beberapa warning dari npm (seperti tampilan di atas). Ini bisa kita abaikan selama tidak ada pesan error. Untuk membuktikan file Bootstrap sudah ada, bisa cek ke folder `my-app\node_modules\bootstrap`:



Gambar: Folder bootstrap di dalam node_modules

Folder bootstrap ini tidak perlu kita utak-atik. Sekarang silahkan buka file `src\index.js`, lalu isi dengan kode berikut:

```
src\index.js

1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import 'bootstrap/dist/css/bootstrap.min.css';
5 import './my-style.css';
6
7 ReactDOM.createRoot(document.getElementById('root'))
8 .render(
9   <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
```

Perintah import di baris 1 dan 2 berguna untuk mengakses file utama library React, yakni `react` dan `react-dom/client`.

Lalu di baris 3 terdapat perintah untuk import komponen `App` dari file `src\App.js` yang sudah kita siapkan sebelumnya. Kode untuk komponen `App` sendiri akan dibuat sesaat lagi.

Di baris 4 giliran perintah import file Bootstrap. Secara default, jika file yang di-import tidak diawali tanda `./`, maka `create react app` akan mencarinya ke dalam folder `node_modules`.

Kemudian diikuti perintah untuk mengimport file CSS `my-style.css` di baris 5. Dalam project ini saya hanya memakai 1 file CSS untuk semua komponen.

Terakhir antara baris 7 – 12 terdapat method `ReactDOM.createRoot().render()` yang akan merender komponen `<App />` ke dalam tag `<div id="root">` di file `public\index.html`.

Setelah itu, isi juga kode berikut ke dalam file `my-style.css`:

```
src\my-style.css

1  /* Agar container tidak terlalu lebar */
2  @media (min-width: 1200px) {
3      .container {
4          max-width: 1140px !important;
5      }
6  }
7
8  table{
9      table-layout: fixed;
10 }
11
12 td {
13     vertical-align: middle;
14 }
15
16 small {
17     display: block;
18     color: red;
19 }
```

Kode CSS yang diperlukan tidak terlalu banyak, sebab kita akan memakai Bootstrap untuk membuat design tampilan.

Perintah di baris 2–6 berfungsi untuk menonaktifkan lebar container xxl milik Bootstrap 5 yang menurut saya terlalu lebar. Ini lebih ke preferensi pribadi saja, penjelasan lebih lanjut tentang apa itu container xxl Bootstrap, tersedia di buku [Bootstrap 5 Uncover](#)²².

Antara baris 8 – 19 terdapat kode untuk merapikan tampilan tag `<table>`, `<td>` dan `<small>`. Khusus untuk selector `small`, nantinya akan dipakai ketika menampilkan pesan error form.

22. <https://www.duniaIlkom.com/bootstrap-uncover-panduan-belajar-bootstrap/>

Mengenal library React Bootstrap

Jika kita ingin menggunakan framework CSS Bootstrap ke dalam kode React, sebenarnya ada library khusus bernama **React Bootstrap** (react-bootstrap.github.io²³).

Library React Bootstrap menyediakan kode yang lebih efisien untuk dipakai ke dalam React. Misalnya untuk membuat tampilan alert, dengan cara biasa butuh kode berikut:

```
<div class="alert alert-success alert-dismissible fade show"></div>
```

Namun jika menggunakan React Bootstrap, bisa ditulis dengan:

```
<Alert show={show} variant="success">
```

Selain kode yang lebih singkat, integrasi fitur interaktif Bootstrap (yang butuh JavaScript) juga akan lebih mudah jika menggunakan React Bootstrap.

Namun karena di sini kita hanya membuat project sederhana, saya akan tetap menggunakan file Bootstrap "native". Materi React Bootstrap lebih pas menjadi bab tersendiri di buku React in Depth #1 yang mudah-mudahan nanti bisa menyusul.

16.2. Menampilkan Data Mahasiswa

Sekarang kita akan masuk ke kode program untuk menampilkan data mahasiswa. Silahkan buka file `src\App.js`, lalu isi dengan kode berikut:

```
src\App.js

1 import React, { useState } from 'react';
2
3 // Data awal tabel mahasiswa
4 const arrMahasiswas = [
5   {
6     nim: "18010245",
7     nama: "Eka Putra",
8     jurusan: "Teknik Informatika",
9     asalProvinsi: "DKI Jakarta"
10    },
11    {
12      nim: "19010214",
13      nama: "Lisa Permata",
14      jurusan: "Sistem Informasi",
15      asalProvinsi: "Sumatera Barat"
16    },
17    {
18      nim: "20010710",
19      nama: "Rudi Setiawan",
20      jurusan: "Ilmu Komputer",
21      asalProvinsi: "Jawa Tengah"
```

23. <https://react-bootstrap.github.io>

```

22     },
23     {
24       nim: "20010790",
25       nama: "Friska Ramadhani",
26       jurusan: "Ilmu Komputer",
27       asalProvinsi: "Kalimantan Barat"
28   }
29 ];
30
31 const App = () => {
32   const [mahasiswas, setMahasiswas] = useState(arrMahasiswas);
33
34   return (
35     <div className="container mt-5">
36
37       <div className="row mt-5">
38         <div className="col">
39           <h1 className="text-center">Tabel Mahasiswa</h1>
40
41         <table className="table mt-4">
42           <thead>
43             <tr>
44               <th>NIM</th>
45               <th>Nama</th>
46               <th>Jurusan</th>
47               <th>Asal Provinsi</th>
48               <th></th>
49             </tr>
50           </thead>
51           <tbody>
52             {
53               mahasiswas.map((mahasiswa) =>
54                 <tr key={mahasiswa.nim}>
55                   <td>{mahasiswa.nim}</td>
56                   <td>{mahasiswa.nama}</td>
57                   <td>{mahasiswa.jurusan}</td>
58                   <td>{mahasiswa.asalProvinsi}</td>
59                   <td>
60                     <button className="btn btn-secondary me-2">Edit</button>
61                     <button className="btn btn-danger">Hapus</button>
62                   </td>
63                 </tr>
64             )
65           }
66
67         <tr>
68           <td colSpan="5">
69             <form>
70               <div className="row row-cols-5 g-3">
71                 <div className="col">
72                   <input type="text" className="form-control"
73                     name="nim" placeholder="00000000" />
74                 </div>
75                 <div className="col">
76                   <input type="text" className="form-control"

```

```

77             name="nama" placeholder="Fulan Fulana" />
78         </div>
79         <div className="col">
80             <input type="text" className="form-control"
81                 name="jurusan" placeholder="Sistem Informasi" />
82         </div>
83         <div className="col">
84             <input type="text" className="form-control"
85                 name="asalProvinsi" placeholder="DKI Jakarta" />
86         </div>
87         <div className="col">
88             <button type="submit" className="btn btn-primary">
89                 Tambah</button>
90         </div>
91     </div>
92     </form>
93   </td>
94 </tr>
95 </tbody>
96 </table>
97 </div>
98
99 </div>
100 </div>
101 )
103 }
104
105 export default App;

```

NIM	Nama	Jurusan	Asal Provinsi	
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	<button>Edit</button> <button>Hapus</button>
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	<button>Edit</button> <button>Hapus</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Edit</button> <button>Hapus</button>
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat	<button>Edit</button> <button>Hapus</button>
00000000	Fulan Fulana	Sistem Informasi	DKI Jakarta	<button>Tambah</button>

Gambar: Menampilkan data mahasiswa

Di baris 1 terdapat perintah import react serta membuat alias untuk `useState`. Ini diperlukan karena nantinya kita butuh mengakses `useState` hook.

Selanjutnya antara baris 4-29, konstanta `arrMahasiswa` diisi 4 object data mahasiswa. Setiap object memiliki property `nim`, `nama`, `jurusan` dan `asalProvinsi`. Inilah data awal yang akan kita pakai dalam project CRUD mahasiswa. Nilai nim sudah unik sehingga bisa dipakai sebagai identitas dari setiap mahasiswa (tidak perlu property `id` terpisah).

Komponen `App` dimulai dari baris 31, yang diawali dengan pembuatan state `mahasiswa` di baris 32. Nilai awal state diambil dari konstanta `arrMahasiswa`.

Kemudian di baris 34 masuk ke kode JSX. Karena file Bootstrap sudah diimport dalam file `index.js`, maka komponen `App` sudah bisa mengakses class CSS milik Bootstrap seperti `.container`, `.mt-5`, `.col` dan `.row`. Di baris 35-38, terdapat kode untuk membuat `container` dan struktur grid Bootstrap.

Jika anda belum pernah belajar Bootstrap, boleh abaikan nilai atribut `className` di setiap tag HTML. Class Bootstrap ini hanya untuk mempercantik tampilan saja, tidak berpengaruh ke materi React.

Pada baris 39 terdapat tag `<h1>` yang berisi teks **Tabel Mahasiswa**, yang diikuti tag `<table>` antara baris 41-96. Dalam tabel inilah kita akan menampilkan seluruh data mahasiswa beserta tombol untuk menambah, mengedit dan menghapus data.

Di baris 53 terdapat method `mahasiswa.map()` untuk menampilkan semua data yang tersimpan di dalam state `mahasiswa`. Dalam setiap iterasi, akses nilai `mahasiswa.nim`, `mahasiswa.nama`, `mahasiswa.jurusan` dan `mahasiswa.asalProvinsi`. Tidak lupa, atribut `key` untuk tag `<tr>` diambil dari nilai `mahasiswa.nim`.

Tombol "Edit" dan "Hapus" sudah saya siapkan di sisi kanan setiap baris tabel. Kedua tombol belum berfungsi karena kita belum menulis event apapun.

Setelah semua data mahasiswa tampil, terdapat form di akhir tabel (baris 69-92). Ini adalah form untuk menambah data mahasiswa. Idenya mirip seperti project Todo, dimana setelah data mahasiswa diinput, form akan kembali kosong dan data langsung masuk ke state `mahasiswa`. Fitur ini akan kita buat sesaat lagi.

Membuat Komponen RowMahasiswa

Agar pengelolaan kode program menjadi lebih mudah, saya ingin memecah `App` menjadi komponen yang lebih kecil. Ini tidak harus dilakukan, lebih ke kesukaan setiap programmer saja. Akan tetapi memecah komponen menjadi best-practice yang sangat baik untuk diikuti.

Bagian pertama yang ingin saya pisah adalah tag `<tr>` yang ada di dalam method `mahasiswa.map()`. Baris ini terus diulang sebanyak data mahasiswa dan nantinya akan memiliki kode yang

cukup panjang, sehingga sangat pas untuk dipisah. Silahkan edit baris 52-65 dari sebelumnya berisi kode berikut:

src\App.js

```
1  {
2    mahasiswas.map((mahasiswa) =>
3      <tr key={mahasiswa.nim}>
4        <td>{mahasiswa.nim}</td>
5        <td>{mahasiswa.nama}</td>
6        <td>{mahasiswa.jurusan}</td>
7        <td>{mahasiswa.asalProvinsi}</td>
8        <td>
9          <button className="btn btn-secondary me-2">Edit</button>
10         <button className="btn btn-danger">Hapus</button>
11       </td>
12     </tr>
13   )
14 }
```

Menjadi:

src\App.js

```
1  {
2    mahasiswas.map((mahasiswa) =>
3      <RowMahasiswa
4        key={mahasiswa.nim}
5        mahasiswa={mahasiswa}
6      />
7    )
8  }
```

Di sini saya mengirim data setiap mahasiswa sebagai *props* `mahasiswa` ke dalam komponen `RowMahasiswa`. Karena tag `<RowMahasiswa>` ada di dalam perulangan `map()`, maka jika state `mahasiswas` berisi 4 object, komponen `RowMahasiswa` akan dipanggil sebanyak 4 kali juga.

File untuk komponen `RowMahasiswa` sudah kita siapkan, yakni di folder `components`. Sebelum masuk ke file tersebut, silahkan tambah perintah `import` di baris awal file `src\App.js`:

```
import RowMahasiswa from './components/RowMahasiswa';
```

Ini harus ditulis agar komponen `App` bisa mengakses komponen `RowMahasiswa`.

Masuk ke file `components\RowMahasiswa.js`, isi dengan kode berikut:

src\components\RowMahasiswa.js

```
1  const RowMahasiswa = (props) => {
2    return (
3      <tr>
4        <td>{props.mahasiswa.nim}</td>
5        <td>{props.mahasiswa.nama}</td>
```

```
6   <td>{props.mahasiswa.jurusan}</td>
7   <td>{props.mahasiswa.asalProvinsi}</td>
8   <td>
9     <button className="btn btn-secondary me-2">Edit</button>
10    <button className="btn btn-danger">Hapus</button>
11  </td>
12 </tr>
13 )
14 }
15
16 export default RowMahasiswa;
```

Di sini kita hanya memindahkan kode yang sama dari komponen App, yakni membuat sebuah baris tabel dari tag `<tr>` untuk mengakses data `nim`, `nama`, `jurusan`, `asalProvinsi` dan 2 buah tombol untuk Edit dan Hapus. Semua data mahasiswa ini diambil dari `props.mahasiswa`.

Save file `RowMahasiswa.js` dan pastikan tampilan aplikasi CRUD mahasiswa tetap seperti sebelumnya.

Membuat Komponen RowTambahMahasiswa

Bagian kedua yang ingin saya pisah dari komponen App adalah form di bagian bawah tabel. Form ini nantinya dipakai untuk proses penambahan data mahasiswa. Silahkan hapus tag `<tr>` dari `App.js` mulai dari baris 67–94, dan ganti dengan kode program berikut:

```
<RowTambahMahasiswa />
```

Kemudian tambahkan juga perintah import di bagian atas:

```
import RowTambahMahasiswa from './components/RowTambahMahasiswa';
```

Agar lebih jelas, berikut saya tampilkan kembali semua kode program dari file `App.js`:

src\App.js

```
1 import React, { useState } from 'react';
2 import RowMahasiswa from './components/RowMahasiswa';
3 import RowTambahMahasiswa from './components/RowTambahMahasiswa';
4
5 // Data awal tabel mahasiswa
6 const arrMahasiswas = [
7   {
8     nim: "18010245",
9     nama: "Eka Putra",
10    jurusan: "Teknik Informatika",
11    asalProvinsi: "DKI Jakarta"
12  },
13  {
14    nim: "19010214",
15    nama: "Lisa Permata",
16    jurusan: "Sistem Informasi",
17    asalProvinsi: "Sumatera Barat"
```

```

18     },
19     {
20       nim: "20010710",
21       nama: "Rudi Setiawan",
22       jurusan: "Ilmu Komputer",
23       asalProvinsi: "Jawa Tengah"
24     },
25     {
26       nim: "20010790",
27       nama: "Friska Ramadhani",
28       jurusan: "Ilmu Komputer",
29       asalProvinsi: "Kalimantan Barat"
30     }
31   ];
32
33 const App = () => {
34   const [mahasiswas, setMahasiswas] = useState(arrMahasiswas);
35
36   return (
37     <div className="container mt-5">
38
39       <div className="row mt-5">
40         <div className="col">
41           <h1 className="text-center">Tabel Mahasiswa</h1>
42
43           <table className="table mt-4">
44             <thead>
45               <tr>
46                 <th>NIM</th>
47                 <th>Nama</th>
48                 <th>Jurusan</th>
49                 <th>Asal Provinsi</th>
50                 <th></th>
51             </tr>
52           </thead>
53           <tbody>
54             {
55               mahasiswas.map((mahasiswa) =>
56                 <RowMahasiswa
57                   key={mahasiswa.nim}
58                   mahasiswa={mahasiswa}
59                 />
60               )
61             }
62             <RowTambahMahasiswa />
63           </tbody>
64         </table>
65       </div>
66
67     </div>
68   )
69 }
70
71 }
72

```

```
73 export default App;
```

Kodenya menjadi lebih singkat karena sudah dipecah ke dalam komponen <RowMahasiswa/> di baris 56, dan komponen <RowTambahMahasiswa /> di baris 62.

Dan kode program dari file RowTambahMahasiswa.js:

```
src\components\RowTambahMahasiswa.js
```

```
1 const RowTambahMahasiswa = () => {
2   return (
3     <tr>
4       <td colSpan="5">
5         <form>
6           <div className="row row-cols-5 g-3">
7             <div className="col">
8               <input type="text" className="form-control"
9                 name="nim" placeholder="00000000" />
10            </div>
11            <div className="col">
12              <input type="text" className="form-control"
13                name="nama" placeholder="Fulan Fulana" />
14            </div>
15            <div className="col">
16              <input type="text" className="form-control"
17                name="jurusan" placeholder="Sistem Informasi" />
18            </div>
19            <div className="col">
20              <input type="text" className="form-control"
21                name="asalProvinsi" placeholder="DKI Jakarta" />
22            </div>
23            <div className="col">
24              <button type="submit" className="btn btn-primary">
25                Tambah</button>
26              </div>
27            </div>
28          </form>
29        </td>
30      </tr>
31    )
32  }
33
34 export default RowTambahMahasiswa;
```

Untuk saat ini, komponen RowTambahMahasiswa hanya berisi struktur form HTML dasar tanpa pemrosesan apapun. Save file di atas dan kembali periksa tampilkan CRUD mahasiswa di web browser.

Jika ragu tentang urutan kode program yang ada, semua file tersedia di file belajar_react.zip yang ada di Google Drive. Jika tampilannya tidak sesuai atau ada yang error, bisa samakan dengan file yang ada.

16.3. Menambah Data Mahasiswa

Proses penambahan data mahasiswa dilakukan dari form yang ada di bagian bawah tabel. Ketika form di submit, data akan diinput ke dalam state `mahasiswa`.

Namun sebelum ke sana, kita perlu membuat kode program validasi terlebih dahulu, yakni memastikan data form sudah sesuai dengan kriteria atau belum. Sebagai contoh, saya ingin membatasi inputan `nim` harus berisi 8 karakter angka, tidak boleh lebih dan tidak boleh diisi huruf. Kemudian inputan `nama`, `jurusan` dan `asalProvinsi` juga tidak boleh kosong (wajib diisi).

Semua proses validasi perlu ditulis ke dalam komponen `RowTambahMahasiswa`, karena sebelumnya struktur form sudah kita pindah dari komponen `App` ke komponen ini.

Saya akan memakai teknik pembuatan validasi yang sama seperti bahasan di bab form processing, yakni membuat 2 buah state berbentuk *nested object*. State pertama berisi data form, dan state kedua untuk menampung pesan error.

Berikut modifikasi komponen `RowTambahMahasiswa` dengan tambahan fitur validasi:

`src\components\RowTambahMahasiswa.js`

```

1 import React, { useState } from 'react';
2
3 const RowTambahMahasiswa = () => {
4
5   // state untuk data inputan form
6   const [formInput, setFormInput] = useState({
7     nim: '',
8     nama: '',
9     jurusan: '',
10    asalProvinsi: '',
11  });
12
13   // state untuk menampung pesan error
14   const [errors, setErrors] = useState({
15     nim: '',
16     nama: '',
17     jurusan: '',
18     asalProvinsi: ''
19   });
20
21   // function untuk membuat 2 ways binding antara form dengan state
22   const handleInputChange = (event) => {
23     setFormInput({ ...formInput, [event.target.name]: event.target.value })
24   }
25
26   const handleFormSubmit = (e) => {
27     e.preventDefault();
28     let pesanErrors = {};
29
30     // validasi nim
31     if (formInput.nim.trim() === "") {

```

```
32     pesanErrors.nim = "Nim tidak boleh kosong";
33 }
34 else if (!/^[0-9]{8}$/.test(formInput.nim)) {
35     pesanErrors.nim = "Nim harus 8 karakter angka";
36 }
37 else {
38     pesanErrors.nim = "";
39 }
40
41 // validasi nama
42 if (formInput.nama.trim() === "") {
43     pesanErrors.nama = "Nama tidak boleh kosong";
44 }
45 else {
46     pesanErrors.nama = "";
47 }
48
49 // validasi jurusan
50 if (formInput.jurusan.trim() === "") {
51     pesanErrors.jurusan = "Jurusan tidak boleh kosong";
52 }
53 else {
54     pesanErrors.jurusan = "";
55 }
56
57 // validasi asalProvinsi
58 if (formInput.asalProvinsi.trim() === "") {
59     pesanErrors.asalProvinsi = "Asal Provinsi tidak boleh kosong";
60 }
61 else {
62     pesanErrors.asalProvinsi = "";
63 }
64
65 // update error state
66 setErrors(pesanErrors);
67
68 // cek apakah seluruh form valid atau masih ada error
69 let formValid = true;
70 for (let inputName in pesanErrors) {
71     if (pesanErrors[inputName].length > 0) {
72         formValid = false;
73     }
74 }
75
76 // proses data jika form valid
77 if (formValid) {
78     console.log(formInput);
79
80     // kosongkan inputan form
81     setFormInput({
82         nim: "",
83         nama: "",
84         jurusan: "",
85         asalProvinsi: "",
86     })
}
```

```
87     }
88 }
89
90 return (
91   <tr>
92     <td colSpan="5">
93       <form onSubmit={handleFormSubmit}>
94         <div className="row row-cols-5 g-3">
95           <div className="col">
96             <input type="text" className="form-control" name="nim"
97               placeholder="00000000" onChange={handleInputChange}
98               value={formInput.nim} />
99             {errors.nim && <small>{errors.nim}</small>}
100            </div>
101           <div className="col">
102             <input type="text" className="form-control" name="nama"
103               placeholder="Fulan Fulana" onChange={handleInputChange}
104               value={formInput.nama} />
105             {errors.nama && <small>{errors.nama}</small>}
106            </div>
107           <div className="col">
108             <input type="text" className="form-control" name="jurusan"
109               placeholder="Sistem Informasi" onChange={handleInputChange}
110               value={formInput.jurusan} />
111             {errors.jurusan && <small>{errors.jurusan}</small>}
112            </div>
113           <div className="col">
114             <input type="text" className="form-control" name="asalProvinsi"
115               placeholder="DKI Jakarta" onChange={handleInputChange}
116               value={formInput.asalProvinsi} />
117             {errors.asalProvinsi && <small>{errors.asalProvinsi}</small>}
118            </div>
119           <div className="col">
120             <button type="submit" className="btn btn-primary">
121               Tambah</button>
122             </div>
123           </div>
124         </form>
125       </td>
126     </tr>
127   )
128 }
129
130 export default RowTambahMahasiswa;
```

The screenshot shows a table titled "Tabel Mahasiswa" with four columns: NIM, Nama, Jurusan, and Asal Provinsi. Each row contains a set of input fields for these fields. Below the table, there is a form with four input fields and a "Tambah" button. The inputs are labeled: "Nim tidak boleh kosong", "Nama tidak boleh kosong", "Jurusan tidak boleh kosong", and "Asal Provinsi tidak boleh kosong". The "Tambah" button has a blue border, indicating it is the target of a click.

NIM	Nama	Jurusan	Asal Provinsi
18010245	Eka Putra	Teknik Informatika	DKI Jakarta
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat
00000000	Fulan Fulana	Sistem Informasi	DKI Jakarta
Nim tidak boleh kosong	Nama tidak boleh kosong	Jurusan tidak boleh kosong	Asal Provinsi tidak boleh kosong
			Tambah

Gambar: Pesan error validasi tampil di sisi bawah form

Kodenya cukup panjang karena kita perlu men-validasi 4 inputan form, yakni `nim`, `nama`, `jurusan` dan `asalProvinsi`.

Lompat dahulu ke struktur JSX, tag `<form>` di baris 93 memiliki tambahan atribut `onSubmit={handleFormSubmit}`. Ini berarti ketika di submit, fungsi `handleFormSubmit()` akan memproses semua nilai form, termasuk proses validasi akan kita buat ke dalam fungsi ini.

Perhatikan juga semua inputan form sudah memiliki tambahan atribut `onChange` dan `value`. Sebagaimana materi kita di bab tentang *form processing*, tag `<input>` harus berpasangan dengan React state agar menjadi *controller components*. Setiap kali terjadi event `onChange`, nilai state untuk inputan tersebut juga akan di update.

Sebagai contoh, di baris 97-98 terdapat atribut `onChange={handleInputChange}` dan `value={formInput.nim}`. Di sini, fungsi `handleInputChange()` akan memproses nilai input dari form ke dalam state, sedangkan atribut `value` akan menerima nilai state `formInput.nim` untuk diisi sebagai inputan form. Seluruh tag `<input>` form memiliki kedua atribut ini.

Jika ternyata inputan form tidak lolos validasi, pesan error akan tampil di dalam tag `<small>` pada bagian bawah setiap inputan. Sebagai contoh, di baris 99 terdapat kode berikut:

```
{errors.nim && <small>{errors.nim}</small>}
```

Dari kode ini bisa ditebak kalau pesan error untuk inputan `nim` akan disimpan ke dalam state `errors.nim`. Begitu juga dengan pesan error untuk inputan lain yang akan disimpan ke dalam state `errors.nama`, `errors.jurusan` dan `errors.asalProvinsi`.

Masuk ke kode React, di baris 1 terdapat perintah `import react` beserta alias `useState`. Ini diperlukan karena kita akan memakai `useState hook` dalam program ini.

Di baris 6-11 terdapat pendefinisian state `formInput` untuk menampung semua inputan form. Agar mudah dikelola, saya membuatnya dalam bentuk *nested object*, sehingga jika kita ingin mengakses nilai inputan `nim`, itu tersimpan dalam `formInput.nim`, atau jika ingin mengakses nilai `jurusan`, bisa dari `formInput.jurusan`.

Di baris 14-29, giliran pendefinisian state `errors`. State ini dipakai untuk menampung semua pesan error yang juga dibuat dalam bentuk *nested object*.

Masuk ke baris 22, terdapat fungsi `handleInputChange()`. Fungsi ini bertugas untuk menghubungkan semua inputan form dengan state `formInput`. Syarat tambahan, di dalam setiap tag `<input>` harus ada atribut `name` dengan nama yang sama seperti nama object state `formInput`. Teknik ini sudah pernah di bahas dalam bab form processing.

Karena kita menggunakan *functional component* dan state dalam bentuk *nested object*, proses update nilai state harus dengan bantuan *spread operator* "..." seperti di baris 23. Ini diperlukan agar React bisa mendeteksi perubahan nilai state dan melakukan proses re-render. Masalah ini juga pernah kita bahas di bab `useState` hook.

Selanjutnya di baris 26 terdapat fungsi `handleSubmit()` yang akan aktif saat form di submit. Seperti biasa, perintah `e.preventDefault()` dipakai untuk menonaktifkan perilaku default form agar tidak dikirim ke server back-end. Setelah itu terdapat pendefinisian variabel `pesanErrors` untuk menampung pesan error di baris 28.

Proses validasi dilakukan antara baris 31-63. Saya menggunakan pola pemeriksaan yang sudah sering kita pakai, dimana setiap kondisi if-else akan memeriksa apakah inputan form sesuai dengan syarat atau tidak. Jika syarat tidak terpenuhi, simpan pesan error ke dalam variabel `pesanErrors`.

Misalnya karena semua inputan tidak boleh kosong, maka syarat `trim() === ""` harus bisa dipenuhi. Untuk inputan `nim`, terdapat syarat tambahan yang saya tulis dalam bentuk *regular expression*, yakni:

```
else if (!/^[\d-]{8}$.test(formInput.nim))
```

Pola `^\d{8}$` hanya akan terpenuhi jika `formInput.nim` berisi persis 8 digit angka, tidak boleh kurang atau lebih, termasuk juga tidak boleh diisi karakter selain angka.

Pesan error yang di simpan dalam variabel `pesanErrors` selanjutnya di update ke dalam state `errors` dengan perintah `setErrors(pesanErrors)` seperti di baris 66.

Untuk memastikan seluruh inputan sudah lolos validasi, dilakukan dengan cara memeriksa isi variabel `pesanErrors` menggunakan perulangan `for in` antara baris 69-74. Teknik ini juga sudah beberapa kali kita pakai.

Terakhir, jika tidak ada pesan error lagi, blok if antara baris 77 – 87 akan di proses. Untuk sekarang, cukup tampilkan isi state `formInput` ke dalam tab console dengan perintah

`console.log(formInput)`. Nantinya blok if ini akan kita ganti dengan proses input ke dalam state `mahasiswa`s. Setelah itu kosongkan nilai inputan form dengan cara mengisi string kosong ke setiap nilai state.

Silahkan tes kode kita dengan berbagai nilai input. Pastikan semua syarat validasi sudah sesuai dan data mahasiswa bisa tampil hanya jika lolos validasi:

The screenshot shows a browser window titled "CRUD Mahasiswa" displaying a list of students. Below the table, there are input fields for adding a new student: nim (00000000), nama (Fulan Fulana), jurusan (Sistem Informasi), and asalProvinsi (DKI Jakarta). A "Tambah" button is also present. The browser's developer tools are open, specifically the Console tab, which shows a warning message: "src\App.js: Line 34:22: 'setMahasiswa' is assigned a value but never used no-unused-vars". An arrow points from this message to the "nim" field in the screenshot. Another red arrow points to the "Tambah" button.

Gambar: Nilai inputan form sukses tampil ke tab console

Karena validasi sudah selesai, sekarang kita akan tambah kode program untuk menginput data mahasiswa ke dalam state `mahasiswa`s.

Masih ingatkah caranya? Betul, kita harus menggunakan teknik `state-up` dengan mengirim event handler sebagai props dari komponen App ke komponen `RowTambahMahasiswa`. Ini diperlukan karena state `mahasiswa`s ada di dalam komponen App, sedangkan pemrosesan form ada di komponen `RowTambahMahasiswa`.

Buka file `src\App.js`, lalu tambah satu function berikut:

`src\App.js`

```
1  ...
2  const App = () => {
3      const [mahasiswa, setMahasiswa] = useState(arrMahasiswa);
4
5      // handler untuk menambah data mahasiswa,
6      // akan di-trigger dari komponen RowTambahMahasiswa
7      const handleTambahMahasiswa = (data) => {
8          const newMahasiswa = [
9              ...mahasiswa, data
10         ];
11         setMahasiswa(newMahasiswa);
12     }
13     ...
14 }
```

Function `handleTambahMahasiswa()` dirancang agar bisa menerima 1 parameter, yakni `data`. Nantinya `data` akan berisi 1 object mahasiswa yang sudah lengkap dengan semua property, yakni `nim`, `nama`, `jurusan` dan `asalProvinsi`.

Di baris 8, konstanta `newMahasiswas` berisi hasil penambahan state `mahasiswas` dengan parameter `data` tadi. Kemudian di baris 11, perintah `setMahasiswas(newMahasiswas)` akan mengupdate penambahan data ke dalam state `mahasiswas`.

Selanjutnya, kita perlu mengirim function `handleTambahMahasiswa()` sebagai props ke komponen `RowTambahMahasiswa`. Maka edit juga kode JSX dari `src\App.js`:

`src\App.js`

```
1 ...  
2     <tbody>  
3     {  
4         mahasiswas.map((mahasiswa) =>  
5             <RowMahasiswa  
6                 key={mahasiswa.nim}  
7                 mahasiswa={mahasiswa}  
8             />  
9         )  
10    }  
11    <RowTambahMahasiswa onTambahMahasiswa={handleTambahMahasiswa} />  
12  </tbody>  
13  ...
```

Di baris 11, tag `<RowTambahMahasiswa>` ditambah dengan atribut `onTambahMahasiswa={handleTambahMahasiswa}`, inilah cara mengirim fungsi `handleTambahMahasiswa` sebagai props.

Sesampainya di komponen `RowTambahMahasiswa`, kita perlu memanggil fungsi props `onTambahMahasiswa()` dari dalam fungsi `handleFormSubmit()`, yakni tepat saat seluruh proses validasi terpenuhi. Sebelumnya baris ini kita isi dengan perintah `console.log(formInput)`, sekarang ganti dengan `props.onTambahMahasiswa(formInput)`:

`src\components\RowTambahMahasiswa.js`

```
1 import React, { useState } from 'react';  
2  
3 const RowTambahMahasiswa = (props) => {  
4     ...  
5  
6     const handleFormSubmit = (e) => {  
7         ...  
8         // proses data jika form valid  
9         if (formValid) {  
10             props.onTambahMahasiswa(formInput);  
11             ...
```

Selain memodifikasi baris 10, jangan lupa untuk menambah props sebagai parameter dari komponen `RowTambahMahasiswa` di baris 3.

Save semua perubahan dan silahkan tes tambah mahasiswa baru menggunakan form di bagian bawah tabel:

NIM	Nama	Jurusan	Asal Provinsi		
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	Edit	Hapus
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	Edit	Hapus
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	Edit	Hapus
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat	Edit	Hapus
20010792	Anissa Lestari	Ilmu Komputer	Jawa Tengah	Edit	Hapus

Gambar: Penambahan mahasiswa baru sudah berhasil

Sip, data mahasiswa sudah berhasil tampil ke dalam tabel.

Sebelum kita masuk ke proses Edit data, alur logika penambahan mahasiswa saat ini sebenarnya masih ada yang kurang. Syarat validasi tambahan yang ingin saya buat adalah: nim harus unik, tidak boleh ada nim berulang yang bisa diinput.

Saat ini validasi tersebut belum ada, maka angka nim yang sama bisa saja diinput ke dalam state mahasiswas dan React akan "komplain" karena nim ini juga saya pakai sebagai pengisi atribut key.

Jadi, bagaimana menambahkan validasi "nim harus unik"?

Karena data mahasiswa disimpan ke dalam state (bukan ke dalam database yang sebenarnya), maka kita harus bandingkan nim saat proses input dengan seluruh nim yang ada di state mahasiswa.

Proses pemeriksaan ini lebih pas dilakukan dari dalam komponen RowTambahMahasiswa, yakni saat proses validasi. Akan tetapi data semua nim mahasiswa hanya ada di dalam state mahasiswas di komponen App. Maka solusinya, saya akan kirim nilai state mahasiswas sebagai props.

Silahkan modifikasi pemanggilan komponen <RowTambahMahasiswa> dari file src\App.js menjadi sebagai berikut:

src\App.js

```
1 ...  
2   <tbody>  
3   ...  
4     <RowTambahMahasiswa  
5       mahasiswas={mahasiswas}  
6       onTambahMahasiswa={handleTambahMahasiswa}  
7     />  
8   </tbody>  
9   ...
```

Sekarang terdapat tambahan atribut `mahasiswas={mahasiswas}` di baris 5. Maka sesampainya di dalam komponen `RowTambahMahasiswa`, data `mahasiswas` sudah bisa diakses dari `props.mahasiswas`.

Proses pemeriksaan `nim` seharusnya ada di dalam fungsi `handleFormSubmit()`, karena akan menjadi bagian dari proses validasi data. Akan tetapi supaya kodennya lebih rapi, saya akan buat fungsi bantu bernama `cekDuplikasiNim()`. Berikut kode yang diperlukan:

src\components\RowTambahMahasiswa.js

```
1 import React, { useState } from 'react';  
2  
3 const RowTambahMahasiswa = (props) => {  
4   ...  
5   // function untuk memeriksa apakah ada nim yang sama atau tidak  
6   const cekDuplikasiNim = () => {  
7     return (props.mahasiswas.find((mahasiswa) =>  
8       mahasiswa.nim === formInput.nim));  
9   }  
10  ...
```

Fungsi `cekDuplikasiNim()` akan mengembalikan hasil pemanggilan method `find()` bawaan JavaScript (baris 7-8).

Di dalam JavaScript native, method `find()` bisa dipakai untuk proses pencarian data di sebuah array. Method `find()` dipanggil dari variabel berisi array yang akan diperiksa. Dalam contoh ini, array tersebut adalah `props.mahasiswas`.

Kemudian, method `find()` butuh sebuah fungsi `callback` sebagai argument. Fungsi inilah yang akan menentukan syarat yang ingin dicari dan akan terus diulang untuk setiap element array. Di sini saya ingin membandingkan apakah isi dari `mahasiswa.nim` (`nim` yang tersimpan di dalam state `mahasiswas`), sama dengan `formInput.nim` (`nim` yang saat ini ada di dalam inputan form).

Method `find()` akan mengembalikan angka `nim` yang ditemukan (seperti "18010245"), atau mengembalikan `undefined` jika tidak ditemukan.

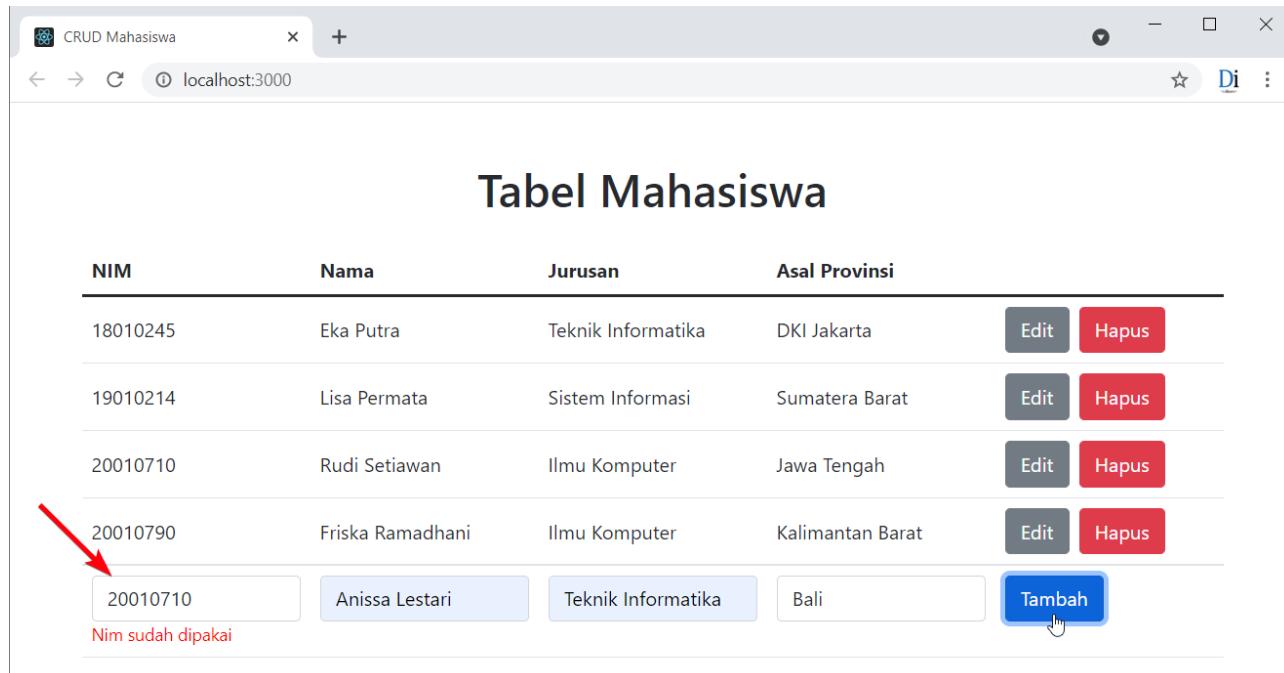
Jika butuh penjelasan lebih lengkap tentang cara penggunaan method `find()`, tersedia di buku **JavaScript Uncover**.

Dalam JavaScript, angka atau string acak bisa dikonversi menjadi boolean `true`, sedangkan `undefined` akan menjadi boolean `false`. Sehingga fungsi `cekDuplikasiNim()` bisa langsung kita pakai sebagai salah satu syarat validasi untuk inputan `nim`:

src\components\RowTambahMahasiswa.js

```
1 ...  
2     const handleFormSubmit = (e) => {  
3         ...  
4         // validasi nim  
5         if (formInput.nim.trim() === "") {  
6             pesanErrors.nim = "Nim tidak boleh kosong";  
7         }  
8         else if (!/^[0-9]{8}$/.test(formInput.nim)) {  
9             pesanErrors.nim = "Nim harus 8 karakter angka";  
10        }  
11        else if (cekDuplikasiNim()) {  
12            pesanErrors.nim = "Nim sudah dipakai";  
13        }  
14        else {  
15            pesanErrors.nim = "";  
16        }  
17    }
```

Di baris 11, jika terdapat nim yang sama, maka isi pesan error dengan string "Nim sudah dipakai". Dengan penambahan ini, silahkan tes input nim mahasiswa dengan angka yang sudah ada, seharusnya sekarang akan tampil pesan error:



NIM	Nama	Jurusan	Asal Provinsi	
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	<button>Edit</button> <button>Hapus</button>
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	<button>Edit</button> <button>Hapus</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Edit</button> <button>Hapus</button>
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat	<button>Edit</button> <button>Hapus</button>
20010710	Anissa Lestari	Teknik Informatika	Bali	<button>Tambah</button>

Nim sudah dipakai

Gambar: Error karena terdapat duplikasi nim

Dengan tambahan validasi duplikasi nim, kode untuk komponen RowTambahMahasiswa sudah final dan tidak akan kita ubah lagi, sebab fungsinya hanya untuk penambahan data saja. Berikut saya tampilkan kembali kode program lengkap untuk komponen RowTambahMahasiswa:

src\components\RowTambahMahasiswa.js

```

1 import React, { useState } from 'react';
2
3 const RowTambahMahasiswa = (props) => {
4
5     // state untuk data inputan form
6     const [formInput, setFormInput] = useState({
7         nim: "",
8         nama: "",
9         jurusan: "",
10        asalProvinsi: ""
11    });
12
13     // state untuk menampung pesan error
14     const [errors, setErrors] = useState({
15         nim: "",
16         nama: "",
17         jurusan: "",
18         asalProvinsi: ""
19    });
20
21     // function untuk membuat 2 ways binding antara form dengan state
22     const handleInputChange = (event) => {
23         setFormInput({ ...formInput, [event.target.name]: event.target.value })
24     }
25
26     // function untuk memeriksa apakah ada nim yang sama atau tidak
27     const cekDuplikasiNim = () => {
28         return (props.mahasiswas.find((mahasiswa) =>
29             mahasiswa.nim === formInput.nim));
30     }
31
32     const handleFormSubmit = (e) => {
33         e.preventDefault();
34         let pesanErrors = {};
35
36         // validasi nim
37         if (formInput.nim.trim() === "") {
38             pesanErrors.nim = "Nim tidak boleh kosong";
39         }
40         else if (!/^[0-9]{8}$/.test(formInput.nim)) {
41             pesanErrors.nim = "Nim harus 8 karakter angka";
42         }
43         else if (cekDuplikasiNim()) {
44             pesanErrors.nim = "Nim sudah dipakai";
45         }
46         else {
47             pesanErrors.nim = "";
48         }

```

```

49
50     // validasi nama
51     if (formInput.nama.trim() === "") {
52         pesanErrors.nama = "Nama tidak boleh kosong";
53     }
54     else {
55         pesanErrors.nama = "";
56     }
57
58     // validasi jurusan
59     if (formInput.jurusan.trim() === "") {
60         pesanErrors.jurusan = "Jurusan tidak boleh kosong";
61     }
62     else {
63         pesanErrors.jurusan = "";
64     }
65
66     // validasi asalProvinsi
67     if (formInput.asalProvinsi.trim() === "") {
68         pesanErrors.asalProvinsi = "Asal Provinsi tidak boleh kosong";
69     }
70     else {
71         pesanErrors.asalProvinsi = "";
72     }
73
74     // update error state
75     setErrors(pesanErrors);
76
77     // cek apakah seluruh form valid atau masih ada error
78     let formValid = true;
79     for (let inputName in pesanErrors) {
80         if (pesanErrors[inputName].length > 0) {
81             formValid = false;
82         }
83     }
84
85     // proses data jika form valid
86     if (formValid) {
87         props.onTambahMahasiswa(formInput);
88
89         // kosongkan inputan form
90         setFormInput({
91             nim: "",
92             nama: "",
93             jurusan: "",
94             asalProvinsi: "",
95         })
96     }
97 }
98
99 return (
100 <tr>
101     <td colSpan="5">
102         <form onSubmit={handleFormSubmit}>
103             <div className="row row-cols-5 g-3">

```

```

104     <div className="col">
105         <input type="text" className="form-control" name="nim"
106             placeholder="00000000" onChange={handleInputChange}
107             value={formInput.nim} />
108             {errors.nim && <small>{errors.nim}</small>}
109     </div>
110     <div className="col">
111         <input type="text" className="form-control" name="nama"
112             placeholder="Fulan Fulana" onChange={handleInputChange}
113             value={formInput.nama} />
114             {errors.nama && <small>{errors.nama}</small>}
115     </div>
116     <div className="col">
117         <input type="text" className="form-control" name="jurusan"
118             placeholder="Sistem Informasi" onChange={handleInputChange}
119             value={formInput.jurusan} />
120             {errors.jurusan && <small>{errors.jurusan}</small>}
121     </div>
122     <div className="col">
123         <input type="text" className="form-control" name="asalProvinsi"
124             placeholder="DKI Jakarta" onChange={handleInputChange}
125             value={formInput.asalProvinsi} />
126             {errors.asalProvinsi && <small>{errors.asalProvinsi}</small>}
127     </div>
128     <div className="col">
129         <button type="submit" className="btn btn-primary">
130             Tambah</button>
131     </div>
132     </div>
133   </form>
134 </td>
135 </tr>
136 )
137 }
138
139 export default RowTambahMahasiswa;

```

Selanjutnya kita akan masuk ke bahasan tentang mengubah atau mengedit data mahasiswa.

16.4. Edit Data Mahasiswa

Dalam aplikasi React, proses edit data umumnya dilakukan dengan jendela *popup* atau jendela *modal*. Proses ini kadang butuh library tambahan untuk menampilkan jendela tersebut, Bootstrap sendiri sebenarnya memiliki komponen modal yang bisa kita gunakan.

Akan tetapi kali ini saya ingin membuat konsep yang sedikit berbeda. Ketika tombol "Edit" di klik, baris tabel akan berubah menjadi inputan form dan data bisa langsung di edit langsung dari dalam tabel. Dengan cara ini kita tidak perlu jendela modal terpisah.

Berikut tampilan yang dimaksud:

Sebagian besar kode yang akan kita buat ada di komponen `RowMahasiswa`. Karena inilah

komponen yang bertanggung jawab untuk menampilkan setiap baris data mahasiswa.

Ide dasar dari tampilan di atas adalah, buat sebuah state `editStatus` sebagai *flag* atau penanda. Saat tombol "Edit" di klik, set `editStatus` menjadi `true` sebagai tanda kita sedang masuk ke dalam "Edit mode".

Kemudian di dalam kode JSX, cek isi dari `editStatus` ini. Jika bernilai `true`, tampilkan baris tabel dengan sebuah form untuk proses edit. Tidak lupa tombol "Edit" diganti menjadi tombol "Simpan".

Setelah proses edit selesai, user harus menekan tombol "Simpan" agar perubahan data sampai ke state `mahasiswas`. Tepat saat itu terjadi, update kembali state `editStatus` menjadi `false` sebagai tanda kita keluar dari "Edit mode". Di dalam kode JSX, jika `editStatus` berisi `false`, maka baris tabel akan tampil seperti biasa (bukan dalam bentuk form).

Kita akan bahas cara penerapan konsep ini secara bertahap. Sebagai pembanding, berikut saya tampilkan kembali isi komponen `RowMahasiswa` saat ini:

src\components\RowMahasiswa.js

```
1 const RowMahasiswa = (props) => {
2   return (
3     <tr>
4       <td>{props.mahasiswa.nim}</td>
5       <td>{props.mahasiswa.nama}</td>
6       <td>{props.mahasiswa.jurusan}</td>
7       <td>{props.mahasiswa.asalProvinsi}</td>
8       <td>
9         <button className="btn btn-secondary me-2">Edit</button>
10        <button className="btn btn-danger">Hapus</button>
11      </td>
12    </tr>
13  )
14}
15
16 export default RowMahasiswa;
```

Komponen `RowMahasiswa` dipakai untuk menampilkan 1 baris data mahasiswa. Perhatikan juga bahwa komponen ini sudah menerima `props` berupa data mahasiswa yang ingin ditampilkan.

Membuat "Edit Mode"

Untuk membuat fitur "Edit mode", kode di atas perlu dimodifikasi sebagai berikut:

src\components\RowMahasiswa.js

```
1 import React, { useState } from 'react';
2
3 const RowMahasiswa = (props) => {
4
5   // simpan props mahasiswa ke dalam state agar mudah diakses
```

```
6  const [formInput, setFormInput] = useState({
7    nim: props.mahasiswa.nim,
8    nama: props.mahasiswa.nama,
9    jurusan: props.mahasiswa.jurusan,
10   asalProvinsi: props.mahasiswa.asalProvinsi,
11 });
12
13 // state untuk penanda "Edit mode"
14 const [editStatus, setEditStatus] = useState(false);
15
16 // function untuk membuat 2 ways binding antara form dengan state
17 const handleInputChange = (event) => {
18   setFormInput({ ...formInput, [event.target.name]: event.target.value })
19 }
20
21 // tombol Edit di klik
22 const handleEditClick = () => {
23   setEditStatus(true);
24 }
25
26 // form di submit
27 const handleFormSubmit = (e) => {
28   e.preventDefault();
29   setEditStatus(false);
30 }
31
32 return (
33   <React.Fragment>
34     {/* Tampilkan form jika tombol Edit di klik, atau tampilkan row normal
35      */}
36     {editStatus ?
37       <tr>
38         <td colSpan="5">
39           <form onSubmit={handleFormSubmit}>
40             <div className="row row-cols-5 g-3">
41               <div className="col">
42                 <input type="text" className="form-control" disabled
43                   defaultValue={formInput.nim} name="nim" />
44               </div>
45               <div className="col">
46                 <input type="text" className="form-control" name="nama"
47                   onChange={handleInputChange} value={formInput.nama} />
48               </div>
49               <div className="col">
50                 <input type="text" className="form-control" name="jurusan"
51                   onChange={handleInputChange} value={formInput.jurusan} />
52               </div>
53               <div className="col">
54                 <input type="text" className="form-control"
55                   name="asalProvinsi" onChange={handleInputChange}
56                   value={formInput.asalProvinsi} />
57               </div>
58               <div className="col">
59                 <button className="btn btn-success me-2" type="submit">
60                   Simpan</button>
61               </div>
62             </div>
63           </form>
64         </td>
65       </tr>
66     }
67   )
68 )
69
70 <tbody>
71   <tr>
72     <td>1</td>
73     <td>Andi</td>
74     <td>Teknik Informatika</td>
75     <td>Surabaya</td>
76     <td><button onClick={handleEditClick}>Edit</button></td>
77   </tr>
78 </tbody>
79
80 <tfoot>
81   <tr>
82     <th>No</th>
83     <th>Nama</th>
84     <th>Jurusan</th>
85     <th>Asal Provinsi</th>
86     <th>Action</th>
87   </tr>
88 </tfoot>
89
90 </table>
91
92 </div>
93
94 </div>
95
96 </div>
97
98 </div>
99
100 </div>
```

```

60             <button className="btn btn-warning" type="reset">
61                 Batal</button>
62             </div>
63         </div>
64     </form>
65     </td>
66 </tr>
67 :
68 <tr>
69     <td>{formInput.nim}</td>
70     <td>{formInput.nama}</td>
71     <td>{formInput.jurusan}</td>
72     <td>{formInput.asalProvinsi}</td>
73     <td>
74         <button className="btn btn-secondary me-2"
75             onClick={handleEditClick}>Edit</button>
76         <button className="btn btn-danger">Hapus</button>
77     </td>
78 </tr>
79 }
80
81 </React.Fragment >
82 )
83 }
84
85 export default RowMahasiswa;

```

Agar lebih mudah dipahami, kita mulai dari struktur JSX terlebih dahulu (baris 33 – 81).

Sebagai container terluar, terdapat tag `<React.Fragment>` di baris 33. Tag ini diperlukan karena child element langsung berbentuk pemeriksaan kondisi. Kondisi yang dimaksud ada di baris 35, yakni `"editStatus ?"`. Ini merupakan awal dari short-circuit conditionals.

Jika state `editStatus` berisi nilai `true`, maka kode program antara baris 36-66 akan ditampilkan. Kode ini berisi tag `<tr>` yang didalamnya terdapat form untuk proses update data mahasiswa. Form ini terdiri dari inputan `nim`, `nama`, `jurusan` dan `asalProvinsi`.

Tag `<form>` di baris 38 berisi atribut `onSubmit={handleFormSubmit}`. Artinya, saat form di submit, pemrosesan data akan dilakukan oleh fungsi `handleFormSubmit()`.

Sebagaimana layaknya form yang diproses dengan React, setiap tag `<input>` memiliki atribut `onChange={handleInputChange}` serta atribut `value` yang nilainya merujuk ke state dari inputan tersebut. Struktur form ini kurang lebih mirip seperti yang baru saja kita buat di komponen `RowTambahMahasiswa`.

Pengecualian ada untuk inputan `nim`. Untuk proses edit ini, saya memutuskan `nim` tidak bisa diubah, sebab `nim` menjadi nomor unik dari setiap mahasiswa. Untuk itu, tag `<input>` di baris 41-42 di set sebagai `disabled` agar nilainya tidak bisa diubah.

Nilai awal untuk inputan `nim` menggunakan atribut `defaultValue={formInput.nim}`, dan bukan `value={formInput.nim}`. Ini merupakan syarat tambahan dari React bahwa jika tag `<input>`

tidak memiliki event `onChange`, maka nilai awal harus diisi dari `defaultValue`. Jika tetap menggunakan atribut `value`, akan tampil pesan warning di tab console.

Di akhir form, terdapat 2 buah tombol dari tag `<button>`, yakni tombol "Simpan" (baris 58-59) dan tombol "Batal" (baris 60-61). Tombol "Simpan" memiliki atribut `type="submit"` sehingga berfungsi sebagai tombol submit form, sedangkan tombol "Batal" memiliki atribut `type="reset"` yang berfungsi untuk membatalkan proses edit.

Semua inputan form ada di dalam tag `<tr>` dan `<td colspan="5">`, sehingga masih ada di dalam tabel mahasiswa.

Di baris 67, terdapat tanda titik dua ":" yang merupakan sambungan dari `short-circuit conditionals` `"editStatus ?"` di baris 35. Blok kode program antara baris 68-78 hanya akan tampil jika state `editStatus` berisi nilai `false`, yang artinya kita sudah keluar dari "Edit mode".

Apabila syarat ini terpenuhi, data mahasiswa tampil dalam bentuk baris tabel biasa, plus tambahan tombol "Edit" dan tombol "Hapus" di sisi kanan. Tombol "Edit" memiliki atribut `onClick={handleEditClick}`, yang berarti saat tombol ini di klik, fungsi `handleEditClick()` akan berjalan.

Selesai dengan kode JSX, kita masuk ke kode React. Di baris 6, terdapat pendefinisian state `formInput` untuk menampung semua data form. Sama seperti di `RowTambahMahasiswa`, state ini berbentuk *nested object*. Nilai awal state langsung diambil dari `props.mahasiswa` yang didapat dari komponen `App`.

Lanjut, di baris 14 terdapat pendefinisian state `editStatus` dengan nilai awal `false`. Inilah state yang menjadi penentu "Edit mode". Jika nilainya `false`, maka tabel mahasiswa tampil normal. Tapi jika nilainya `true`, akan tampil form untuk proses edit data. Perubahan nilai `editStatus` ini nantinya akan di proses oleh berbagai *event handler*.

Di baris 17, fungsi `handleInputChange()` ditujukan untuk proses update state dari inputan form. Fungsi ini sudah sering kita gunakan sebelumnya.

Kemudian di baris 22 terdapat fungsi `handleEditClick()` yang berisi perintah `setEditStatus(true)`. Ini adalah *event handler* yang akan berjalan saat tombol "Edit" di klik. Jika itu terjadi, ubah state `editStatus` menjadi `true`. Karena nilai state berubah, komponen `RowMahasiswa` akan di render ulang dan di dalam JSX akan masuk ke blok yang menampilkan form edit.

Terakhir di baris 27 terdapat fungsi `handleFormSubmit()` yang akan berjalan saat form di submit. Untuk saat ini hanya berisi perintah `e.preventDefault()` dan `setEditStatus(false)`. Artinya saat form di submit, tukar kembali isi state `editStatus` menjadi `false`. Perubahan ini akan merender ulang komponen `RowMahasiswa` dan di dalam JSX akan masuk ke blok yang menampilkan data tabel saja (bukan dalam bentuk form).

Silahkan tes tombol "Edit" dan "Simpan" di setiap baris. Tampilannya akan berubah dari bentuk

form ke bentuk data biasa:

NIM	Nama	Jurusan	Asal Provinsi	
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	<button>Edit</button> <button>Hapus</button>
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	<button>Simpan</button> <button>Batal</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Edit</button> <button>Hapus</button>
20010790	Friska Ramadhan	Ilmu Komputer	Kalimantan Barat	<button>Simpan</button> <button>Batal</button>
00000000	Fulan Fulana	Sistem Informasi	DKI Jakarta	<button>Tambah</button>

Gambar: Baris tabel berubah menjadi form saat tombol Edit di klik

Ketika tombol Edit di klik, baris tabel akan berubah menjadi form. Setelah itu jika tombol Simpan di klik, tampilannya kembali ke bentuk semula. Untuk tombol Batal dan Hapus saat ini belum berfungsi dan akan kita buat sesaat lagi.

Perhatikan juga bahwa setiap baris tabel bisa diedit secara terpisah, tidak harus menunggu satu baris selesai terlebih dahulu. Inilah keunggulan dari konsep component di React. Setiap baris memiliki "dunianya sendiri", dan tidak saling bentrok satu sama lain. Jika tampilan di atas dibuat dengan JavaScript biasa, setiap baris tabel harus dibedakan agar datanya tidak saling bentrok.

Form tambah di baris paling bawah juga tetap bisa berfungsi. Jika kita menambah data baru, data tersebut sudah punya tombol Edit yang juga bisa di klik.

Membuat Validasi Form Edit

Selanjutnya kita akan tambah fitur validasi dan pemrosesan form untuk menyimpan data ke dalam state `mahasiswa`.

Untuk menampilkan pesan error, perlu tambahan state `errors` dibagian atas komponen:

`src\components\RowMahasiswa.js`

```

1 import React, { useState } from 'react';
2
3 const RowMahasiswa = (props) => {
4   ...
5   // state untuk menampung pesan error

```

```
6  const [errors, setErrors] = useState({
7    nama: "",
8    jurusan: "",
9    asalProvinsi: ""
10   });
11 ...
```

Sama seperti teknik yang kita pakai selama ini, state errors ditulis dalam bentuk nested object. Pesan error untuk nim tidak diperlukan karena di dalam form nim memang tidak bisa di-edit.

Masuk ke fungsi handleFormSubmit(), perlu tambahan kode program untuk proses validasi:

src\components\RowMahasiswa.js

```
1 import React, { useState } from 'react';
2
3 const RowMahasiswa = (props) => {
4 ...
5 // form di submit
6 const handleFormSubmit = (e) => {
7   e.preventDefault();
8   let pesanErrors = {};
9
10  // validasi nama
11  if (formInput.nama.trim() === "") {
12    pesanErrors.nama = "Nama tidak boleh kosong";
13  }
14  else {
15    pesanErrors.nama = "";
16  }
17
18  // validasi jurusan
19  if (formInput.jurusan.trim() === "") {
20    pesanErrors.jurusan = "Jurusan tidak boleh kosong";
21  }
22  else {
23    pesanErrors.jurusan = "";
24  }
25
26  // validasi asalProvinsi
27  if (formInput.asalProvinsi.trim() === "") {
28    pesanErrors.asalProvinsi = "Asal Provinsi tidak boleh kosong";
29  }
30  else {
31    pesanErrors.asalProvinsi = "";
32  }
33
34  // update error state
35  setErrors(pesanErrors);
36
37  // cek apakah seluruh form valid atau masih ada error
38  let formValid = true;
39  for (let propName in pesanErrors) {
40    if (pesanErrors[propName].length > 0) {
```

```

41         formValid = false;
42     }
43 }
44
45 // proses data jika form valid
46 if (formValid) {
47     setEditStatus(false);
48 }
49 }
50 ...

```

Proses validasi ada di baris 11-32 yang kurang lebih sama seperti di komponen RowTambahMahasiswa. Termasuk juga proses update state errors di baris 35, serta menentukan apakah semua inputan lolos validasi atau tidak dengan perulangan for in di baris 38-43.

Jika data inputan lolos validasi, perintah `setEditStatus(false)` di baris 47 akan berjalan untuk keluar dari "Edit mode".

Kode kita belum selesai, karena perlu memodifikasi kode JSX untuk menampilkan pesan error validasi. Berikut perubahan yang diperlukan:

src\components\RowMahasiswa.js

```

1 <form onSubmit={handleFormSubmit}>
2   <div className="row row-cols-5 g-3">
3     <div className="col">
4       <input type="text" className="form-control" disabled
5           defaultValue={formInput.nim} name="nama" />
6     </div>
7     <div className="col">
8       <input type="text" className="form-control" name="nama"
9           onChange={handleInputChange} value={formInput.nama} />
10      {errors.nama && <small>{errors.nama}</small>}
11    </div>
12    <div className="col">
13      <input type="text" className="form-control" name="jurusan"
14          onChange={handleInputChange} value={formInput.jurusan} />
15      {errors.jurusan && <small>{errors.jurusan}</small>}
16    </div>
17    <div className="col">
18      <input type="text" className="form-control"
19          name="asalProvinsi" onChange={handleInputChange}
20          value={formInput.asalProvinsi} />
21      {errors.asalProvinsi && <small>{errors.asalProvinsi}</small>}
22    </div>
23    <div className="col">
24      <button className="btn btn-success me-2" type="submit">
25        Simpan</button>
26      <button className="btn btn-warning" type="reset">
27        Batal</button>
28    </div>
29  </div>
30 </form>

```

Tambahan kode program ada di baris 10, 15 dan 21. Ini akan menampilkan pesan error tepat di bawah setiap inputan form.

Save file RowMahasiswa.js dan tes kosongkan salah satu inputan form untuk melihat pesan error validasi:

NIM	Nama	Jurusan	Asal Provinsi	
18010245	Eka Putra	Teknik Informatika	DKI Jakarta	<button>Edit</button> <button>Hapus</button>
19010214			Sumatera Barat	<button>Simpan</button> <button>Batal</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Edit</button> <button>Hapus</button>
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat	<button>Edit</button> <button>Hapus</button>
	00000000	Fulan Fulana	Sistem Informasi	<button>DKI Jakarta</button> <button>Tambah</button>

Gambar: Pesan validasi inputan form edit berhasil tampil

Simpan Data Edit

Dengan kode yang sudah ada, sebenarnya proses edit sudah bisa dilakukan. Silahkan tes ubah nama mahasiswa, klik tombol Simpan, dan tampilan tabel mahasiswa akan berubah. Akan tetapi perubahan ini hanya sampai di state formInput milik komponen RowMahasiswa.

Harusnya perubahan data ini diupdate ke dalam state mahasiswas yang ada di komponen App.

Sebagaimana cara yang biasa dilakukan untuk perubahan state antar komponen, kita perlu membuat "state up", yakni mengirim fungsi update dari komponen MyApp ke dalam komponen RowMahasiswa sebagai props.

Silahkan buka file src\App.js, lalu tambah satu fungsi berikut:

src\App.js

```
1 ...  
2 const App = () => {  
3     const [mahasiswas, setMahasiswas] = useState(arrMahasiswas);  
4  
5     // handler untuk mengedit data mahasiswa  
6     // akan di-trigger dari komponen RowMahasiswa  
7     const handleEditMahasiswa = (data) => {  
8         // cari index dari mahasiswa yang akan diedit berdasarkan nomor nim  
9         const result = mahasiswas.findIndex(
```

```
10     (mahasiswa) => mahasiswa.nim === data.nim
11 );
12
13 // copy mahasiswas karena fungsi splice akan mengubah array asal (mutate)
14 const newMahasiswas = mahasiswas;
15 newMahasiswas.splice(result, 1, data);
16 setMahasiswas([...newMahasiswas]);
17
18 // !! jika hanya menggunakan setMahasiswas(newMahasiswas),
19 // react tidak akan me-re-render halaman karena
20 // newMahasiswas = mahasiswa masih merujuk ke object yang sama.
21 }
22 ...
```

Fungsi `handleEditMahasiswa()` menerima satu parameter, yang nantinya akan kita kirim dari dalam komponen `RowMahasiswa`. Parameter data akan berisi 1 object mahasiswa yang sudah di update.

Sebelum melakukan proses update, kita harus mencari posisi index dari array mahasiswa yang ingin di update. Jika state `mahasiswas` saat ini berisi 5 object, mahasiswa yang perlu di update?

Inilah fungsi dari kode program di baris 9-11. Saya menggunakan method `findIndex()` bawaan JavaScript untuk mencari posisi index dari mahasiswa yang nim-nya sama dengan yang ada di dalam parameter `data.nim`. Index ini kemudian disimpan ke dalam konstanta `result`.

Sebagai contoh, misalkan state `mahasiswas` berisi 4 element array berikut:

```
const mahasiswas = [
  {
    nim: "18010245",
    ...
  },
  {
    nim: "19010214",
    ...
  },
  {
    nim: "20010710",
    ...
  },
  {
    nim: "20010790",
    ...
  }
];
```

Dan data yang ingin di update berisi informasi berikut:

```
const data = { nim: "20010710", ... };
```

Maka konstanta `result` akan berisi angka 2, sebab itulah posisi index dari element yang memiliki nim "20010710".

Setelah posisi index di dapat, proses update dilakukan antara baris 14-16. Pertama, isi state `mahasiswas` saya copy ke dalam konstanta `newMahasiswas`. Kemudian method `splice()` bawaan JavaScript akan mengganti 1 element array `newMahasiswas` yang indexnya ada di dalam konstanta `result` dengan data baru yang ada di dalam parameter `data`. Setelah perintah ini, konstanta `newMahasiswas` akan berisi data mahasiswa yang sudah di update.

Terakhir, state `mahasiswas` di update dengan perintah `setMahasiswas([...newMahasiswas])`. Di sini saya butuh bantuan *spread operator* agar React mendeteksi terjadi perubahan nilai state dan melakukan proses re-render. Jika langsung ditulis dengan perintah `setMahasiswas(newMahasiswas)`, proses re-render tidak akan ter-trigger.

Pemahaman seputar materi JavaScript dasar sangat membantu dalam pembuatan kode React. Contohnya seperti dalam mini project ini, kita butuh bantuan berbagai method dari Array object bawaan JavaScript, seperti `find()`, `findIndex()` dan `splice()`.

Fungsi `handleEditMahasiswa()` ini perlu kita kirim sebagai props, maka silahkan modifikasi proses pemanggilannya di dalam struktur JSX komponen App:

```
src\App.js
1 ...
2   mahasiswas.map((mahasiswa) =>
3     <RowMahasiswa
4       key={mahasiswa.nim}
5       mahasiswa={mahasiswa}
6       onEditMahasiswa={handleEditMahasiswa}
7     />
8   )
9 ...
```

Dengan tambahan atribut `onEditMahasiswa={handleEditMahasiswa}`, maka kita bisa memanggilnya dari komponen `RowMahasiswa`, tepatnya di fungsi `handleSubmit()` pada bagian yang diproses setelah seluruh validasi selesai:

```
src\components\RowMahasiswa.js
```

```
1 ...
2 // proses data jika form valid
3 if (formValid) {
4   props.onEditMahasiswa(formInput);
5   setStatus(false);
6 }
7 ...
```

Di baris 4, argument `formInput` akan menjadi parameter `data` sesampainya ke fungsi `handleEditMahasiswa()` di komponen App. Dengan tambahan kode ini, data mahasiswa yang diedit sudah sampai ke dalam state `mahasiswas`.

Membuat Fitur Reset Data

Pada saat masuk ke *edit mode*, di sisi baris paling kanan terdapat tombol "Batal". Tombol ini saya siapkan agar user bisa membatalkan proses edit dan nilai data mahasiswa kembali ke nilai sebelumnya.

Dalam merancang fitur ini, kita perlu bantuan sebuah state baru untuk menampung nilai inputan form tepat pada saat tombol Edit di klik. Ini diperlukan karena event `onChange` di setiap inputan akan terus menimpa nilai awal state.

Sebagai contoh kasus, misalkan nama mahasiswa saat ini adalah "Rudi". Ketika user men-klik tombol "Edit", akan muncul inputan form sebagai tanda kita sudah masuk ke *edit mode*. Di dalam tag `<input>` ini, nama mahasiswa diambil dari state `formInput.nama`.

Kemudian user mengetik beberapa karakter untuk mengubah nama ini menjadi "Rudianto". Setiap kali karakter di ketik, event `onChange` akan langsung mengupdate nilai state `formInput.nama`. Jika tiba-tiba user men-klik tombol "Batal", kita tidak memiliki informasi mengenai nilai sebelumnya, sebab isi state `formInput.nama` juga sudah berubah.

Inilah alasan perlu membuat state penampung untuk menyimpan nilai form awal. Jika ternyata user berubah pikiran dan men-klik tombol "Batal", tinggal copy nilai state penampung tadi ke dalam state `formInput`.

Pertama, kita siapkan state penampung yang dimaksud:

src\components\RowMahasiswa.js

```
1 import React, { useState } from 'react';
2
3 const RowMahasiswa = (props) => {
4   ...
5   // state untuk menampung nilai form sebelum "Edit mode"
6   const [dataReset, setDataReset] = useState({});
```

State untuk menampung nilai sementara saya beri nama `dataReset`. Nilai awal untuk `dataReset` cukup object kosong karena akan kita isi pada saat tombol "Edit" di klik, yakni di dalam fungsi `handleEditClick()`:

src\components\RowMahasiswa.js

```
1 ...
2   // tombol Edit di klik
3   const handleEditClick = () => {
4
5     // Simpan data untuk proses reset
6     setDataReset({ ...formInput });
7
8     // Masuk ke "edit mode"
9     setEditStatus(true);
```

```
10    }
11  ...
```

Tambahan perintah ada di baris 6, berupa `setDataReset({ ...formInput })`. Kode ini dipakai untuk mencopy nilai state `formInput` ke dalam state `dataReset`. Inilah data sementara yang akan kita perlukan jika user men-klik tombol "Batal". Proses copy dilakukan dengan bantuan spread operator "...".

Lanjut, kita perlu menambah event *handler* saat tombol "Batal" di klik. Ini bisa saja dilakukan dengan menambah atribut `onClick` ke dalam tag `<button>`, akan tetapi karena tombol "Batal" memiliki atribut `type="reset"` dan berada di dalam tag `<form>`, kita juga bisa memanfaatkan event `onReset` milik form.

Silahkan tambah atribut `onReset={handleFormReset}` ke dalam kode JSX tag `<form>` sebagai berikut:

src\components\RowMahasiswa.js

```
1  ...
2  <form onSubmit={handleFormSubmit} onReset={handleFormReset}>
3  ...
```

Dan tambah juga fungsi `handleFormReset()` ke dalam kode React:

src\components\RowMahasiswa.js

```
1  ...
2  // tombol Batal di klik
3  const handleFormReset = (e) => {
4      e.preventDefault();
5
6      // Kembalikan isi form ke posisi sebelum tombol edit di klik
7      setFormInput({ ...dataReset })
8
9      // Hapus pesan error (jika ada)
10     setErrors({});
11
12     // Keluar dari "Edit mode"
13     setEditStatus(false);
14 }
15 ...
```

Perintah `setFormInput({ ...dataReset })` di baris 7 dipakai untuk mengcopy isi state `dataReset` ke dalam state `formInput`. Inilah yang mengembalikan data form ke nilai sebelum proses edit berjalan.

Di baris 10, perintah `setErrors({})` berfungsi untuk menghapus pesan error jika ada. Ini sebagai antisipasi jika user membatalkan proses edit saat pesan error validasi masih tampil. Terakhir, kita keluar dari *edit mode* dengan perintah `setEditStatus(false)`.

Silahkan tes tombol "Batal" ini di aplikasi CRUD Mahasiswa. Edit satu baris, ubah beberapa nilai atau bisa juga dikosongkan, lalu klik tombol "Batal", seharusnya data tabel akan kembali ke nilai awal.

16.5. Hapus Data Mahasiswa

CRUD terakhir yang akan di rancang adalah menghapus data mahasiswa. Tombol "Hapus" ini sudah tersedia di dalam tabel dan tinggal kita tambah dengan sebuah *event handler*.

Sebagaimana biasa, proses penghapusan mahasiswa harus dilakukan dari komponen App, karena disitulah state `mahasiswas` berada. Function ini nantinya harus dikirim sebagai `props` ke dalam komponen `RowMahasiswa` untuk dijalankan oleh tombol "Hapus".

Berikut tambahan kode program di file `src\App.js`:

```
src\App.js

1  ...
2  const App = () => {
3    ...
4
5    // handler untuk menghapus data mahasiswa di komponen RowMahasiswa
6    const handleHapusMahasiswa = (e) => {
7
8      // cari index dari mahasiswa yang akan dihapus berdasarkan nomor nim
9      const result = mahasiswas.findIndex(
10        (mahasiswa) => mahasiswa.nim === e.target.id
11      );
12
13      // copy mahasiswas karena fungsi splice akan mengubah array asal (mutate)
14      const newMahasiswas = mahasiswas;
15      newMahasiswas.splice(result, 1);
16      setMahasiswas([...newMahasiswas]);
17
18      // Cara alternatif penghapusan dengan method filter
19      // const newMahasiswas = mahasiswas.filter(
20      //   mahasiswa => mahasiswa.nim !== e.target.id
21      // );
22      // setMahasiswas(newMahasiswas);
23    }
24  ...
```

Function `handleHapusMahasiswa()` menerima 1 parameter `e`. Parameter `e` ini akan berisi *event object* dari tombol "Hapus". Lebih lanjut tentang hal ini akan kita bahas saat pengaksesan `props` dari dalam komponen `RowMahasiswa` sesaat lagi. Di dalam parameter `e`, terdapat informasi mengenai nomor nim mahasiswa yang ingin dihapus dari property `e.target.id`.

Sama seperti proses update data, di baris 9-11 saya memakai method `findIndex()` bawaan JavaScript untuk mencari posisi index mahasiswa yang ada di `e.target.id`. Nilai ini kemudian

disimpan ke dalam konstanta `result`.

Proses penghapusan dilakukan pada baris 15 dengan bantuan method `splice()` yang juga bawaan JavaScript. Perintah `newMahasiswas.splice(result, 1)` akan menghapus 1 element dari array `newMahasiswas` pada posisi index yang disimpan dalam konstanta `result`. Hasil ini kemudian di update ke dalam state mahasiswa di baris 16.

Di bagian komentar bawah, saya menambah cara alternatif untuk menghapus sebuah element dari array dengan posisi tertentu, yakni menggunakan method `filter()`. Method `filter()` akan mengembalikan semua element array selama memenuhi function `callback` yang ditulis.

Langkah berikutnya, kirim function `handleHapusMahasiswa()` sebagai props di dalam tag `<RowMahasiswa />`:

src\App.js

```
1  ...
2    mahasiswa.map((mahasiswa) =>
3      <RowMahasiswa
4        key={mahasiswa.nim}
5        mahasiswa={mahasiswa}
6        onEditMahasiswa={handleEditMahasiswa}
7        onHapusMahasiswa={handleHapusMahasiswa}
8      />
9    )
10 ...
```

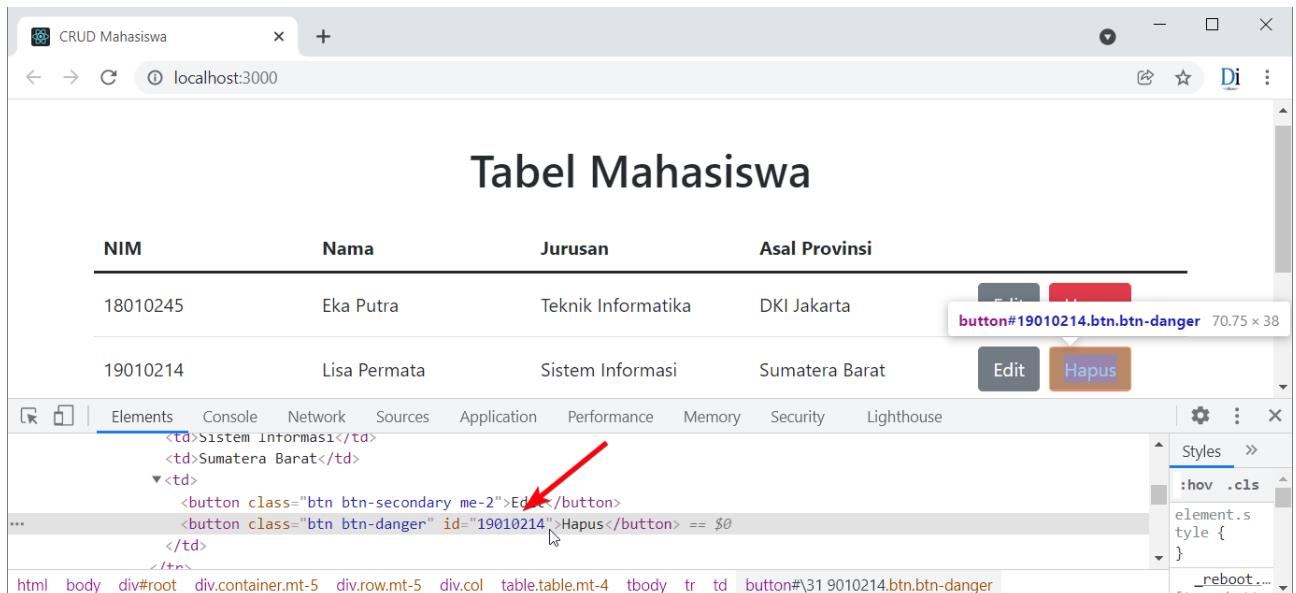
Di baris 7, saya menambah atribut `onHapusMahasiswa={handleHapusMahasiswa}`, sehingga nantinya bisa diakses dari komponen `RowMahasiswa` sebagai `props.onHapusMahasiswa`.

Untuk kode program di `RowMahasiswa`, kita perlu memikirkan bagaimana cara mendapatkan nilai nim dari mahasiswa yang akan dihapus. Salah satu solusi adalah dengan menempatkan atribut `id={formInput.nim}` ke dalam tombol "Hapus" dengan kode program berikut:

src\components\RowMahasiswa.js

```
1  ...
2    <button className="btn btn-danger" id={formInput.nim}
3      onClick={props.onHapusMahasiswa}>Hapus</button>
4  ...
```

Sebagai pengingat, di dalam komponen `RowMahasiswa`, state `formInput.nim` berisi nilai nim dari mahasiswa yang sedang di akses. Maka dengan kode program di atas, tag `<button>` dari tombol "Hapus" akan berisi nilai id dari setiap mahasiswa yang ada di baris tersebut. Berikut tampilan source code yang digenerate dari web browser:



Gambar: Atribut id berisi nim ada di dalam tag <button> tombol "Hapus"

Selanjutnya event `onClick` dari tombol "Hapus" langsung mengakses `props.onHapusMahasiswa` di baris 3. Ini tidak lain merujuk ke function `handleHapusMahasiswa()` milik komponen App yang sebelumnya sudah kita tulis.

Jika anda masih ingat materi tentang form processing, setiap function yang dipanggil sebagai *event handler* otomatis menerima 1 argument yang berisi *event object*. Event object inilah yang ditampung oleh parameter `e` di dalam pendefinisian fungsi `handleHapusMahasiswa()`. Dengan demikian, property `e.target.id` bisa diakses untuk mendapatkan nomor `nim` dari tombol "Hapus".

Dengan selesainya fitur hapus mahasiswa ini, kode program untuk aplikasi CRUD mahasiswa juga sudah selesai. Sebagai perbandingan, berikut saya tampilkan kode lengkap dari komponen App:

src\App.js

```

1 import React, { useState } from 'react';
2 import RowMahasiswa from './components/RowMahasiswa';
3 import RowTambahMahasiswa from './components/RowTambahMahasiswa';
4
5 // Data awal tabel mahasiswa
6 const arrMahasiswas = [
7   {
8     nim: "18010245",
9     nama: "Eka Putra",
10    jurusan: "Teknik Informatika",
11    asalProvinsi: "DKI Jakarta"
12  },
13  {
14    nim: "19010214",
15    nama: "Lisa Permata",

```

```

16     jurusan: "Sistem Informasi",
17     asalProvinsi: "Sumatera Barat"
18   },
19   {
20     nim: "20010710",
21     nama: "Rudi Setiawan",
22     jurusan: "Ilmu Komputer",
23     asalProvinsi: "Jawa Tengah"
24   },
25   {
26     nim: "20010790",
27     nama: "Friska Ramadhani",
28     jurusan: "Ilmu Komputer",
29     asalProvinsi: "Kalimantan Barat"
30   }
31 ];
32
33 const App = () => {
34   const [mahasiswas, setMahasiswas] = useState(arrMahasiswas);
35
36   // handler untuk menambah data mahasiswa,
37   // akan di-trigger dari komponen RowTambahMahasiswa
38   const handleTambahMahasiswa = (data) => {
39     const newMahasiswas = [
40       ...mahasiswas, data
41     ];
42     setMahasiswas(newMahasiswas);
43   }
44
45   // handler untuk mengedit data mahasiswa
46   // akan di-trigger dari komponen RowMahasiswa
47   const handleEditMahasiswa = (data) => {
48     // cari index dari mahasiswa yang akan diedit berdasarkan nomor nim
49     const result = mahasiswas.findIndex(
50       (mahasiswa) => mahasiswa.nim === data.nim
51     );
52
53     // copy mahasiswas karena fungsi splice akan mengubah array asal (mutate)
54     const newMahasiswas = mahasiswas;
55     newMahasiswas.splice(result, 1, data);
56     setMahasiswas([...newMahasiswas]);
57
58     // !! jika hanya menggunakan setMahasiswas(newMahasiswas),
59     // react tidak akan me-re-render halaman karena
60     // newMahasiswas = mahasiswa masih merujuk ke object yang sama.
61   }
62
63   // handler untuk menghapus data mahasiswa di komponen RowMahasiswa
64   const handleHapusMahasiswa = (e) => {
65
66     // cari index dari mahasiswa yang akan dihapus berdasarkan nomor nim
67     const result = mahasiswas.findIndex(
68       (mahasiswa) => mahasiswa.nim === e.target.id
69     );
70

```

```
71     // copy mahasiswas karena fungsi splice akan mengubah array asal (mutate)
72     const newMahasiswas = mahasiswas;
73     newMahasiswas.splice(result, 1);
74     setMahasiswas([...newMahasiswas]);
75
76     // Cara alternatif penghapusan dengan method filter
77     // const newMahasiswas = mahasiswas.filter(
78     //   mahasiswa => mahasiswa.nim !== e.target.id
79     // );
80     // setMahasiswas(newMahasiswas);
81   }
82
83   return (
84     <div className="container mt-5">
85
86       <div className="row mt-5">
87         <div className="col">
88           <h1 className="text-center">Tabel Mahasiswa</h1>
89
90           <table className="table mt-4">
91             <thead>
92               <tr>
93                 <th>NIM</th>
94                 <th>Nama</th>
95                 <th>Jurusan</th>
96                 <th>Asal Provinsi</th>
97                 <th></th>
98               </tr>
99             </thead>
100            <tbody>
101              {
102                mahasiswas.map((mahasiswa) =>
103                  <RowMahasiswa
104                    key={mahasiswa.nim}
105                    mahasiswa={mahasiswa}
106                    onEditMahasiswa={handleEditMahasiswa}
107                    onHapusMahasiswa={handleHapusMahasiswa}
108                  />
109                )
110              }
111              <RowTambahMahasiswa
112                mahasiswa={mahasiswas}
113                onTambahMahasiswa={handleTambahMahasiswa}
114              />
115            </tbody>
116          </table>
117        </div>
118
119      </div>
120    )
121  )
122 }
123 }
124
125 export default App;
```

Kode program ini saya tambah beberapa baris komentar untuk memudahkan pemahaman. Dan berikut kode program akhir dari komponen RowMahasiswa:

```
src\components\RowMahasiswa.js

1 import React, { useState } from 'react';
2
3 const RowMahasiswa = (props) => {
4
5     // simpan props mahasiswa ke dalam state agar mudah diakses
6     const [formInput, setFormInput] = useState({
7         nim: props.mahasiswa.nim,
8         nama: props.mahasiswa.nama,
9         jurusan: props.mahasiswa.jurusan,
10        asalProvinsi: props.mahasiswa.asalProvinsi,
11    });
12
13     // state untuk menampung pesan error
14     const [errors, setErrors] = useState({
15         nama: "",
16         jurusan: "",
17         asalProvinsi: ""
18    });
19
20     // state untuk penanda "Edit mode"
21     const [editStatus, setEditStatus] = useState(false);
22
23     // state untuk menampung nilai form sebelum "Edit mode"
24     const [dataReset, setDataReset] = useState({});
```

25

```
26     // function untuk membuat 2 ways binding antara form dengan state
27     const handleInputChange = (event) => {
28         setFormInput({ ...formInput, [event.target.name]: event.target.value })
29    }
30
31     // tombol Edit di klik
32     const handleEditClick = () => {
33
34         // Simpan data untuk proses reset
35         setDataReset({ ...formInput });
36
37         // Masuk ke "Edit mode"
38         setEditStatus(true);
39    }
40
41     // tombol Batal di klik
42     const handleFormReset = (e) => {
43         e.preventDefault();
44
45         // Kembalikan isi form ke posisi sebelum tombol edit di klik
46         setFormInput({ ...dataReset });
47
48         // Hapus pesan error (jika ada)
49         setErrors({});
```

```
50      // Keluar dari "Edit mode"
51      setEditStatus(false);
52  }
53
54
55  // form di submit
56  const handleFormSubmit = (e) => {
57    e.preventDefault();
58    let pesanErrors = {};
59
60    // validasi nama
61    if (formInput.nama.trim() === "") {
62      pesanErrors.nama = "Nama tidak boleh kosong";
63    }
64    else {
65      pesanErrors.nama = "";
66    }
67
68    // validasi jurusan
69    if (formInput.jurusan.trim() === "") {
70      pesanErrors.jurusan = "Jurusan tidak boleh kosong";
71    }
72    else {
73      pesanErrors.jurusan = "";
74    }
75
76    // validasi asalProvinsi
77    if (formInput.asalProvinsi.trim() === "") {
78      pesanErrors.asalProvinsi = "Asal Provinsi tidak boleh kosong";
79    }
80    else {
81      pesanErrors.asalProvinsi = "";
82    }
83
84    // update error state
85    setErrors(pesanErrors);
86
87    // cek apakah seluruh form valid atau masih ada error
88    let formValid = true;
89    for (let propName in pesanErrors) {
90      if (pesanErrors[propName].length > 0) {
91        formValid = false;
92      }
93    }
94
95    // proses data jika form valid
96    if (formValid) {
97      props.onEditMahasiswa(formInput);
98      setEditStatus(false);
99    }
100  }
101
102 return (
103   <React.Fragment>
104     /* Tampilkan form jika tombol Edit di klik, atau tampilkan row normal
```

```

*/}
105     {editStatus ?
106         <tr>
107             <td colSpan="5">
108                 <form onSubmit={handleFormSubmit} onReset={handleFormReset}>
109                     <div className="row row-cols-5 g-3">
110                         <div className="col">
111                             <input type="text" className="form-control" disabled
112                                 defaultValue={formInput.nim} name="nama" />
113                         </div>
114                         <div className="col">
115                             <input type="text" className="form-control" name="nama"
116                                 onChange={handleInputChange} value={formInput.nama} />
117                             {errors.nama && <small>{errors.nama}</small>}
118                         </div>
119                         <div className="col">
120                             <input type="text" className="form-control" name="jurusan"
121                                 onChange={handleInputChange} value={formInput.jurusan} />
122                             {errors.jurusan && <small>{errors.jurusan}</small>}
123                         </div>
124                         <div className="col">
125                             <input type="text" className="form-control"
126                                 name="asalProvinsi" onChange={handleInputChange}
127                                 value={formInput.asalProvinsi} />
128                             {errors.asalProvinsi && <small>{errors.asalProvinsi}</small>}
129                         </div>
130                         <div className="col">
131                             <button className="btn btn-success me-2" type="submit">
132                                 Simpan</button>
133                             <button className="btn btn-warning" type="reset">
134                                 Batal</button>
135                             </div>
136                         </div>
137                     </form>
138                 </td>
139             </tr>
140             :
141             <tr>
142                 <td>{formInput.nim}</td>
143                 <td>{formInput.nama}</td>
144                 <td>{formInput.jurusan}</td>
145                 <td>{formInput.asalProvinsi}</td>
146                 <td>
147                     <button className="btn btn-secondary me-2"
148                         onClick={handleEditClick}>Edit</button>
149                     <button className="btn btn-danger" id={formInput.nim}
150                         onClick={props.onHapusMahasiswa}>Hapus</button>
151                 </td>
152             </tr>
153         }
154     </React.Fragment >
155 )
156 }
157 }
158

```

```
159 export default RowMahasiswa;
```

Kode program ini memang cukup panjang, namun bisa menjadi referensi jika ingin membuat fitur yang serupa.

Fitur CRUD menjadi salah satu konsep dasar yang sangat sering kita pakai dalam pembuatan aplikasi. Meskipun data yang tersimpan hanya di memory, setidaknya bisa menjadi gambaran cara membuat aplikasi CRUD di React.

Pemrosesan data dari dan ke database sudah di luar ruang lingkup React sebagai framework JavaScript front-end, akan tetapi nantinya kita akan lihat bagaimana penerapannya menggunakan API.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

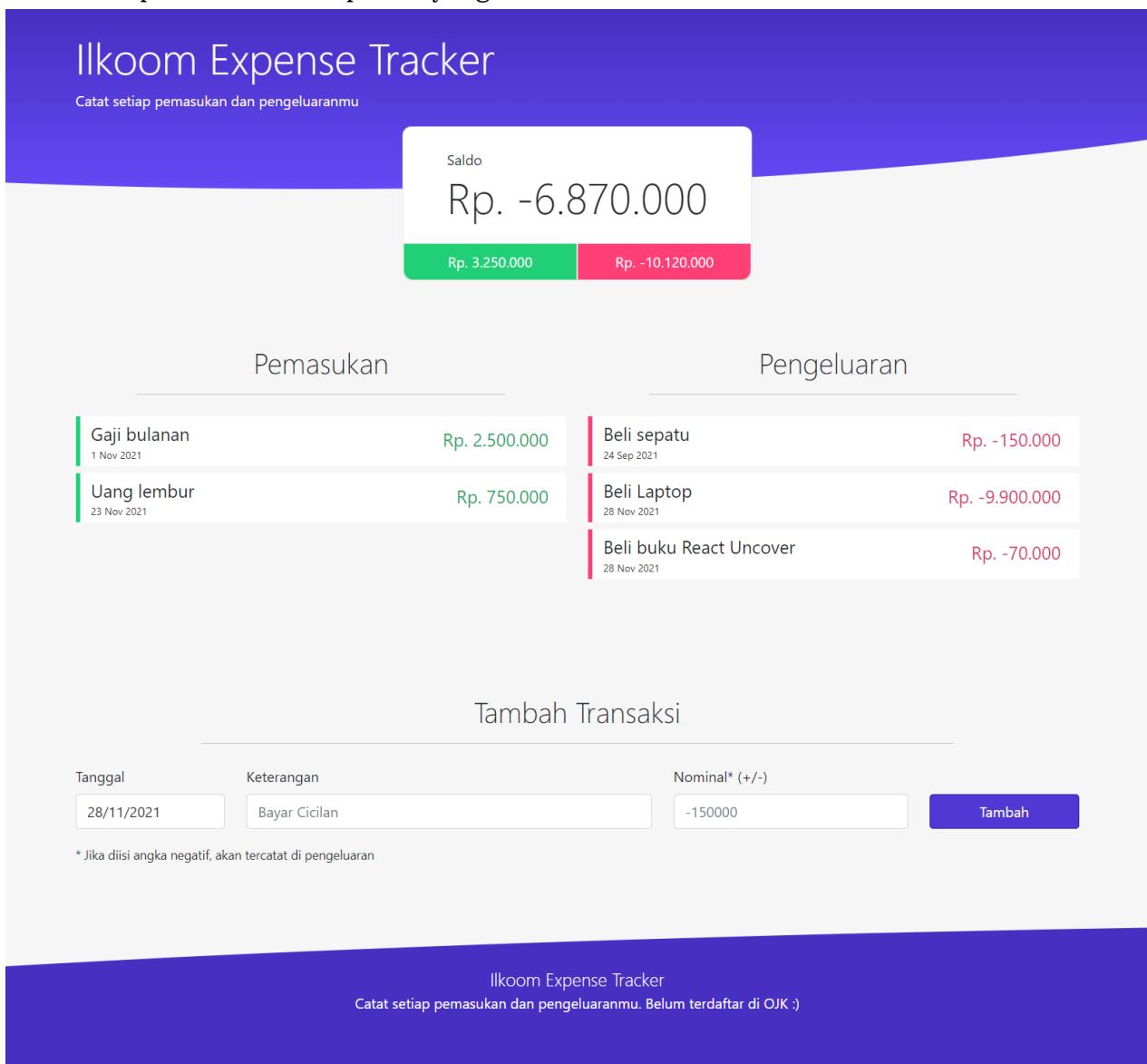
Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

17. Mini Project: Ilkoom Expense Tracker

Kita masuk ke mini project selanjutnya, yakni membuat aplikasi **Ilkoom Expense Tracker**. "Expense Tracker" merujuk ke sistem pencatatan pengeluaran, aplikasi ini akan menyimpan pemasukan dan pengeluaran user serta menampilkan total selisih antara keduanya. Sedangkan "Ilkoom" sekedar *branding* untuk merk saja, beberapa mini project di buku duniaIlkom lain juga menggunakan kata "Ilkoom".

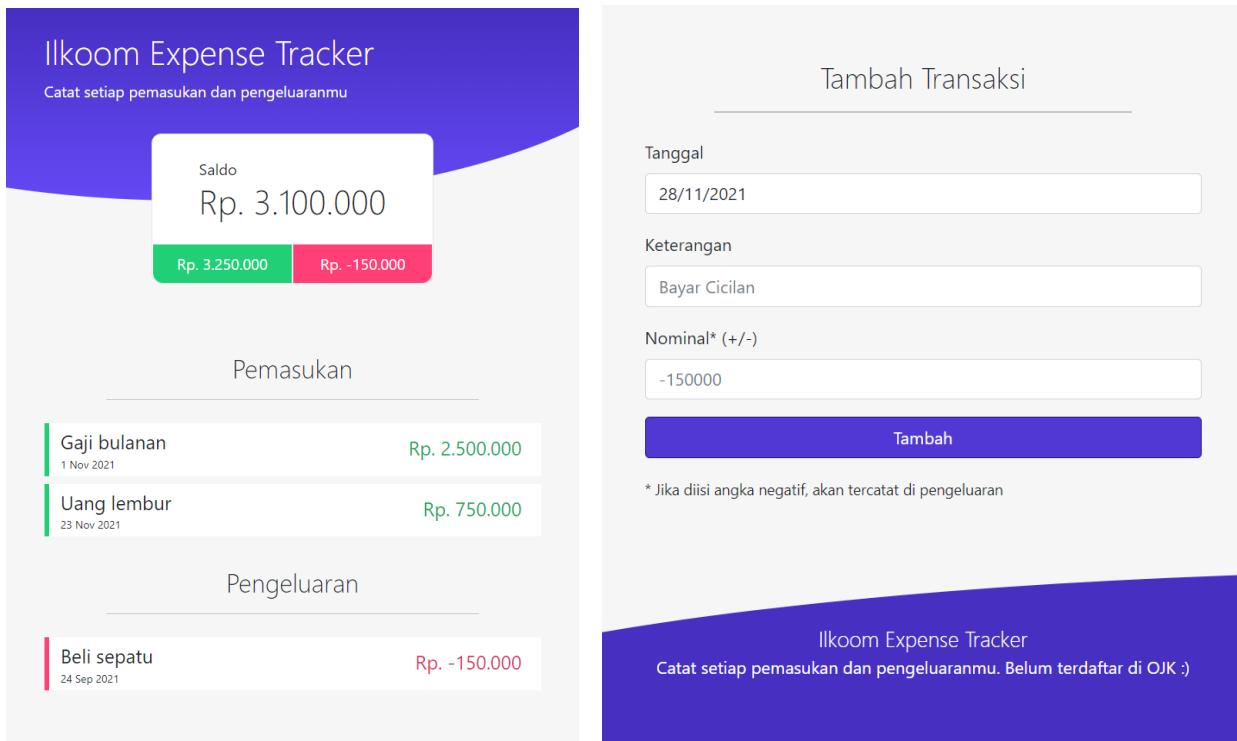
Berikut tampilan akhir dari aplikasi yang akan kita buat:



Gambar: Tampilan akhir halaman Ilkoom Expense Tracker

Dari form di bagian bawah, user bisa menginput pemasukan atau pengeluaran. Aplikasi akan mencatat setiap transaksi dan menampilkan total saldo di bagian atas. Jika saldo ini bernilai negatif, artinya user harus menghemat pengeluaran.

Sama seperti mini project CRUD mahasiswa, saya juga menggunakan framework CSS Bootstrap untuk membantu design tampilan, plus beberapa kode CSS dasar. Tampilan halaman juga sudah responsive:



Gambar: Tampilan halaman Ilkoom Expense Tracker di ukuran layar kecil

17.1. Persiapan Awal

Kita kembali menggunakan **create react app** dalam perancangan aplikasi. Anda boleh menginstall *create react app* baru atau bisa juga menggunakan file dari mini project sebelumnya (hapus seluruh file di folder `my-app\src`).

Langkah pertama adalah mengedit sedikit file `public\index.html`:

`public\index.html`

```
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="utf-8" />
5   <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <meta name="theme-color" content="#000000" />
8   <meta name="description" content="Aplikasi Ilkoom Expense Tracker -"
```

```
9     React Uncover DuniaIlkom" />
10    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
11    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
12    <title>Ilkoom Expense Tracker</title>
13  </head>
14
15  <body>
16    <noscript>You need to enable JavaScript to run this app.</noscript>
17    <div id="root"></div>
18  </body>
19 </html>
```

Perubahan ada di baris 2, 9-10 dan 13. Ini hanya untuk tampilan saja dan tidak berpengaruh ke kode React. Di baris 2 saya menukar atribut lang="en" menjadi lang="id", lalu di baris 9-10 menukar isi atribut content, serta di baris 13 mengganti isi tag <title>.

Selanjutnya kita juga butuh framework CSS Bootstrap. Jika belum tersedia, silahkan buka terminal, masuk ke folder **my-app**, lalu jalankan perintah berikut:

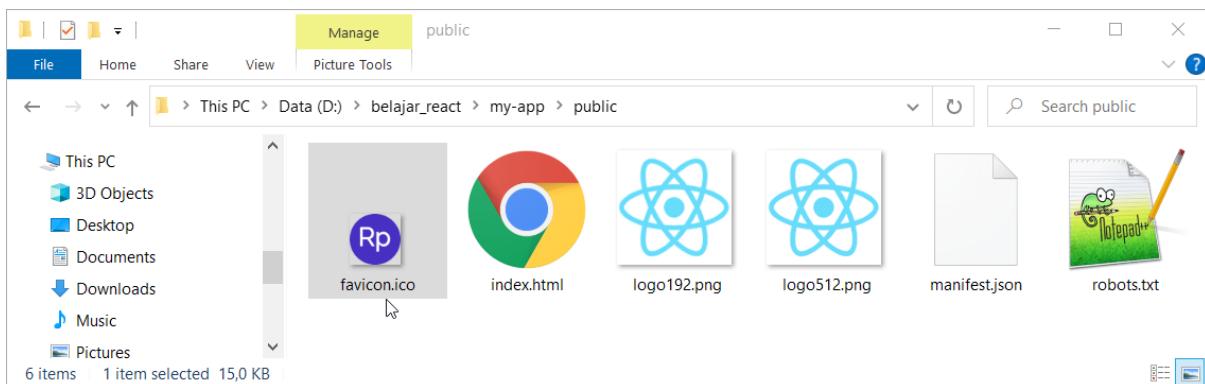
```
npm install bootstrap@5
```

Perintah ini akan mendownload file Bootstrap dan menyimpannya di folder **node_modules**:



Gambar: Menginstall Bootstrap dari terminal VS Code

Saya juga menyiapkan file **favicon.ico** yang bisa diakses dari file **belajar_react.zip** di Google Drive. Ini tidak wajib, hanya sedikit "pemanis" saja:



Gambar: File favicon untuk aplikasi Ilkoom Expense Tracker

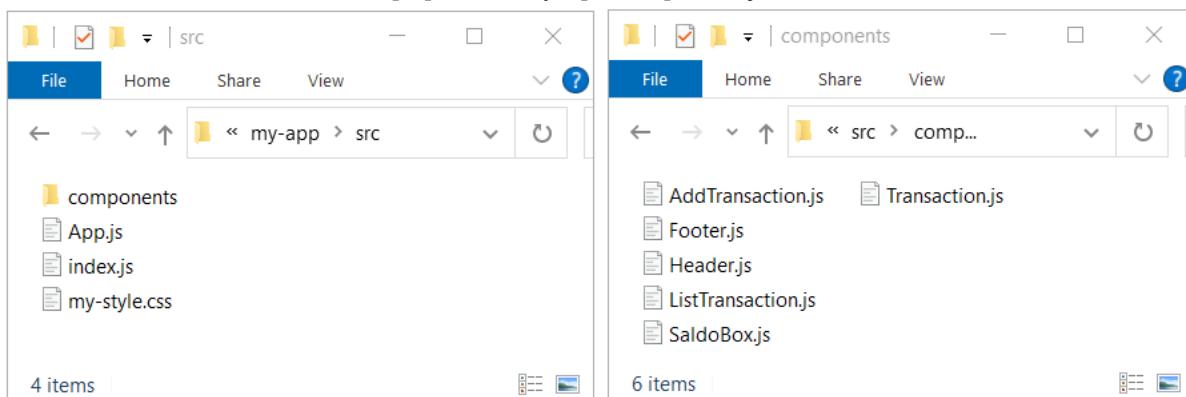
Ke dalam folder `my-app\src`, buat 3 file (isinya kosongkan saja terlebih dahulu):

- `App.js`
- `index.js`
- `my-style.css`

Kemudian buat juga folder **components** di dalam **src** dan isi dengan 6 file berikut:

- `AddTransaction.js`
- `Footer.js`
- `Header.js`
- `ListTransaction.js`
- `SaldoBox.js`
- `Transaction.js`

Keenam file belum berisi kode apapun, hanya persiapan saja.



Gambar: Isi folder `my-app\src` dan `my-app\src\components`

Berdasarkan struktur file bisa dilihat kalau komponen yang akan kita pakai cukup banyak, tapi ini juga memudahkan dalam pengelolaan kode program.

Selanjutnya, buka file `src\index.js` dan isi dengan kode berikut:

`src\index.js`

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import 'bootstrap/dist/css/bootstrap.min.css'
5 import './my-style.css'
6
7 ReactDOM.createRoot(document.getElementById('root'))
8 .render(
9   <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
```

Kode untuk file `index.js` ini sama persis seperti di project CRUD mahasiswa. Kita mengakses

komponen App, import file `bootstrap.min.css`, serta import file `my-style.css`. Komponen `<App />` menjadi komponen utama aplikasi yang akan di-inject ke dalam tag `<div id="root">` di file `public\index.html`.

Kemudian, isi juga kode berikut ke dalam file `my-style.css`:

```
src\my-style.css

1  /* ===== */
2  /* Global Style */
3  /* ===== */
4
5  :root{
6      --warna-pemasukan : #1dcc70;
7      --warna-pengeluaran : #ff396f;
8  }
9
10 /* Agar container tidak terlalu lebar */
11 @media (min-width: 1200px) {
12     .container {
13         max-width: 1140px !important;
14     }
15 }
16
17 body {
18     background-color: whitesmoke;
19 }
20
21 /* ===== */
22 /* Header */
23 /* ===== */
24
25 #header {
26     background-color: #412abe;
27     background: linear-gradient(180deg, rgba(65, 42, 190, 1) 0%,
28                             rgba(94, 67, 244, 1) 100%);
29     clip-path: ellipse(87% 100% at 31.5% 0%);
30     color: white;
31     height: 200px;
32     z-index: 1;
33 }
34
35 /* ===== */
36 /* Saldo Box */
37 /* ===== */
38
39 #saldo-box .container {
40     display: flex;
41     flex-direction: column;
42     justify-content: center;
43     align-items: center;
44 }
45
46 .total-saldo {
```

```
47    border-radius: 10px 10px;
48    background: white;
49    margin-top: -70px;
50    z-index: 2;
51    border: 1px solid rgb(68 68 68 / 15%);
52  }
53
54 .mini-saldo{
55   display: flex;
56   flex-direction: column;
57   justify-content: center;
58   align-items: center;
59   color: white;
60   margin: 0;
61   text-align: center;
62   flex: 1 1;
63 }
64
65 .mini-saldo-pemasukan {
66   border-bottom-left-radius: 10px;
67   border-right: 1px solid #eee;
68   background-color: var(--warna-pemasukan);
69 }
70
71 .mini-saldo-pengeluaran {
72   border-bottom-right-radius: 10px;
73   border-left: 1px solid #eee;
74   background-color: var(--warna-pengeluaran);
75 }
76
77 /* ===== */
78 /* Transaction */
79 /* ===== */
80
81 .list-transaction {
82   display: flex;
83   justify-content: space-between;
84   align-items: center;
85   background-color: white;
86   position: relative;
87 }
88 .list-transaction small {
89   font-size: 0.7rem;
90 }
91
92 .list-pemasukan {
93   border-left: 5px solid var(--warna-pemasukan);
94 }
95
96 .list-pengeluaran {
97   border-left: 5px solid var(--warna-pengeluaran);
98 }
99
100 .list-pemasukan .text {
101   color: #1aa35b;
```

```
102   white-space: nowrap;
103 }
104
105 .list-pengeluaran .text {
106   color: #d6325f;
107   white-space: nowrap;
108 }
109
110 .list-transaction .delete-icon {
111   background-color: var(--warna-pengeluaran);
112   color: #ffffff;
113   position: absolute;
114   right: 0;
115   top: 0;
116   width: 20px;
117   height: 20px;
118   text-align: center;
119   font-size: 16px;
120   cursor: pointer;
121   border-top-right-radius: 3px 3px;
122   line-height: 16px;
123   opacity: 0;
124 }
125 .list-transaction:hover .delete-icon {
126   opacity: 1;
127   transition: all ease-in-out 0.4s;
128 }
129
130 /* ===== */
131 /* Form */
132 /* ===== */
133
134 .btn-primary {
135   background-color: #4b32d0;
136   border-color: #221088;
137 }
138
139 .btn-primary:hover {
140   background-color: #3924a7;
141   border-color: #261773;
142 }
143
144 .btn-primary:focus {
145   background-color: #4b32d0;
146   border-color: #221088;
147   box-shadow: 0 0 0 0.25rem rgb(200 191 255);
148 }
149
150 .error-box {
151   color: red;
152 }
153
154 .error-box ul {
155   border: 1px solid var(--warna-pengeluaran);
156 }
```

```
157
158 .error-box p {
159   margin: 0;
160 }
161
162 /* ===== */
163 /* Footer */
164 /* ===== */
165
166 #footer {
167   background-color: #412abe;
168   clip-path: ellipse(189% 100% at 143.13% 100%);
169 }
```

Untuk project ini saya menggunakan cukup banyak kode CSS. Di awal terdapat pendefinisian CSS variable --warna-pemasukan dan --warna-pengeluaran, sebagai kode warna untuk beberapa property CSS lain. Pengaturan desain layout menggunakan CSS Flexbox agar lebih rapi.

Jika butuh penjelasan detail dari setiap kode CSS yang ada, DuniaIlkom juga menyediakan buku/eBook [CSS Uncover](#).

17.2. Komponen Header dan Footer

Project **Ilkoom Expense Tracker** di pecah menjadi beberapa komponen. Diantaranya terdapat 2 komponen yang hanya berisi kode JSX untuk tampilan saja. Komponen yang dimaksud adalah **Header** dan **Footer**. Sesuai namanya, kedua komponen bertanggung jawab untuk menampilkan bagian atas (*header*) dan bagian bawah (*footer*) halaman web.

Silahkan buka file `components\Header.js` dan isi dengan kode berikut:

`src\components\Header.js`

```
1 const Header = () => {
2   return (
3     <header id="header">
4       <div className="container py-4">
5         <h1 className="display-5">Ilkoom Expense Tracker</h1>
6         <p>Catat setiap pemasukan dan pengeluaranmu</p>
7       </div>
8     </header>
9   )
10 }
11
12 export default Header;
```

Kemudian buka juga file `components\Footer.js` dan isi dengan kode berikut:

src\components\Footer.js

```
1 const Footer = () => {
2   return (
3     <footer id="footer" className="text-white py-5">
4       <div className="container">
5         <div className="row">
6           <div className="col text-center">
7             <p className="lead mb-0">Ilkoom Expense Tracker</p>
8             <p>Catat setiap pemasukan dan pengeluaranmu.
9               Belum terdaftar di OJK :)</p>
10            </div>
11          </div>
12        </div>
13      </footer>
14    )
15  }
16
17 export default Footer;
```

Kedua komponen hanya berisi kode JSX dengan berbagai tag HTML.

Pemisahan komponen visual seperti ini bisa dilakukan setelah template awal project selesai. Misalnya di awal kita buat template dari kode HTML dan CSS terlebih dahulu, baru setelah desain selesai, template dipecah menjadi berbagai komponen React.

Agar hasilnya bisa terlihat, silahkan edit file App.js dan isi dengan kode berikut:

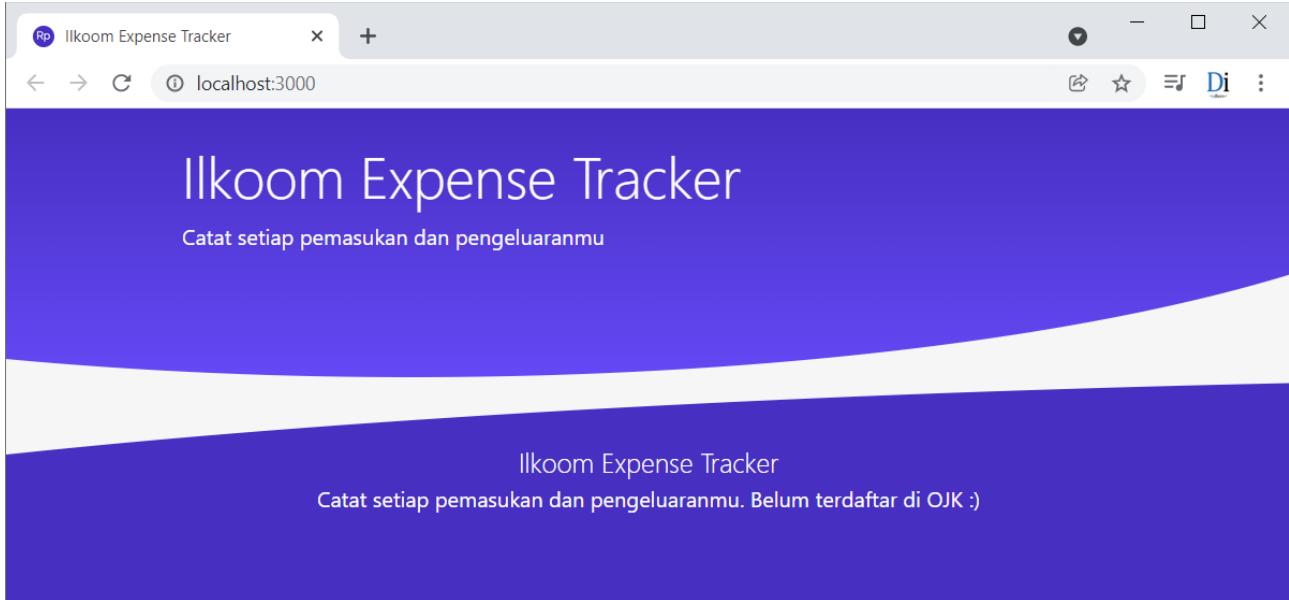
src\App.js

```
1 import React from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4
5 const App = () => {
6   return (
7     <React.Fragment>
8       <Header />
9       <Footer />
10      </React.Fragment>
11    )
12  }
13
14 export default App;
```

Komponen App menjadi komponen utama dari aplikasi kita. Komponen ini akan di update secara berkala seiring pembahasan materi.

Untuk saat ini komponen App masih sangat dasar. Komponen Header dan Footer di-import di baris 2-3, lalu ditampilkan ke dalam kode JSX di baris 8-9. Tag <React.Fragment> dipakai sebagai container dari kedua komponen.

Save file di atas, jalankan `npm start` dan cek hasilnya di web browser:



Gambar: Tampilan bagian header dan footer Ilkoom Expense Tracker

Efek visual yang terlihat berasal dari beberapa class Bootstrap di dalam kode JSX serta tambahan kode CSS di file `my-style.css`. Efek garis melengkung didapat dari penggunaan property `clip-path` milik CSS3.

Hasilnya memang terlihat agak aneh karena konten utama website masih belum ada. Meskipun begitu, langkah ini memperlihatkan kode program kita sudah sukses berjalan dan tidak ada error dari kode React.

17.3. Komponen Transaction dan ListTransaction

Kita masuk ke pembuatan komponen utama dari Ilkoom Expense Tracker, yakni menampilkan daftar transaksi. Silahkan tambah data `sample` ke dalam komponen App:

src\App.js

```
1 import React, { useState } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4
5 const initTransactions = [
6   {
7     "id": "619941539079",
8     "tanggal": new Date("01 Nov 2021 9:30").getTime(),
9     "keterangan": "Gaji bulanan",
10    "nominal": 2500000,
11  },
12  {
13    "id": "749179155708",
14    "tanggal": new Date("23 Nov 2021 10:00").getTime(),
15    "keterangan": "Uang lembur ",
```

```
16     "nominal": 750000,  
17   },  
18   {  
19     "id": "568004092688",  
20     "tanggal": new Date("24 Sept 2021 10:30").getTime(),  
21     "keterangan": "Beli sepatu",  
22     "nominal": -150000,  
23   }  
24 ];  
25  
26 const App = () => {  
27   const [transactions, setTransaction] = useState(initTransactions);  
28   console.log(transactions);  
29  
30   return (  
31     <React.Fragment>  
32       <Header />  
33       <Footer />  
34     </React.Fragment>  
35   )  
36 }  
37  
38 export default App;
```

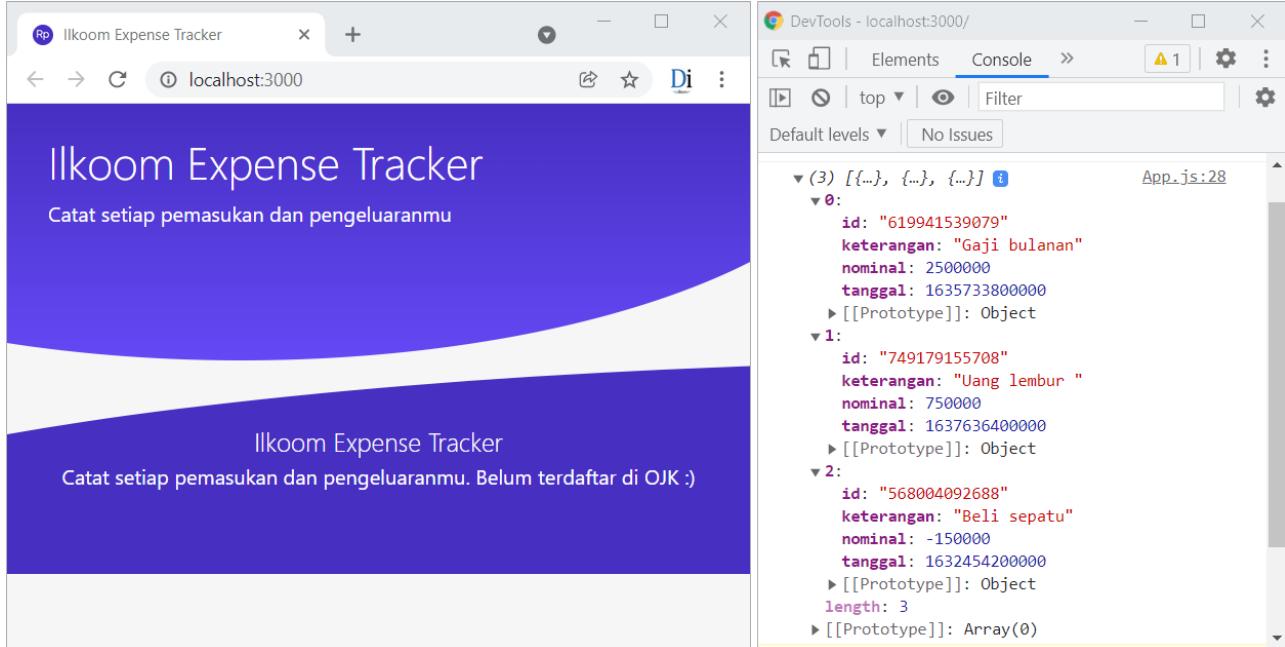
Di baris 1 terdapat tambahan import alias `{useState}` karena dalam kode ini kita sudah menggunakan state.

Di baris 6-25 konstanta `initTransactions` diisi dengan array yang terdiri dari 3 object. Setiap object memiliki property **id**, **tanggal**, **keterangan** dan **nominal**:

- Property **id** berisi 12 angka sebagai penanda identitas setiap transaksi. Fungsi utama dari property ini sekedar untuk mengisi property key yang di syaratkan React.
- Property **tanggal** berisi tanggal saat transaksi diinput. Di sini saya menggunakan **Date** object bawaan JavaScript untuk mengkonversi tanggal menjadi timestamp. Hasilnya, property **tanggal** berisi angka seperti 1635733800000. Nantinya tanggal ini akan kita generate otomatis dengan mengambil tanggal sistem (tanggal di komputer).
- Property **keterangan** berisi penjelasan dari setiap transaksi. Ini bisa diisi suka-suka oleh user, seperti "Hasil lembur" atau "Beli pulsa" sesuai dengan tujuan transaksi tersebut.
- Property **nominal** berisi angka transaksi. Angka ini bisa positif atau negatif tergantung apakah itu pemasukan atau pengeluaran.

Konstanta `initTransactions` selanjutnya menjadi nilai awal dari state `transactions` di baris 28. Disinilah kita mengelola seluruh transaksi dalam aplikasi Ilkoom Expense Tracker.

Di baris 29 saya menambah perintah `console.log(transactions)` sekedar untuk melihat apakah isi state `transactions` sesuai dengan keinginan atau tidak. Save kode di atas, dan lihat ke dalam tab console:



Gambar: Isi state transactions terlihat di tab console

Sejauh ini tidak ada masalah, di dalam tab Console bisa terlihat isi dari state `transactions` dengan data yang sesuai.

Berikutnya, state `transactions` akan kita tampilkan ke dalam web browser. Proses menampilkan data ini akan menjadi tugas dari komponen `Transaction` dan `ListTransaction`. Komponen `Transaction` berfungsi sebagai "container" untuk membagi posisi tampilan apakah termasuk kategori pemasukan, atau kategori pengeluaran.

Sebelum ke sana, silahkan modifikasi lagi komponen App dengan tambahan kode berikut:

```
src\App.js
1 import React, { useState } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
5
6 const initTransactions = [
7   {
8     ...
9   }
10];
11
12 const App = () => {
13   const [transactions, setTransaction] = useState(initTransactions);
14
15   return (
16     <React.Fragment>
17       <Header />
18       <Transaction transactions={transactions} />
19       <Footer />
```

```
20      </React.Fragment>
21    )
22 }
23
24 export default App;
```

Tambahan kode program ada di baris 4 untuk mengimport file ./components/Transaction, serta memanggilnya dari dalam kode JSX di baris 18. Ke dalam tag <Transaction/> ikut dikirim atribut transactions={transactions}, sehingga isi state transactions bisa diakses sebagai props.transactions dari dalam komponen Transaction.

Buka file Transaction.js dan isi dengan kode program berikut:

src\components\Transaction.js

```
1 import ListTransaction from './ListTransaction';
2
3 const Transaction = (props) => {
4   return (
5     <section id="transaction">
6       <div className="container py-5">
7         <div className="row">
8
9           <div className="col-12 col-lg-6 mb-4" id="pemasukan">
10             <h2 className="fw-light mb-3 text-center">Pemasukan</h2>
11             <hr className="w-75 mx-auto mb-4" />
12           {
13             props.transactions.map((transaction) => (
14               (transaction.nominal > 0) &&
15               <ListTransaction
16                 key={transaction.id}
17                 id={transaction.id}
18                 tanggal={transaction.tanggal}
19                 keterangan={transaction.keterangan}
20                 nominal={transaction.nominal}
21               />
22             ))
23           }
24         </div>
25
26         <div className="col-12 col-lg-6 mb-4" id="pengeluaran">
27           <h2 className="fw-light text-center mb-3">Pengeluaran</h2>
28           <hr className="w-75 mx-auto mb-4" />
29         {
30           props.transactions.map((transaction) => (
31             (transaction.nominal <= 0) &&
32             <ListTransaction
33               key={transaction.id}
34               id={transaction.id}
35               tanggal={transaction.tanggal}
36               keterangan={transaction.keterangan}
37               nominal={transaction.nominal}
38             />
39           ))
40         }
41       </div>
42     </div>
43   </section>
44 }
```

```

39           ))
40       }
41     </div>
42   </div>
43   </div>
44 </section>
45 )
46 }
47 }
48
49 export default Transaction;

```

Meskipun agak panjang, mayoritas kode hanya berisi tag HTML, terutama tag `<div>` dengan berbagai class bawaan Bootstrap.

Di baris 1 terdapat perintah untuk mengimport komponen `ListTransaction`. Ini diperlukan karena proses menampilkan data transaksi akan saya teruskan ke komponen `ListTransaction`.

Komponen `Transaction` hanya bertugas membagi jenis transaksi saja, apakah transaksi itu masuk ke kelompok pemasukan, atau ke kelompok pengeluaran. Proses pemeriksaan ada di baris 13-14 dan juga di baris 30-31 lewat perulangan `props.transactions.map()`.

Perulangan ini dijalankan untuk semua object yang ada di dalam `props.transactions`. Sekedar pengingat, `props.transactions` berisi state `transactions` yang dikirim dari komponen `App`.

Jika nominal untuk transaksi tersebut bernilai positif, tempatkan transaksi di sisi kiri halaman (memenuhi syarat `transaction.nominal > 0` di baris 14). Atau jika nilai property `nominal` bernilai negatif, tempatkan transaksi di sisi kanan halaman (memenuhi syarat `transaction.nominal < 0` di baris 14).

Berikut tampilan yang saya inginkan (saat ini belum bisa diakses):

Pemasukan	Pengeluaran
Gaji bulanan 1 Nov 2021	Rp. 2.500.000
Uang lembur 23 Nov 2021	Rp. 750.000
	Beli sepatu 24 Sep 2021
	Rp. -150.000

Gambar: Pemisahan transaksi antara pemasukan dan pengeluaran

Tampilan sisi kiri dan kanan didapat dari class bawaan Bootstrap. Tag `<div>` di baris 9 dan 26 akan tampil sebanyak 6 segment di *breakpoint large* (class `.col-1g-6`), sehingga langsung terpisah di sisi kiri dan kanan. Namun jika halaman di perkecil, transaksi pemasukan akan tampil di sisi atas dan transaksi pengeluaran tampil di sisi bawah (class `.col-12`).

Untuk setiap perulangan `.map()`, data transaksi akan dikirim ke tag `<ListTransaction />` melalui atribut `id={transaction.id}`, `tanggal={transaction.tanggal}`, `keterangan=`

{transaction.keterangan}, dan nominal={transaction.nominal}. Semua data ini akan bisa diakses sebagai props dari dalam komponen ListTransaction.

Kode untuk mengakses tag <ListTransaction /> di baris 15-21 sebenarnya sama persis dengan baris 32-38. Perulangan ini hanya untuk memisahkan transaksi pemasukan dan transaksi pengeluaran.

Sekarang kita masuk ke kode untuk komponen ListTransaction:

src\components\ListTransaction.js

```
1  const ListTransaction = (props) => {
2
3    // Atur tanggal untuk tampilan
4    const getTanggal = (tanggal) => {
5      let dateObj = new Date(tanggal);
6
7      const arrBulan = ["Jan", "Feb", "Mar", "Apr", "Mei", "Jun",
8        "Jul", "Ags", "Sep", "Okt", "Nov", "Des"];
9
10     let bulan = arrBulan[dateObj.getMonth()];
11     return `${dateObj.getDate()} ${bulan} ${dateObj.getFullYear()}`;
12   }
13
14  // untuk menentukan style list transaction (warna border kiri)
15  let classTransaction = (props.nominal > 0) ?
16    "list-pemasukan" : "list-pengeluaran";
17
18  return (
19    <div className={`list-transaction p-1 mb-2 ${classTransaction}`}>
20      <div className="ms-2 d-flex flex-column">
21        <p className="m-0 fs-5">{props.keterangan}</p>
22        <small>{getTanggal(props.tanggal)}</small>
23      </div>
24      <p className="fs-5 mb-1 me-3 text">
25        Rp. {props.nominal.toLocaleString('id-ID')}
26      </p>
27      <span className="delete-icon">x</span>
28    </div>
29  )
30 }
31
32 export default ListTransaction;
```

Komponen ListTransaction bertugas untuk menampilkan 1 data transaksi berikut:

Gaji bulanan	Rp. 2.500.000
1 Nov 2021	

Gambar: Tampilan 1 transaksi

Data transaksi terdiri dari 3 komponen: keterangan yang dalam hasil di atas tampil sebagai "Gaji bulanan", tanggal dalam format "1 Nov 2021" serta nominal dalam bentuk "Rp. 2.500.000".

Data keterangan bisa kita ambil langsung dari `props.keterangan` seperti di baris 21. Akan tetapi untuk data `tanggal` dan `nominal` perlu sedikit pemrosesan lebih lanjut.

Sesampainya di komponen `ListTransaction`, data `props.tanggal` berisi angka timestamp seperti 1635733800000. Angka ini akan dikonversi menjadi tanggal biasa oleh fungsi `getTanggal()` antara baris 4-12.

Fungsi `getTanggal()` menerima 1 parameter dalam format `timestame`, yang kemudian dipakai untuk men-generater **Date object** milik JavaScript di baris 5. Data object ini selanjutnya disimpan ke dalam variabel `dateObj`.

Jika sudah berbentuk Date object, kita bisa mengakses berbagai method bawaan JavaScript dengan cara meng-extract komponen tanggal, diantaranya `dateObj.getDate()` untuk menampilkan angka tanggal, `dateObj.getMonth()` untuk menampilkan digit, dan `dateObj.getFullYear()` untuk menampilkan 4 digit tahun. Berikut contohnya dalam JavaScript native:

```
1 <script type="text/javascript">
2   let dateObj = new Date(1635733800000);
3
4   console.log(dateObj.getDate());      // 1
5   console.log(dateObj.getMonth());     // 10
6   console.log(dateObj.getFullYear());  // 2021
7 </script>
```

Hasil dari digit bulan sedikit "aneh" karena mulai dari angka 0 untuk bulan Januari. Namun ini menjadi nilai yang pas sebagai index array. Itulah fungsi dari konstanta `arrBulan` yang berisi array inisial nama bulan di baris 7-8. Nama bulan ini kemudian diakses pada baris 10.

Hasilnya, perintah `return` di baris 11 akan menampilkan string seperti "1 Nov 2021". Fungsi `getTanggal()` kemudian diakses dari dalam kode JSX di baris 22.

Untuk nilai `nominal`, saya menggunakan method `toLocaleString()` bawaan JavaScript untuk menformat tampilan angka agar terdapat tanda titik sebagai penanda digit ribuan. Method `toLocaleString()` butuh 1 argument berupa inisial nama negara dan bahasa yang dalam contoh ini diisi dengan '`ID-id`'.

Hasilnya, angka 2500000 akan tampil sebagai 2.500.000 sesuai format penulisan dalam Bahasa Indonesia. Proses ini dilakukan langsung dari dalam kode JSX di baris 25.

Di baris 15-16 terdapat pemeriksaan kondisi apakah `props.nominal` berisi angka > 0 atau tidak. Jika `true`, maka isi variabel `classTransaction` dengan string "`list-pemasukan`", atau jika `false` isi dengan string "`list-pengeluaran`".

Variabel `classTransaction` saya pakai di baris 19, tepatnya di dalam atribut `class`. Ini dipakai untuk menentukan warna border dari transaksi, apakah memiliki border hijau (untuk pemasukan) atau border merah (untuk pengeluaran).

Jika transaksi memiliki nominal > 0 , maka kode di baris 19 akan di proses sebagai:

```
<div className="list-transaction p-1 mb-2 list-pemasukan">
```

Dalam file `my-style.css`, saya sudah siapkan selector `.list-pemasukan` dengan kode berikut:

```
.list-pemasukan {  
  border-left: 5px solid var(--warna-pemasukan);  
}  
  
.list-pemasukan .text {  
  color: #1aa35b;  
  white-space: nowrap;  
}
```

Maka list transaction akan tampil dengan border dan teks nominal hijau:



Gambar: Tampilan untuk transaksi pemasukan

Akan tetapi jika transaksi yang di proses memiliki nominal ≤ 0 , maka kode di baris 19 akan di proses sebagai:

```
<div className="list-transaction p-1 mb-2 list-pengeluaran">
```

Di dalam file `my-style.css`, saya sudah siapkan selector `.list-pengeluaran` dengan kode sebagai berikut:

```
.list-pengeluaran {  
  border-left: 5px solid var(--warna-pengeluaran);  
}  
  
.list-pengeluaran .text {  
  color: #d6325f;  
  white-space: nowrap;  
}
```

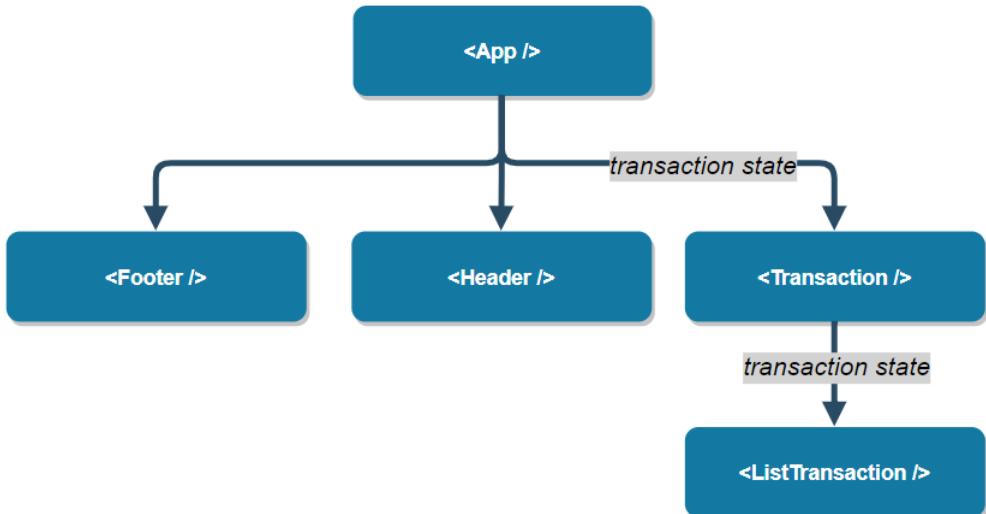
Sehingga list transaction akan tampil dengan border dan teks nominal merah:



Gambar: Tampilan untuk transaksi pengeluaran

Terakhir di baris 27 terdapat tag `x` yang saya siapkan sebagai tombol untuk menghapus transaksi. Tekniknya sama seperti di aplikasi Todo, yakni menampilkan tanda x di sudut kanan atas. Akan tetapi saat ini tombol tersebut belum berfungsi karena kita belum membuat event handler-nya.

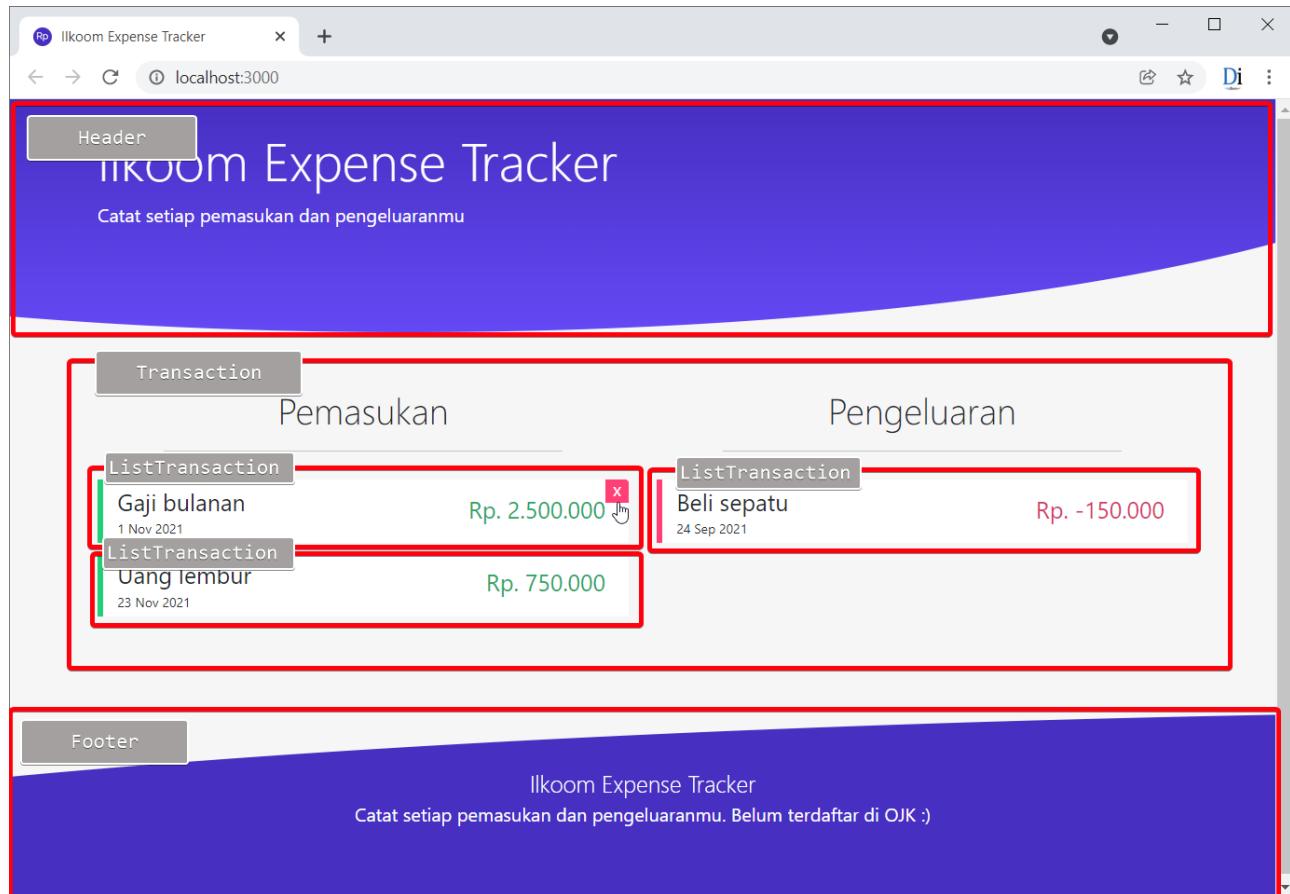
Huff... semoga anda bisa memahami alur pengiriman state dan interaksi antar komponen sejauh ini:



Gambar: Diagram komponen dari aplikasi Ilkoom Expense Tracker

Aplikasi kita sudah memiliki 5 komponen dengan App sebagai komponen utama. Komponen Footer dan Header hanya untuk design tampilan, sedangkan komponen Transaction dan ListTransaction dipakai untuk menampilkan transaksi. Kedua komponen ini menerima state transaction dari komponen App yang dikirim dalam bentuk *props*.

Berikut tampilan dari setiap komponen:



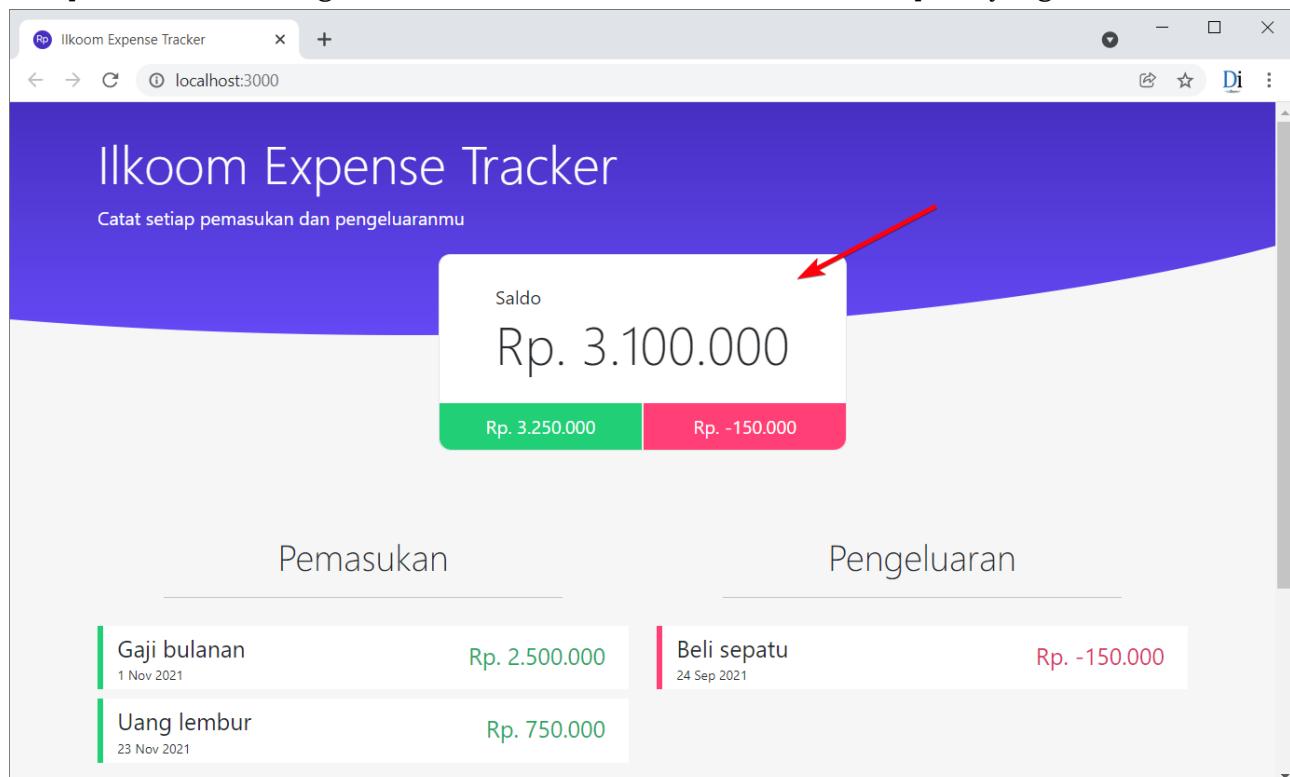
Gambar: Tampilan setiap komponen

Dalam aplikasi kita, komponen `ListTransaction` akan terus bertambah sesuai banyaknya transaksi.

Konsep pemisahan komponen dari React membuat pengelolaan kode program menjadi lebih sederhana. Kita tidak perlu khawatir setiap transaksi akan saling bentrok, karena berada di dalam ruang lingkup tersendiri.

17.4. Komponen SaldoBox

Komponen SaldoBox berfungsi untuk menampilkan total saldo dari semua transaksi. Posisi komponen ini ada dibagian atas sebelum list transaksi. Berikut tampilan yang akan kita buat:



Gambar: Tampilan komponen SaldoBox

Terdapat 3 angka di dalam `SaldoBox`. Angka di sisi atas merupakan total semua pemasukan dikurangi semua pengeluaran. Sedangkan angka di bagian bawah adalah total dari pemasukan saja (background hijau) dan total dari pengeluaran saja (background merah).

Langkah awal, tambah perintah import komponen `SaldoBox` ke dalam `App.js`, lalu akses juga dari dalam kode JSX:

```
src\App.js
1 import React, { useState } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
```

```
5 import SaldoBox from './components/SaldoBox';
6
7 const initTransactions = [
8   ...
9 ];
10
11 const App = () => {
12   const [transactions, setTransaction] = useState(initTransactions);
13
14   return (
15     <React.Fragment>
16       <Header />
17       <SaldoBox transactions={transactions} />
18       <Transaction transactions={transactions} />
19       <Footer />
20     </React.Fragment>
21   )
22 }
23
24 export default App;
```

Tambahan kode program ada di baris 5 dan 17. Atribut `transactions={transactions}` ikut ditulis ke dalam tag `<SaldoBox />` agar data state `transactions` bisa diakses dari dalam komponen `SaldoBox` sebagai `props`.

Selanjutnya tambahkan kode program berikut ke dalam file `SaldoBox.js`:

src\components\SaldoBox.js

```
1 import React, { useState, useEffect } from 'react';
2
3 const SaldoBox = (props) => {
4   const [saldoPemasukan, setSaldoPemasukan] = useState(0);
5   const [saldoPengeluaran, setSaldoPengeluaran] = useState(0);
6
7   useEffect(() => {
8     let totalPemasukan = 0;
9     let totalPengeluaran = 0;
10
11    // Hitung saldo pemasukan dan pengeluaran
12    props.transactions.forEach((transaction) => {
13      if (transaction.nominal > 0) {
14        totalPemasukan += transaction.nominal;
15      }
16      else {
17        totalPengeluaran += transaction.nominal;
18      }
19    })
20    setSaldoPemasukan(totalPemasukan);
21    setSaldoPengeluaran(totalPengeluaran);
22  }, [props.transactions])
23
24  return (
25    <section id="saldo-box">
```

```

26     <div className="container mb-4">
27       <div className="total-saldo">
28         <p className="pt-4 ps-5 mb-0">Saldo</p>
29         <h2 className="display-5 px-5 pb-3">
30           Rp. {(saldoPemasukan + saldoPengeluaran).toLocaleString('id-ID')}
31         </h2>
32         <div className="d-flex justify-content-center">
33           <p className="mini-saldo mini-saldo-pemasukan py-2">
34             Rp. {saldoPemasukan.toLocaleString('id-ID')}
35           </p>
36           <p className="mini-saldo mini-saldo-pengeluaran py-2">
37             Rp. {saldoPengeluaran.toLocaleString('id-ID')}
38           </p>
39         </div>
40       </div>
41     </div>
42   </section>
43 )
44 }
45
46 export default SaldoBox;

```

Di awal komponen `SaldoBox`, terdapat pendefinisian 2 state: `saldoPemasukan` dan `saldoPengeluaran`. Sesuai namanya, kedua state dipakai untuk menampung total pemasukan dan pengeluaran.

Proses penghitungan saldo ada di dalam `useEffect` hook antara baris 7-22, ini diperlukan karena setiap kali terjadi penambahan atau pengurangan transaksi, total saldo harus dihitung ulang.

State yang menyimpan transaksi sebenarnya ada di dalam komponen `App`, namun isinya bisa kita akses dari `props.transactions` yang ikut dikirim ke dalam `SaldoBox`. Karena itulah `useEffect` ini memiliki *dependency* ke `props.transactions` seperti yang ditulis pada baris 22. Dengan kata lain, setiap kali terjadi perubahan nilai `props.transactions`, `useEffect` akan langsung berjalan dan total saldo ikut dihitung ulang.

Proses perhitungan total saldo cukup sederhana, itu dilakukan dengan perulangan `props.transactions.forEach()` antara baris 12 – 19. Dalam setiap iterasi, cek apakah itu masuk ke kelompok pemasukan atau pengeluaran, kemudian cari total nominal untuk setiap kelompok.

Di dalam perulangan `.forEach()`, total saldo disimpan ke dalam variabel `totalPemasukan` dan `totalPengeluaran`. Setelah itu kedua variabel di update ke dalam state `saldoPemasukan` dan `saldoPengeluaran` di baris 20 dan 21.

Untuk kode JSX, mayoritas diisi dengan tag HTML beserta beberapa class Bootstrap untuk membuat struktur tampilan. Total saldo diakses pada baris 30 dengan menghitung `saldoPemasukan + saldoPengeluaran`. Tambahan method `toLocaleString('id-ID')` dipakai untuk merapikan tampilan angka dengan tanda titik sebagai pemisah ribuan. Saldo pemasukan dan pengeluaran secara terpisah di akses di baris 34 dan 37.

Sebagai tambahan, saldo pengeluaran akan selalu bernilai negatif (atau 0 jika tidak ada pengeluaran sama sekali). Sehingga untuk menghitung total, cukup ditambahkan saja antara `saldoPemasukan + saldoPengeluaran`.

17.5. Komponen AddTransaction

Sekarang kita masuk ke komponen terakhir dan juga paling panjang, yakni komponen `AddTransaction`.

Kode untuk komponen ini cukup panjang karena melibatkan form untuk proses penambahan transaksi. Dan dimana ada form, perlu perintah untuk proses validasi dan menampilkan pesan error. Untungnya, teknik yang akan dipakai kurang lebih sama seperti mini project lain.

Berikut tampilan yang akan kita buat:

Tambah Transaksi

- Keterangan tidak boleh kosong
- Nominal tidak boleh kosong

Tanggal	Keterangan	Nominal* (+/-)	
27/11/2021	Bayar Cicilan	-150000	Tambah

* Jika diisi angka negatif, akan tercatat di pengeluaran

Ilkoom Expense Tracker
Catat setiap pemasukan dan pengeluaranmu. Belum terdaftar di OJK :)

Gambar: Tampilan komponen AddTransaction

Sedikit berbeda dengan contoh form sebelumnya, kali ini semua pesan error validasi ada di bagian atas form. Menggunakan class CSS bawaan Bootstrap, border form yang tidak lolos validasi juga akan berwarna merah. Posisi komponen `AddTransaction` ini ada di bagian halaman, yakni di bawah `ListTransaction` dan di atas `Footer`.

Sebelum masuk ke komponen `AddTransaction`, silahkan buka file `App.js` dan tambah kode berikut untuk proses import komponen:

```
src\App.js
1 import React, { useState } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
```

```
5 import SaldoBox from './components/SaldoBox';
6 import AddTransaction from './components/AddTransaction';
7
8 const initTransactions = [
9   ...
10 ];
11
12 const App = () => {
13   const [transactions, setTransaction] = useState(initTransactions);
14
15   return (
16     <React.Fragment>
17       <Header />
18       <SaldoBox transactions={transactions} />
19       <Transaction transactions={transactions} />
20       <AddTransaction />
21       <Footer />
22     </React.Fragment>
23   )
24 }
25
26 export default App;
```

Tambahan kode program ada di baris 6 dan 20, yakni untuk mengimport komponen AddTransaction dan mengaksesnya dari dalam kode JSX.

Selanjutnya, tambah kode program berikut ke dalam file AddTransaction.js:

src\components\AddTransaction.js

```
1 import React, { useState, useRef } from 'react';
2
3 const AddTransaction = (props) => {
4
5   // Format tampilan tanggal menjadi dd/mm/yyyy
6   const getTanggal = () => {
7     let dateObj = new Date();
8
9     // perlu tambahan padStart() agar tanggal tampil 2 digit dengan awal 0
10    let tanggal = dateObj.getDate().toString().padStart(2, "0");
11
12    // bulan harus ditambah 1 karena getMonth() mulai dari 0 untuk januari
13    let bulan = (dateObj.getMonth() + 1).toString().padStart(2, "0");
14
15    return `${tanggal}/${bulan}/${dateObj.getFullYear()}`;
16  }
17
18  const [formInput, setFormInput] = useState({
19    tanggal: getTanggal(),
20    keterangan: "",
21    nominal: "",
22  });
23
24  const [errors, setErrors] = useState({
```

```

25     tanggal: "",
26     keterangan: "",
27     nominal: "",
28 });
29
30 const formValid = useRef(true);
31
32 const handleInputChange = (event) => {
33   setFormInput({ ...formInput, [event.target.name]: event.target.value })
34 }
35
36 const handleFormSubmit = (e) => {
37   e.preventDefault();
38   let pesanErrors = {};
39
40   // validasi tanggal
41   if (formInput.tanggal.trim() === "") {
42     pesanErrors.tanggal = "Tanggal tidak boleh kosong";
43   }
44   else if (!/^[0-3][0-9]\/[0-1][0-9]\/[0-9]{4}$/.test(formInput.tanggal)) {
45     pesanErrors.tanggal = "Format tanggal tidak sesuai";
46   }
47   else {
48     pesanErrors.tanggal = "";
49   }
50
51   // validasi keterangan
52   if (formInput.keterangan.trim() === "") {
53     pesanErrors.keterangan = "Keterangan tidak boleh kosong";
54   } else {
55     pesanErrors.keterangan = "";
56   }
57
58   // validasi nominal
59   if (formInput.nominal.trim() === "") {
60     pesanErrors.nominal = "Nominal tidak boleh kosong";
61   }
62   else if (!/^[+|-]?\d+$/.test(formInput.nominal)) {
63     pesanErrors.nominal = "Nominal harus angka";
64   }
65   else {
66     pesanErrors.nominal = "";
67   }
68
69   // update error state
70   setErrors(pesanErrors);
71
72   // cek apakah seluruh form valid atau masih ada error
73   formValid.current = true;
74   for (let inputName in pesanErrors) {
75     if (pesanErrors[inputName].length > 0) {
76       formValid.current = false;
77     }
78   }
79

```

```

80    // proses data jika form valid
81    if (formValid.current) {
82
83        let tanggalInput = new Date();
84        tanggalInput.setDate(parseInt(formInput.tanggal.substr(0, 2)));
85        tanggalInput.setMonth(parseInt(formInput.tanggal.substr(3, 2) - 1));
86        tanggalInput.setFullYear(parseInt(formInput.tanggal.substr(6, 4)));
87
88        let transaction = {
89            "id": Math.floor(Math.random() * 100000000000).toString(),
90            "tanggal": tanggalInput.getTime(),
91            "keterangan": formInput.keterangan,
92            "nominal": parseInt(formInput.nominal),
93        };
94
95        console.log(transaction);
96
97        // kosongkan inputan form
98        setFormInput({
99            tanggal: getTanggal(),
100           keterangan: "",
101           nominal: ""
102        })
103    }
104 }
105
106 return (
107   <section id="add-transaction">
108     <div className="container py-5">
109       <h2 className="fw-light mb-3 text-center">Tambah Transaksi</h2>
110       <hr className="w-75 mx-auto mb-4" />
111       {!formValid.current && (
112         <div className="row">
113           <div className="col-12 col-lg-6 error-box my-2">
114             <ul className="py-3">
115               {errors.tanggal && <li> {errors.tanggal}</li>}
116               {errors.keterangan && <li> {errors.keterangan}</li>}
117               {errors.nominal && <li> {errors.nominal}</li>}
118             </ul>
119           </div>
120         </div>
121         <form onSubmit={handleFormSubmit}>
122           <div className="row">
123             <div className="col-12 col-md-3 col-lg-2 mb-3">
124               <label htmlFor="tanggal" className="form-label">Tanggal</label>
125               <input type="text" id="tanggal" name="tanggal"
126                 placeholder="dd/mm/yyyy"
127                 className={`form-control ${errors.tanggal && "is-invalid"}`}
128                 value={formInput.tanggal} onChange={handleInputChange} />
129             </div>
130             <div className="col-12 col-md-6 col-lg-5 mb-3">
131               <label htmlFor="Keterangan" className="form-label">
132                 Keterangan</label>
133               <input type="text" id="keterangan" name="keterangan"
134                 placeholder="Bayar Cicilan" />
135             </div>
136           </div>
137         </form>
138       )
139     </div>
140   </section>
141 )

```

```

135         className={`form-control ${errors.keterangan && "is-invalid"}`}
136         value={formInput.keterangan} onChange={handleInputChange} />
137     </div>
138     <div className="col-12 col-md-3 col-lg-3 mb-3">
139       <label htmlFor="nominal" className="form-label">
140         Nominal* (+/-)</label>
141       <input type="text" id="nominal" name="nominal"
142         placeholder="-150000"
143         className={`form-control ${errors.nominal && "is-invalid"}`}
144         value={formInput.nominal} onChange={handleInputChange} />
145     </div>
146     <div className="col-12 col-lg-2 mb-3 d-flex align-items-end">
147       <button type="submit" className="btn btn-primary flex-fill">
148         Tambah</button>
149     </div>
150   </div>
151   <p><small>* Jika diisi angka negatif,
152     akan tercatat di pengeluaran</small></p>
153 </form>
154 </div>
155 </section>
156 )
157 }
158
159 export default AddTransaction;

```

Kita mulai membahas dari kode JSX terlebih dahulu. Sama seperti komponen lain, sebagian besar kode yang ada terdiri dari tag HTML dengan berbagai class CSS bawaan Bootstrap.

Di baris 111 terdapat perintah *short-circuit conditionals* `!isValid.current &&`. Dari kode ini bisa di tebak kalau `isValid` adalah sebuah *ref* yang nilainya diakses dari property `current`. Ini adalah *flag* atau penanda yang saya pakai untuk menampilkan box pesan error di baris 112 – 120. Jika `isValid.current` bernilai `false`, box pesan error akan tampil (perhatikan tambahan operator negasi " ! ").

Setiap pesan error juga ditulis dalam bentuk *short-circuit conditionals* di baris 115, 116, dan 117. Jika pesan error berisi sesuatu, tampilkan sebagai *unordered list* menggunakan tag ``.

Kemudian di baris 121 terdapat tag `<form>` sebagai penanda dimulainya bagian form. Pada saat form di submit, pemrosesan akan di *handle* oleh fungsi `handleSubmit()` sesuai nilai atribut `onSubmit`.

Form ini terdiri dari 3 inputan: tanggal, keterangan dan nominal. Setiap tag `<input>` memiliki atribut `value` dan `onChange` sebagai syarat dari *controlled component* React.

Sedikit tambahan ada di penulisan atribut `className`. Sebagai contoh, di baris 127 terdapat perintah berikut:

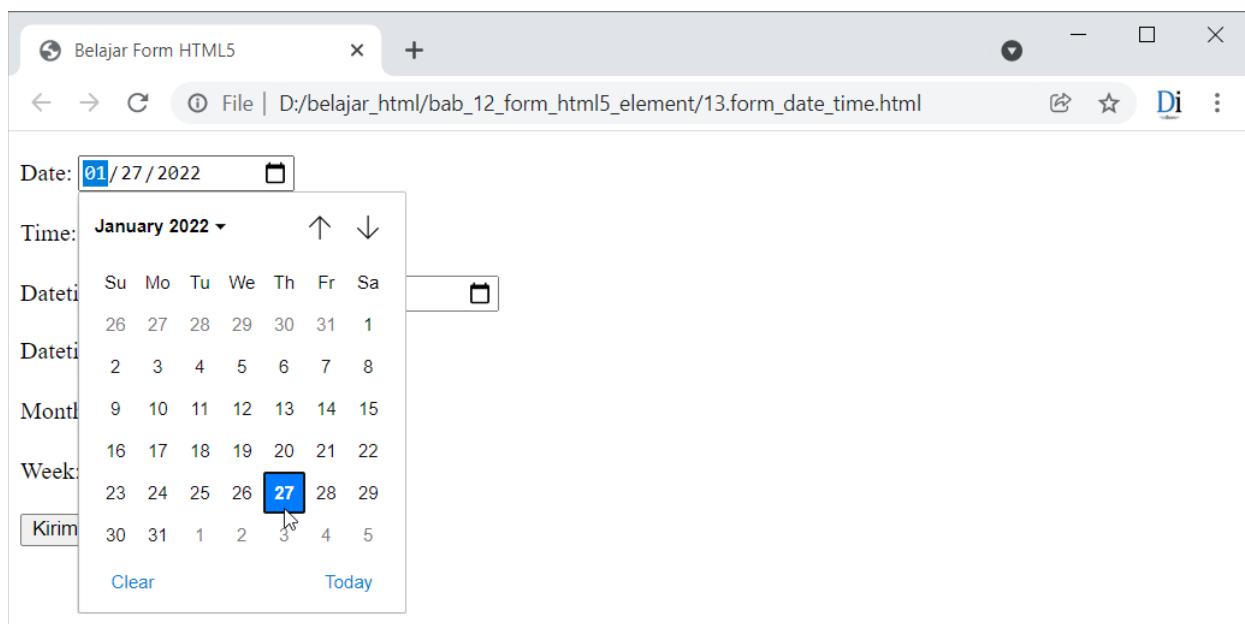
```
className={`form-control ${errors.tanggal && "is-invalid"}`}
```

Tag `<input>` ini akan memiliki tambahan class `.is-invalid` jika pesan error untuk inputan

tersebut berisi sesuatu. Dalam Bootstrap, class `.is-invalid` dipakai untuk membuat efek border berwarna merah ke dalam tag `<input>`. Ketiga inputan form memiliki tambahan kode ini seperti di baris 135 dan juga 143.

Masuk ke kode React, di awal komponen `AddTransaction` terdapat pendefinisian fungsi `getTanggal()`. Fungsi ini berguna untuk men-generate tanggal yang nantinya dipakai sebagai nilai awal dari inputan tanggal. Perintah `new Date()` di baris 7 akan mengambil tanggal di komputer user saat ini, kemudian disimpan ke dalam variabel `dateObj` sebagai `Date` object JavaScript.

Proses input tanggal ke dalam sebuah form memang susah-susah gampang. "Susah" karena inputan tanggal hadir dengan berbagai pilihan. Salah satunya bisa menggunakan tag `<input type="date">` bawaan HTML5. Jika kita memakai tag ini, web browser langsung menyajikan jendela popup "date picker" untuk pemilihan tanggal seperti tampilan berikut:



Gambar: Jendela popup date picker dari tag `<input type="date">` bawaan HTML 5

Meskipun praktis, tapi format yang dipakai bisa berbeda tergantung pengaturan web browser. Jika web browser di set dengan bahasa Inggris, format tanggal yang dipakai adalah `MM/DD/YYYY`, dimana `01/12/2022` merujuk ke 12 Januari 2022. Di Indonesia, kita menggunakan format `DD/MM/YYYY`, sehingga tanggal `01/12/2022` merujuk ke 01 Desember 2022.

Atas dasar masalah ini, saya putuskan untuk tidak memakai tag `<input type="date">`, tapi cukup tag `<input type="text">` biasa. Hanya saja tanggal harus diinput dalam format `DD/MM/YYYY`:

Tanggal	Keterangan
12/01/2022	Bayar Cicilan

Gambar: Inputan tanggal dengan format DD/MM/YYYY

Alternatif lain bisa menggunakan library JS khusus untuk inputan tanggal seperti [date picker](#) bawaan jQuery UI. Namun agar kodnya tidak terlalu rumit, saya akan memakai inputan manual saja.

UI/UX Untuk Inputan Tanggal

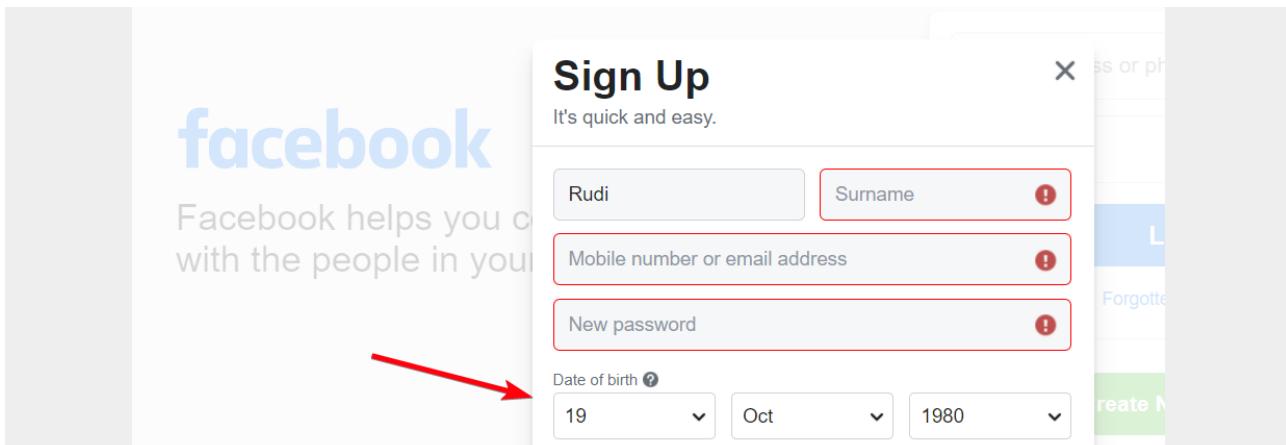
Inputan tanggal dalam sebuah form kadang bisa merepotkan jika "terlalu canggih". Salah satu yang sering membuat ribet adalah memasang date picker di inputan tanggal lahir.

Secara default date picker selalu merujuk ke tanggal hari ini, sehingga jika ada user yang lahir di tahun 19 Oktober 1980, sangat tidak praktis mencari tanggal menggunakan date picker:

The screenshot shows a web form titled "Pendaftaran Pasien Online". It includes fields for NIK (NIK), Name (Nama Lengkap), and Birthplace/Date of Birth (Tempat / Tanggal Lahir). The birthdate field has a date picker calendar open, showing November 2020. A red arrow points from the text below to the date picker's input field, which is highlighted with a green border and contains the placeholder "Tanggal Lahir Tidak Boleh Kosong".

Gambar: Repotnya mengisi tanggal lahir dengan date picker

Kadang menggunakan inputan yang sederhana malah lebih praktis. Misalnya inputan tanggal lahir saat mendaftar Facebook tetap memakai tag <select> biasa:



Gambar: Inputan tanggal lahir saat pendaftaran Facebook

Hal seperti ini memang sering terabaikan, namun tampilan web yang cantik kadang bisa kalah dengan tampilan web sederhana tapi menyajikan user experience yang lebih baik.

Kembali ke fungsi `getTanggal()`, kode program antara baris 10-15 dipakai untuk membuat format DD/MM/YYYY berdasarkan Date object JavaScript.

Untuk membuat tanggal (hari) dalam format 2 digit angka, saya perlu tambahan method `padStart(2, "0")` dari hasil `dateObj.getDate().toString()`. Begitu juga untuk mengakses bulan dalam format 2 digit, perlu menambah hasil `dateObj.getMonth()` dengan angka 1 karena method `getMonth()` JavaScript menghasilkan angka bulan mulai dari 0, bukan 1. Keduanya kemudian disimpan ke dalam variabel `tanggal` di baris 10 dan `bulan` di baris 13.

Di baris 15, string akhir didapat dari gabungan variabel `tanggal`, `bulan` dan `dateObj.getFullYear()`. Pada saat di panggil, fungsi `getTanggal()` akan mengembalikan string tanggal dalam format DD/MM/YYYY. Sebagai contoh jika saat ini tanggal 27 November 2021, maka hasil dari `console.log(getTanggal())` adalah 27/11/2021.

Lanjut di baris 18, terdapat kode untuk mendefinisikan state `formInput`. Seperti biasa, state ini berbentuk *nested object* untuk menyimpan semua nilai inputan form. Untuk tanggal, nilai awal diisi dari hasil pemanggilan fungsi `getTanggal()` seperti di baris 19.

Di baris 24, giliran state `errors` di definisikan untuk menampung semua pesan error.

Di baris 30 saya membuat sebuah *ref* bernama `formValid` dengan nilai awal `true`. Ref ini dipakai sebagai penanda atau *flag* apakah form sudah lolos validasi atau tidak. Nantinya, `formValid` berguna untuk menampilkan pesan error validasi di dalam kode JSX.

Di baris 32 terdapat fungsi `handleInputChange()` sebagai *event handler* untuk semua inputan form. Fungsi ini juga sudah berkali-kali kita pakai.

Antara baris 36-103 terdapat fungsi `handleFormSubmit()` untuk memproses nilai inputan form. Teknik yang dipakai juga sama seperti project-project sebelumnya, dimulai dari proses validasi inputan form antara baris 41-67.

Semua inputan form tidak boleh kosong, sehingga ada pemeriksaan `trim() === ""`. Tambahan validasi ada di baris 44 dan 62 berbentuk *regular expression* untuk memeriksa apakah format inputan sudah sesuai atau belum.

Format *regexp* `^[0-3][0-9]\/[0-1][0-9]\/[0-9]{4}$` di baris 44 hanya akan cocok jika user menginput tanggal dalam format DD/MM/YYYY. Jika diperhatikan lebih lanjut, format ini sebenarnya masih punya kelemahan karena tanggal 30/2/2021 akan tetap lolos, padahal bulan Februari hanya bisa sampai 29.

Untuk aplikasi sederhana seperti contoh kita, saya rasa tidak terlalu masalah. Akan tetapi jika anda merancang aplikasi yang sensitif dengan tanggal, ada baiknya membuat kode program yang lebih kompleks untuk mengantisipasi kesalahan logika bulan Februari ini.

Regular expression `^[\+|\-]\d+$` di baris 62 dipakai untuk memastikan inputan nominal hanya bisa berbentuk angka dengan awalan plus "+" atau minus "-". Jika inputan user tidak memenuhi syarat ini, maka tampilkan pesan error.

Setelah proses validasi, `ref formValid.current` di update pada baris 76 tergantung apakah masih ada pesan error atau tidak. Jika ternyata terdapat error, kode program akan berhenti sampai di baris 78.

Blok kode program antara baris 83-102 hanya dijalankan jika kondisi `if(formValid.current)` bernilai `true`, yakni jika semua inputan user sudah lolos validasi.

Inputan tanggal perlu di proses lagi agar bisa menjadi angka `timestamp`. Caranya, potong string tanggal dari format DD/MM/YYYY menjadi DD, MM dan YYYY, lalu input ke dalam **Date** object JavaScript.

Method `substr()` dipakai untuk memotong string tanggal, lalu dikonversi menjadi integer dengan fungsi `parseInt()`. Selanjutnya method `setDate()`, `setMonth()` dan `setFullYear()` akan memodifikasi nilai object `tanggalInput` sesuai tanggal inputan user.

Setelah itu di baris 88-93 semua nilai form disatukan menjadi 1 object transaction lengkap.

Property `id` di generate dengan angka random hasil perhitungan `Math.random() * 1000000000000`. Ini hanya cara "darurat" karena idealnya angka `id` digenerate menggunakan kolom `auto increment` dari database atau menggunakan library `UUID` (Universally Unique Identifier).

Property `tanggal` diisi dengan nilai `timestamp` yang didapat dari hasil perintah `tanggalInput.getTime()`.

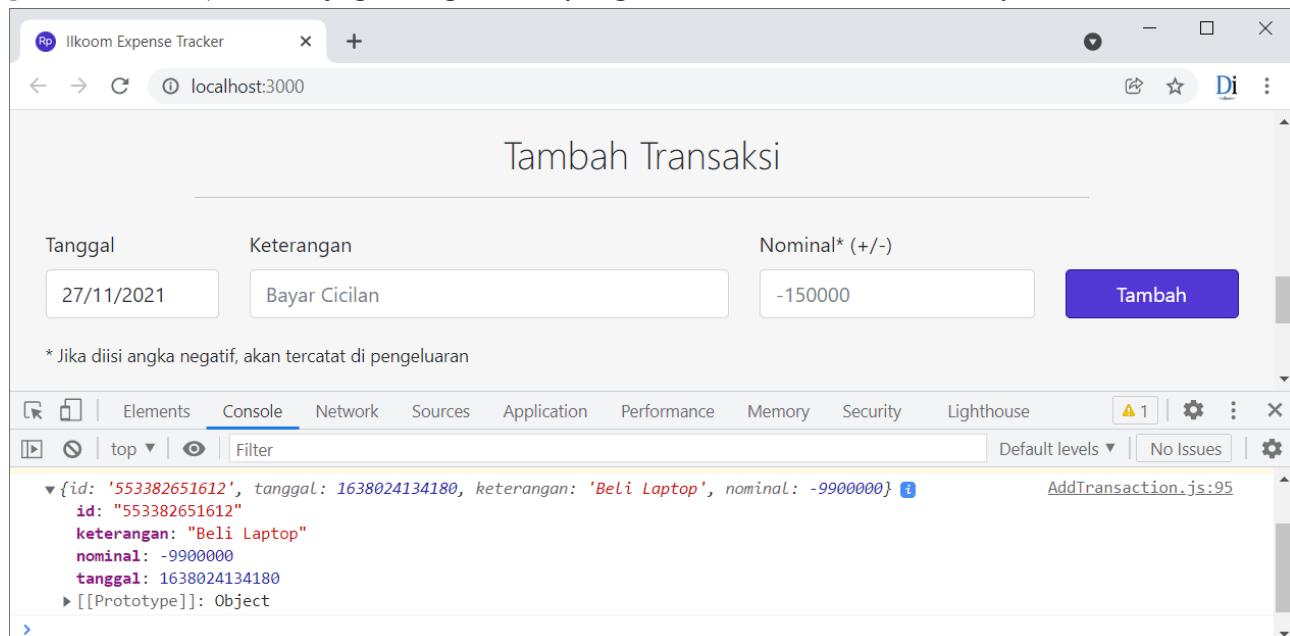
Untuk property `keterangan` dan `nominal`, bisa langsung diisi dengan data inputan user. Khusus untuk `nominal`, perlu tambahan fungsi `parseInt()` agar tipe data string yang berasal dari form dikonversi menjadi tipe data integer.

Isi variabel `transaction` kemudian di tampilkan dengan perintah `console.log(transaction)` di baris 95. Nantinya perintah ini akan kita ganti dengan `props` untuk mengupdate state

transaction yang dikirim dari komponen App.

Terakhir, kode program di baris 98-102 dipakai untuk mengosongkan inputan form dengan cara mengisi string kosong ke dalam state `formInput`.

Silahkan tes komponen `AddTransaction` dengan mengisi nilai yang salah (untuk menguji proses validasi), dan isi juga dengan nilai yang benar untuk melihat hasilnya di tab console.



Gambar: Data transaksi sukses tampil di tab Console

Dalam aplikasi ini, transaksi pengeluaran harus diinput dengan nominal negatif (diawali dengan tanda minus seperti `-9900000`). Ini trik yang saya pakai untuk menyederhanakan inputan form. Alternatif lain, bisa juga buat sebuah tombol atau checkbox untuk membedakan apakah transaksi yang akan di input sebuah pemasukan, atau pengeluaran.

17.6. Menambah Transaction

Konsep "*single source of truth*" dari React hanya mengizinkan state utama disimpan di satu tempat. Dalam aplikasi kita, state `transaction` hanya ada di dalam komponen App. Inilah state utama yang berisi semua data transaksi.

Karena proses input data transaksi ada di dalam komponen `AddTransaction`, kita perlu mengirim sebuah function dari komponen App untuk diakses dari `AddTransaction`. Silahkan buka file `App.js`, lalu tambah function berikut:

```
src\App.js
1 ...
2 const App = () => {
```

```

3   const [transactions, setTransaction] = useState(initTransactions);
4
5   // handler untuk menambah data transaction,
6   // akan di-trigger dari komponen AddTransaction
7   const handleTambahTransaction = (data) => {
8     let newTransactions = [
9       ...transactions, data
10    ];
11
12   // atur ulang urutan transaction agar tanggal terkecil di bagian atas
13   newTransactions.sort((a, b) => a.tanggal - b.tanggal);
14
15   setTransaction(newTransactions);
16 }
17
18 return (
19   <React.Fragment>
20     <Header />
21     <SaldoBox transactions={transactions} />
22     <Transaction transactions={transactions} />
23     <AddTransaction onTambahTransaction={handleTambahTransaction} />
24     <Footer />
25   </React.Fragment>
26 )
27 }
```

Fungsi `handleTambahTransaction()` menerima 1 parameter `data`. Nantinya parameter ini akan berisi 1 object `transaction` lengkap yang terdiri dari property `id`, `tanggal`, `keterangan` dan `nominal`.

Di baris 8, variabel `newTransactions` diisi semua state `transaction` saat ini, ditambah dengan yang tersimpan di parameter `data`. Proses ini dilakukan dengan bantuan spread operator.

Selanjutnya di baris 13 semua object yang ada di dalam variabel `newTransactions` saya susun ulang agar transaksi dengan tanggal paling lama berada di posisi pertama dan berurutan ke tanggal paling baru.

Proses ini dilakukan dengan bantuan method `sort()` bawaan JavaScript. Method `sort()` butuh sebuah argument dalam bentuk `function callback` untuk menentukan syarat urutan. Jika nantinya data transaksi kita simpan ke dalam database, proses pengurutan data bisa dilakukan dengan perintah query `SORT` milik database.

Setelah daftar transaksi di susun ulang, isi variabel `newTransactions` selanjutnya di update ke dalam state `transaction` dengan perintah `setTransaction(newTransactions)` di baris 15.

Tidak lupa, fungsi `handleTambahTransaction` kita kirim sebagai `props` ke dalam tag `<AddTransaction />` di baris 23. Sesampainya di dalam komponen `AddTransaction`, fungsi `handleTambahTransaction()` bisa diakses sebagai `props.onTambahTransaction()`.

Silahkan buka file `AddTransaction.js`, lalu modifikasi kode program di akhir fungsi `handleFormSubmit()` dari yang sebelumnya:

src\components\AddTransaction.js

```
1 ...  
2     if (formValid.current) {  
3  
4         let tanggalInput = new Date();  
5         tanggalInput.setDate(parseInt(formInput.tanggal.substr(0, 2)));  
6         tanggalInput.setMonth(parseInt(formInput.tanggal.substr(3, 2) - 1));  
7         tanggalInput.setFullYear(parseInt(formInput.tanggal.substr(6, 4)));  
8  
9         let transaction = {  
10             "id": Math.floor(Math.random() * 100000000000).toString(),  
11             "tanggal": tanggalInput.getTime(),  
12             "keterangan": formInput.keterangan,  
13             "nominal": parseInt(formInput.nominal),  
14         };  
15  
16         console.log(transaction);  
17  
18         // kosongkan inputan form  
19         setFormInput({  
20             tanggal: getTanggal(),  
21             keterangan: "",  
22             nominal: ""  
23         })  
24     }  
25 ...
```

Menjadi:

src\components\AddTransaction.js

```
1 ...  
2     if (formValid.current) {  
3  
4         let tanggalInput = new Date();  
5         tanggalInput.setDate(parseInt(formInput.tanggal.substr(0, 2)));  
6         tanggalInput.setMonth(parseInt(formInput.tanggal.substr(3, 2) - 1));  
7         tanggalInput.setFullYear(parseInt(formInput.tanggal.substr(6, 4)));  
8  
9         props.onTambahTransaction({  
10             "id": Math.floor(Math.random() * 100000000000).toString(),  
11             "tanggal": tanggalInput.getTime(),  
12             "keterangan": formInput.keterangan,  
13             "nominal": parseInt(formInput.nominal),  
14         });  
15  
16         // kosongkan inputan form  
17         setFormInput({  
18             tanggal: getTanggal(),  
19             keterangan: "",  
20             nominal: ""  
21         })  
22     }  
23 ...
```

Kali ini object transaction langsung di input sebagai argument ke dalam fungsi `props.onTambahTransaction()` di baris 9-14. Argument ini akan ditampung oleh parameter data di dalam fungsi `handleTambahTransaction()` milik komponen App.

Silahkan tes hasilnya di web browser:

The screenshot shows the Ilkoom Expense Tracker application interface. At the top, it says "Ilkoom Expense Tracker" and "Catat setiap pemasukan dan pengeluaranmu". Below this is a "Saldo" box showing "Rp. -6.870.000". Underneath are two buttons: "Rp. 3.250.000" (green) and "Rp. -10.120.000" (red). The main area is divided into "Pemasukan" and "Pengeluaran". The "Pemasukan" section lists "Gaji bulanan" (Rp. 2.500.000) and "Uang lembur" (Rp. 750.000). The "Pengeluaran" section lists "Beli sepatu" (Rp. -150.000), "Beli Laptop" (Rp. -9.900.000), and "Beli buku React Uncover" (Rp. -70.000). At the bottom, there's a "Tambah Transaksi" form with fields for "Tanggal" (28/11/2021), "Keterangan" (Bayar Cicilan), "Nominal* (+/-)" (-150000), and a "Tambah" button. A note below the form says: "* Jika diisi angka negatif, akan tercatat di pengeluaran".

Gambar: Proses penambahan transaksi berhasil

Sip, proses menambah transaksi sudah berhasil. Jika nominal yang diinput diawali tanda negatif " - ", maka transaksi itu akan masuk ke kelompok pengeluaran. Perhatikan juga total saldo yang berasal dari komponen `SaldoBox` otomatis update pada saat terdapat transaksi baru.

17.7. Menghapus Transaction

Tombol untuk menghapus transaction sebenarnya sudah kita siapkan di dalam komponen `ListTransaction`, yakni icon "x" di sudut kanan atas setiap list transaksi. Yang diperlukan saat ini adalah mengirim fungsi penghapusan transaksi dari komponen `App` ke dalam komponen `ListTransaction`.

Pertama, silahkan modifikasi file `App.js` dengan tambahan fungsi berikut:

src\App.js

```
1  ...
2  const App = () => {
3      const [transactions, setTransaction] = useState(initTransactions);
4
5      ...
6      // handler untuk menghapus data transaction di komponen AddTransaction
7      const handleHapusTransaction = (e) => {
8
9          // cari index transaction yang akan dihapus berdasarkan id
10         const result = transactions.findIndex(
11             transaction => (transaction.id === e.target.id)
12         );
13         // copy transactions karena fungsi splice akan mengubah array asal (mutate)
14         const newTransactions = transactions;
15         newTransactions.splice(result, 1);
16         setTransaction([...newTransactions]);
17     }
18
19     return (
20         <React.Fragment>
21             <Header />
22             <SaldoBox transactions={transactions} />
23             <Transaction transactions={transactions}
24                 onHapusTransaction={handleHapusTransaction} />
25             <AddTransaction onTambahTransaction={handleTambahTransaction} />
26             <Footer />
27         </React.Fragment>
28     )
29
30 }
```

Proses penghapusan data menggunakan teknik yang sama seperti di mini project CRUD Mahasiswa, dimana fungsi `handleHapusTransaction()` akan menerima sebuah parameter `e` yang nantinya diisi dengan `event object`.

Konstanta `result` di baris 10 berguna untuk menampung index array dari object transaksi yang akan dihapus. Proses pencarian index menggunakan method `findIndex()` bawaan JavaScript. Setelah indexnya di dapat, object transaksi dihapus menggunakan method `splice(result, 1)`. Penjelasan lebih lanjut tentang cara ini sudah kita bahas di mini project CRUD Mahasiswa.

Fungsi `handleHapusTransaction()` selanjutnya dikirim sebagai `props` ke dalam komponen `<Transaction />` di baris 23.

Loh, bukannya tombol "x" ada di komponen `ListTransaction`? Betul, akan tetapi komponen `ListTransaction` merupakan child dari komponen `Transaction`, sehingga kita harus "lewatkan" sebagai `props` ke dalam komponen `Transaction` terlebih dahulu.

Berikutnya, buka file `Transaction.js`, lalu modifikasi dengan tambahan kode program berikut:

src\components\Transaction.js

```
1 import ListTransaction from './ListTransaction';
2
3 const Transaction = (props) => {
4     return (
5         <section id="transaction">
6             <div className="container py-5">
7                 <div className="row">
8
9                     <div className="col-12 col-lg-6 mb-4" id="pemasukan">
10                         <h2 className="fw-light mb-3 text-center">Pemasukan</h2>
11                         <hr className="w-75 mx-auto mb-4" />
12                         {
13                             props.transactions.map((transaction) => (
14                                 (transaction.nominal > 0) &&
15                                 <ListTransaction
16                                     key={transaction.id}
17                                     id={transaction.id}
18                                     tanggal={transaction.tanggal}
19                                     keterangan={transaction.keterangan}
20                                     nominal={transaction.nominal}
21                                     onHapusTransaction={props.onHapusTransaction}
22                                 />
23                             ))
24                         }
25                     </div>
26
27                     <div className="col-12 col-lg-6 mb-4" id="pengeluaran">
28                         <h2 className="fw-light text-center mb-3">Pengeluaran</h2>
29                         <hr className="w-75 mx-auto mb-4" />
30                         {
31                             props.transactions.map((transaction) => (
32                                 (transaction.nominal <= 0) &&
33                                 <ListTransaction
34                                     key={transaction.id}
35                                     id={transaction.id}
36                                     tanggal={transaction.tanggal}
37                                     keterangan={transaction.keterangan}
38                                     nominal={transaction.nominal}
39                                     onHapusTransaction={props.onHapusTransaction}
40                                 />
41                             ))
42                         }
43                     </div>
44
45                 </div>
46             </div>
47         </section>
48     )
49 }
50
51 export default Transaction;
```

Tambahan kode program ada di baris 21 dan 39, yakni meneruskan props.onHapusTransaction

ke komponen `ListTransaction`.

Lanjut, buka file `ListTransaction.js` dan modifikasi sebagai berikut:

`src\components\ListTransaction.js`

```
1 const ListTransaction = (props) => {
2 ...
3   return (
4     <div className={`list-transaction p-1 mb-2 ${classTransaction}`}>
5       <div className="ms-2 d-flex flex-column">
6         <p className="m-0 fs-5">{props.keterangan}</p>
7         <small>{getTanggal(props.tanggal)}</small>
8       </div>
9       <p className="fs-5 mb-1 me-3 text">
10        Rp. {props.nominal.toLocaleString('id-ID')}
11      </p>
12      <span className="delete-icon" id={props.id}
13        onClick={props.onHapusTransaction}>x</span>
14    </div>
15  )
16 }
17
18 export default ListTransaction;
```

Yang perlu dimodifikasi hanya tag `` di baris 12-13, berupa tambahan atribut `id={props.id}` serta `onClick={props.onHapusTransaction}`. Maka ketika tombol " x " di klik, event object dari tag `` akan ditampung oleh fungsi `handleHapusTransaction()` di komponen `App` sebagai parameter `e`. Dari parameter `e`, bisa diakses `e.target.id` yang akan berisi nomor `id` dari transaksi yang sedang dihapus.

Silahkan tes klik tombol " x " untuk menghapus transaksi:



Gambar: Menghapus transaksi dengan klik tombol " x "

Dengan tambahan fitur hapus transaksi, maka aplikasi **Ilkoom Expense Tracker** juga sudah selesai. Di antara semua metode CRUD, saya tidak membuat fitur update transaksi untuk

menyederhanakan kode program. Jika terjadi kesalahan input, user terpaksa harus menghapus transaksi tersebut lalu menginput kembali transaksi yang benar.

Untuk aplikasi dengan 3 inputan (tanggal, keterangan dan nominal), saya rasa ini tidak terlalu masalah. Namun juga tidak ada salahnya jika anda ingin menambah fitur Edit transaksi sebagai latihan tambahan.

Mini project Ilkoom Expense Tracker ini menerapkan cukup banyak konsep dasar React. Dengan materi yang ada, silahkan coba buat aplikasi sejenis untuk keperluan lain seperti aplikasi pencatat belanjaan, wishlist game di steam, dsb.

Dalam bab setelah ini kita akan lanjut membahas tentang fungsi `fetch()` JavaScript untuk mengakses data API.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

18. JavaScript Fetch API

Kode program React sering dipakai untuk memproses dan menampilkan data API. Dalam bab ini kita akan bahas bagaimana caranya, yang dimulai dari membahas fungsi `fetch()` bawaan JavaScript.

18.1. Pengertian API

Sebelum kita sampai ke pengertian fetch API, saya ingin membahas sekilas tentang API terlebih dahulu. Dalam programming, **API** (singkatan dari Application Programming Interface), adalah sebuah sarana komunikasi dari suatu aplikasi.

Sebagai analogi, remote TV bisa disebut sebagai API karena menjadi sarana komunikasi untuk mengakses semua fitur yang ada di sebuah TV. Begitu juga di programming, API berguna agar aplikasi lain bisa mengakses fitur-fitur yang ada.

Dengan memakai API, kita bisa mengakses informasi dari sebuah aplikasi tanpa harus paham bagaimana informasi itu di proses. Sama seperti remote TV, ketika tombol channel 7 di tekan, TV segera berganti ke channel tersebut. Kita tidak perlu paham bagaimana cara kerjanya.

Ketika ingin membuat aplikasi desktop, sistem operasi Windows atau Linux juga menyediakan banyak API sebagai sarana bagi kita untuk mengakses fitur dari OS tersebut. Misalnya ketika tombol di klik besarkan volume, atau ganti warna background.

Bagi yang pernah belajar bahasa pemrograman PHP, fungsi `mysqli_connect()` juga sebuah API, karena dipakai untuk mengakses database MySQL yang merupakan aplikasi terpisah dari PHP.

Di lingkungan website, API biasa dipakai sebagai sarana komunikasi antar website, atau juga di internal website itu sendiri.

18.2. Pengertian REST API

Sepanjang kita membuat kode program dengan React, API memiliki peranan yang sangat penting. React merupakan framework/library JavaScript yang posisinya ada di front-end. Ketika ingin menyimpan data web secara permanen (misalnya hasil dari inputan form), itu tidak bisa diproses oleh React saja.

Data tersebut harus dikirim ke back-end untuk di proses oleh bahasa pemrograman server seperti PHP atau nodeJS untuk kemudian disimpan ke dalam aplikasi database seperti MySQL.

Komunikasi antara front-end dan back-end ini salah satunya bisa dilakukan dengan API.

Dari beberapa jenis arsitektur API, **REST API** menjadi pilihan yang paling banyak dipakai.

REpresentational State Transfer API, atau disingkat sebagai REST API adalah teknik mengirim dan menerima data menggunakan 4 HTTP *verb* atau *method*, yakni: **get**, **post**, **put/patch**, dan **delete**. Meskipun ada beberapa method lain, tapi 4 inilah yang sering dipakai.

Keempat HTTP verb dari REST API mewakili cara akses CRUD dengan rincian sebagai berikut:

- Method **GET**: dipakai untuk mengakses data (**Read**)
- Method **POST**: dipakai untuk menyimpan data baru (**Create**)
- Method **PUT/PATCH**: dipakai untuk mengubah data yang sudah ada (**Update**)
- Method **DELETE**: dipakai untuk menghapus data (**Delete**)

Jika anda baru pertama kali berkenalan dengan REST API, tidak masalah jika masih bingung dengan penjelasan diatas. Mudah-mudahan nanti akan lebih paham ketika kita masuk ke praktek sesaat lagi.

18.3. Pengertian JSON

JSON merupakan singkatan dari **JavaScript Object Notation**, yakni sebuah cara penulisan string yang menyerupai struktur object di dalam JavaScript. Ketika kita mengakses API dari suatu website, mayoritas akan mengirim data dalam bentuk JSON.

Sebagai contoh, di dalam JavaScript kita bisa membuat object `mahasiswa` yang memiliki 4 property dengan kode berikut:

```

1 <script>
2 {
3     nim: "18010245",
4     nama: "Eka Putra",
5     jurusan: "Teknik Informatika",
6     asalProvinsi: "DKI Jakarta"
7 }
8 </script>
```

Dalam format JSON, data yang sama bisa ditulis sebagai berikut:

```

1 {
2     "nim": "18010245",
3     "nama": "Eka Putra",
4     "jurusan": "Teknik Informatika",
5     "asalProvinsi": "DKI Jakarta"
6 }
```

Cara penulisannya sangat mirip. Perbedaan hanya di penulisan nama property, yang untuk JSON harus berbentuk string di antara tanda kutip.

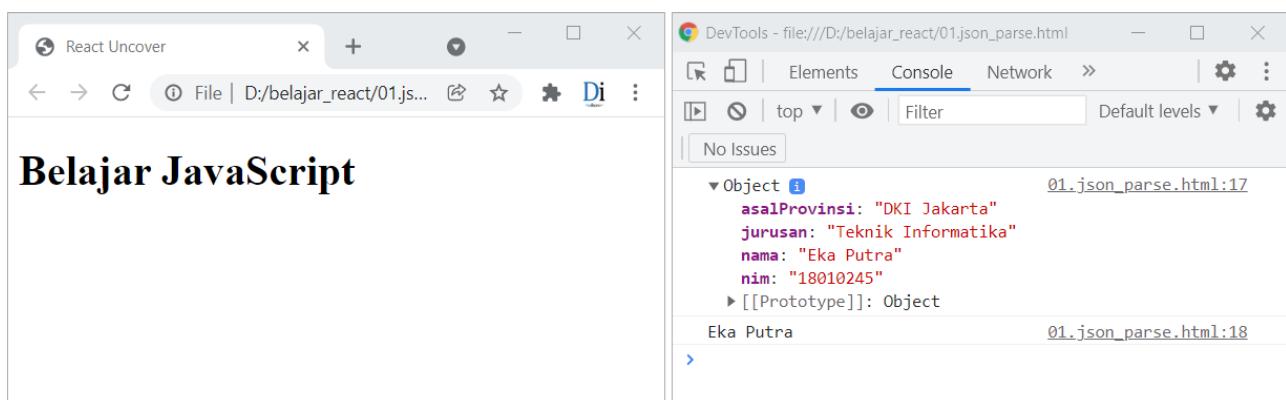
Hampir semua bahasa pemrograman menyediakan cara untuk mengkonversi JSON menjadi object dan sebaliknya, termasuk JavaScript. Kita bisa menggunakan method `JSON.parse()` untuk proses konversi JSON menjadi object, dan method `JSON.stringify()` untuk konversi dari object JavaScript ke dalam bentuk JSON.

Pertama, kita lihat praktik dari `JSON.parse()` terlebih dahulu:

01.json_parse.html

```

1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8  </head>
9
10 <body>
11     <h1>Belajar JavaScript</h1>
12     <script>
13         let mahasiswaJSON = `{"nim":"18010245","nama":"Eka Putra",
14             "jurusan":"Teknik Informatika","asalProvinsi":"DKI Jakarta"}`;
15
16         let mahasiswa = JSON.parse(mahasiswaJSON);
17         console.log(mahasiswa);
18         console.log(mahasiswa.nama);
19     </script>
20 </body>
21
22 </html>
```



Gambar: Proses konversi JSON menjadi JavaScript object

Di baris 13, variabel `mahasiswaJSON` saya isi dengan string dalam bentuk JSON. Kemudian di baris 16 variabel `mahasiswa` akan menampung hasil dari perintah `JSON.parse(mahasiswaJSON)`.

Hasilnya, variabel `mahasiswa` sudah berisi JavaScript object. Termasuk kita bisa mengakses property `nama` dengan perintah `mahasiswa.nama` seperti di baris 18.

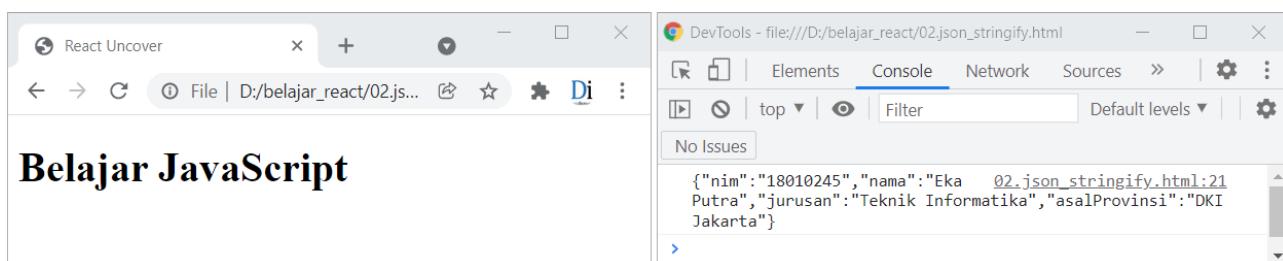
Sebaliknya, method `JSON.stringify()` bisa dipakai untuk proses konversi dari JavaScript

object menjadi JSON:

02.json_stringify.html

```

1 <script>
2   let mahasiswa = {
3     nim: "18010245",
4     nama: "Eka Putra",
5     jurusan: "Teknik Informatika",
6     asalProvinsi: "DKI Jakarta"
7   }
8
9   let mahasiswaJSON = JSON.stringify(mahasiswa);
10  console.log(mahasiswaJSON);
11 </script>
```



Gambar: Proses konversi JavaScript object menjadi JSON

Sekarang di baris 2 saya mengisi variabel mahasiswa dengan object, kemudian perintah `JSON.stringify(mahasiswa)` akan mengkonversi object tadi ke dalam bentuk string dengan format JSON.

Dulunya, data yang dikirim lewat API ada yang berbentuk **XML** (Extensible Markup Language) yakni format penulisan yang terdiri dari tag-tag seperti HTML. Akan tetapi XML sudah banyak ditinggalkan karena JSON dianggap lebih praktis dan lebih singkat sebagai sarana komunikasi data API.

18.4. Pengertian Fetch API

Sejauh ini kita sudah membahas mengenai API, REST API dan juga JSON. Sekarang saatnya masuk ke Fetch API.

Fetch API adalah kumpulan function JavaScript untuk memproses data API. Istilah "memproses data" kadang disebut juga sebagai "mengonsumsi API" (consume an API), karena data JSON yang kita dapat akan "dimakan" terlebih dahulu oleh Fetch API agar bisa diakses sebagai data biasa.

Fitur fetch API sendiri masih relatif baru di JavaScript. Sebelumnya, proses akses data dari API dilakukan dengan **XMLHttpRequest** object (disingkat sebagai XHR). **XHR** menjadi fitur dasar

dari teknologi yang dikenal sebagai **AJAX** (Asynchronous Javascript and XML), yakni cara yang memungkinkan JavaScript bisa berkomunikasi dengan server secara *asynchronous*.

Akan tetapi membuat AJAX dengan XHR object memang agak ribet, karena itulah mayoritas kode program AJAX ditulis dengan bantuan library JavaScript: **jQuery**.

Sekitar tahun 2017, tim pengembang JavaScript menambah fitur `fetch` API agar programmer JavaScript tidak terus bergantung ke jQuery. Jadi, `fetch` API di sini menjadi alternatif pengganti dari XMLHttpRequest object yang lebih praktis.

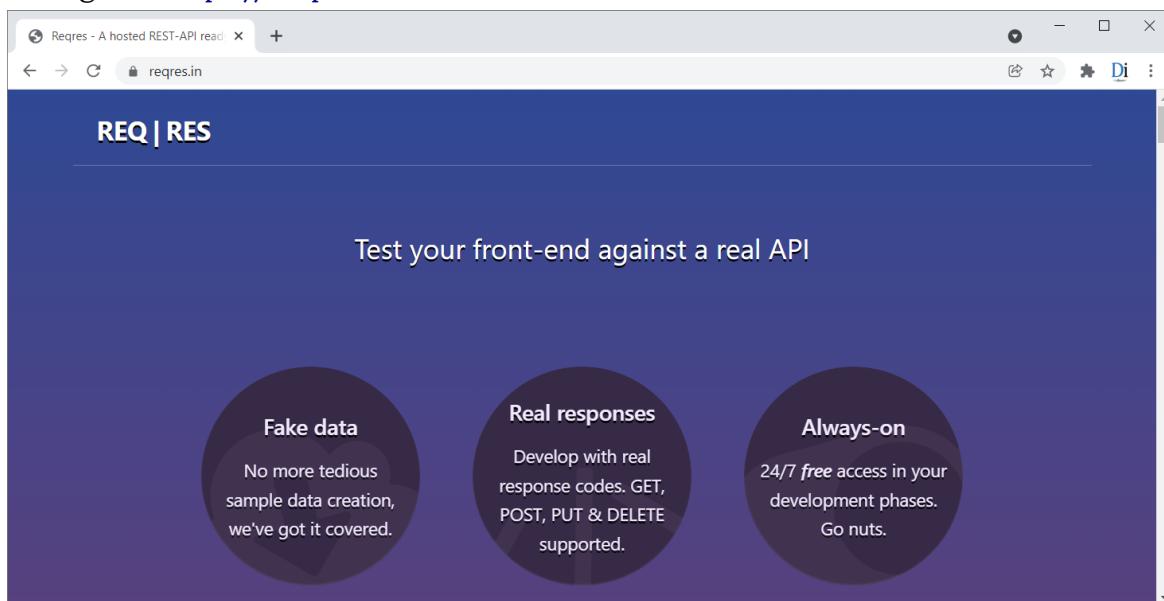
Hadirnya `Fetch` API tidak menghentikan pengembangan library API JavaScript lain. Salah satu yang cukup terkenal adalah [Axios](#)²⁴. Beberapa programmer lebih menyukai Axios daripada `Fetch` API, akan tetapi di buku ini kita hanya akan membahas `fetch` API saja.

18.5. Cara Penggunaan Fetch API

Untuk bisa melakukan praktek dengan `Fetch` API, kita harus mengakses web server yang menyediakan data API. Untungnya, di internet cukup banyak layanan API gratis, sehingga kita tidak perlu membuat manual server API.

Mengakses API reqres.in

Layanan API di internet ada yang khusus dirancang untuk proses testing dan bisa langsung dipakai (tidak perlu register terlebih dahulu). Dalam praktek ini saya akan memakai layanan API testing dari <https://reqres.in>²⁵.



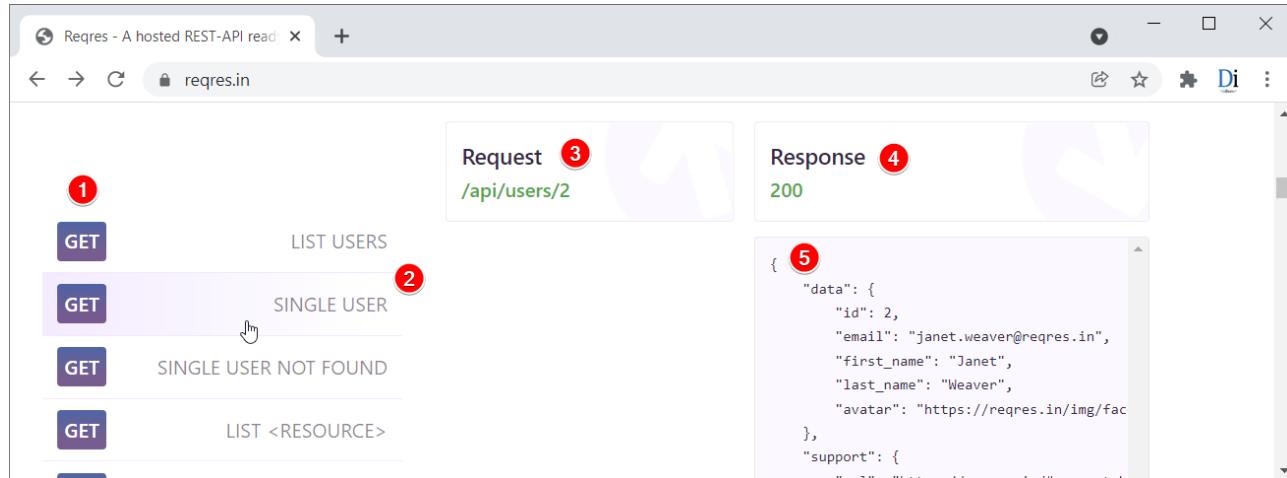
Gambar: Tampilan halaman awal reqres.in

24. <https://axios-http.com/>

25. <https://reqres.in>

Ketika ingin mengakses API, hal pertama yang harus kita pelajari adalah apa alamat URL yang harus diakses. Meskipun terdapat pola dasar, setiap web bebas menentukan alamat apa yang harus diakses untuk setiap API request.

Web reqres.in menyediakan data API untuk data **users**, yakni user fiktif yang terdaftar di database web server tersebut. Silahkan scroll halaman reqres.in agak ke bawah, akan terlihat daftar URL API serta bentuk respon JSON-nya:



Gambar: Daftar alamat URL untuk mengakses API web reqres.in

Di sisi kiri (1) terdapat HTTP method yang harus dipakai, apakah itu `get`, `post`, `put`, `patch` dan `delete`. Untuk API request yang menampilkan data, hampir selalu memakai method `get`.

Di sebelah HTTP method, terdapat penjelasan singkat mengenai fungsi dari setiap API (2), apakah kita ingin mengakses list user, single user, simulasi dari user not found, dsb.

Sebagai contoh, jika "SINGLE USER" di klik, akan tampil alamat URL API dibagian request (3), yang dalam contoh ini adalah `/api/users/2`. Artinya jika kita ingin mengakses API untuk 1 user saja, alamat yang diakses harus ditambah dengan `/api/users/2`. Angka 2 merujuk ke nomor id user dan boleh diganti dengan angka lain.

Pada saat kotak request di klik, akan terbuka tab baru di <https://reqres.in/api/users/2>, inilah alamat lengkap untuk mengakses API dari "SINGLE USER".

Pada kotak "Response" di sisi kanan atas (4), terlihat angka HTTP response dari API request. Angka 200 menandakan kalau API request berjalan lancar dan tidak terjadi error.

Terakhir di kotak kanan (5), terlihat data JSON dari API yang sedang diakses. Karena kita memilih "SINGLE USER", akan tampil data JSON untuk user dengan id = 2.

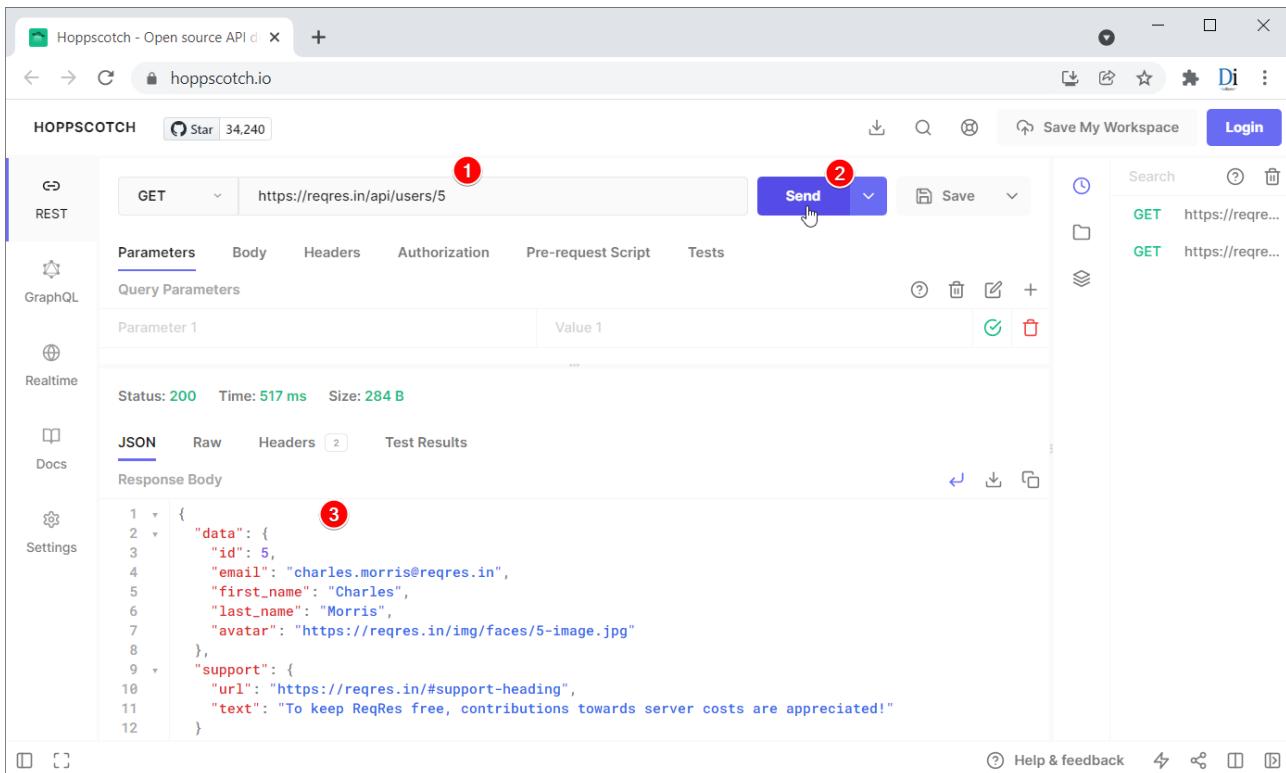
Untuk API yang menggunakan method `get`, hasilnya bisa langsung terlihat di web browser. Sebagai contoh, silahkan ketik <https://reqres.in/api/users/5> di address bar web browser, maka akan tampil string JSON dari user dengan id = 5:

Gambar: Tampilan data JSON dari alamat <https://reqres.in/api/users/5>

Ini adalah teks dalam format JSON, yang nantinya bisa kita konversi menjadi JavaScript object. Untuk data yang agak panjang, hasil JSON biasanya susah untuk dibaca. Solusinya, terdapat tools khusus untuk mempermudah proses pembacaan data JSON. Yang cukup populer adalah **Postman**: www.postman.com.

Postman tersedia dalam bentuk aplikasi desktop (harus diinstall terlebih dahulu), dan juga ada yang berbasis web. Akan tetapi kita harus register dan pilihan menu yang ada menurut saya agak rumit untuk pemula.

Untungnya, terdapat alternatif yang lebih simple dan bisa langsung pakai, yakni hoppscotch.io.

Gambar: Merapikan tampilan JSON API dengan <https://hoppscotch.io>

Begitu halaman hoppscotch.io terbuka, kita bisa langsung ketik alamat URL API di kotak inputan atas (1). Setelah itu klik tombol "Send" (2) dan hasil data JSON akan tampil di bagian bawah (3). Data JSON ini sudah di format agar mudah dibaca, dan berikut saya tampilkan

kembali:

```

1  {
2    "data": {
3      "id": 5,
4      "email": "charles.morris@reqres.in",
5      "first_name": "Charles",
6      "last_name": "Morris",
7      "avatar": "https://reqres.in/img/faces/5-image.jpg"
8    },
9    "support": {
10      "url": "https://reqres.in/#support-heading",
11      "text": "To keep ReqRes free, contributions towards server costs are
12      appreciated!"
13    }
14 }
```

Di sini kita bisa lihat bahwa `reqres.in/api/users/5` mengembalikan data JSON yang terdiri dari 2 object: `data` dan `support`. Object `data` berisi property `id`, `email`, `first_name`, `last_name`, dan `avatar`. Sedangkan object `support` berisi property `url` dan teks yang tidak lain pesan promosi dari `reqres.in`.

Memeriksa struktur JSON seperti ini sangat disarankan sebelum masuk ke kode program.

Fungsi `fetch()` JavaScript

Sekarang saatnya masuk ke kode program. Fetch API menyediakan fungsi `fetch()` untuk memproses data API. Fungsi `fetch()` bisa menerima 2 buah argument: argument pertama berupa alamat URL API, dan argument kedua untuk konfigurasi request.

Berikut format dasar dari fungsi `fetch()`:

```
fetch('<alamat_URL_API>', { <konfigurasi> });
```

Argument kedua dari fungsi `fetch()` bersifat opsional, yang jika tidak ditulis akan memakai pengaturan default bawaan JavaScript. Pengaturan default ini antara lain memakai HTTP method `get`, sehingga jika kita mengakses API dengan HTTP method `get`, argument kedua tidak perlu ditulis.

Fetch API punya berbagai fitur dan konfigurasi yang cukup panjang jika dibahas semua.

Di sini kita akan membahas fitur-fitur utama saja. Materi yang lebih detail akan menjadi jatah buku **JavaScript in Depth** (mudah-mudahan bisa menyusul).

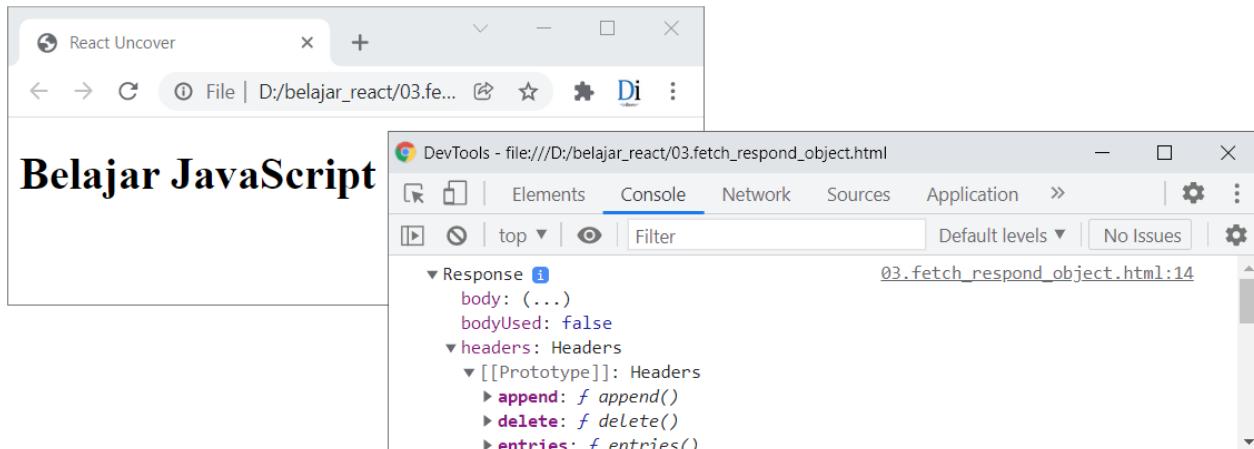
Fungsi `fetch()` mengembalikan nilai dalam bentuk `promise`. Yup, jika anda melompati materi "Asynchronous JavaScript" di bab 2, sekarang saat yang tepat untuk mempelajarinya. Berikut contoh penggunaan dari fungsi `fetch()`:

03.fetch_respond_object.html

```

1 <script>
2   fetch('https://reqres.in/api/users/1')
3     .then(response => console.log(response)) // berisi response object
4 </script>

```



Gambar: Hasil dari fungsi fetch() JavaScript

Di baris 3, terdapat perintah `fetch()` dengan argument berupa alamat URL '`https://reqres.in/api/users/1`'. Alamat ini nantinya mengembalikan string dalam format JSON.

Karena `fetch()` mengembalikan nilai dalam bentuk `promise`, maka bisa langsung di sambung dengan method `.then()` seperti di baris 3. Hasilnya kemudian di tampilkan ke tab console.

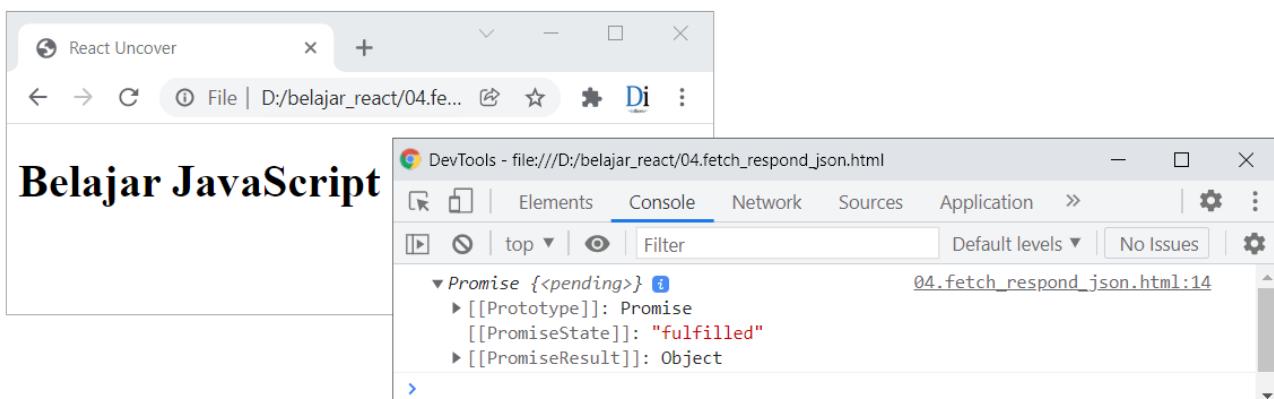
Akan tetapi data yang didapat belum berbentuk JSON, tapi **response object**. Object ini berisi berbagai info tambahan tentang request yang terjadi. Jika kita ingin mengakses data JSON, jalankan method `json()` dari response object seperti contoh berikut:

04.fetch_respond_json.html

```

1 <script>
2   fetch('https://reqres.in/api/users/1')
3     .then(response => console.log(response.json())) // berbentuk promise lagi
4 </script>

```

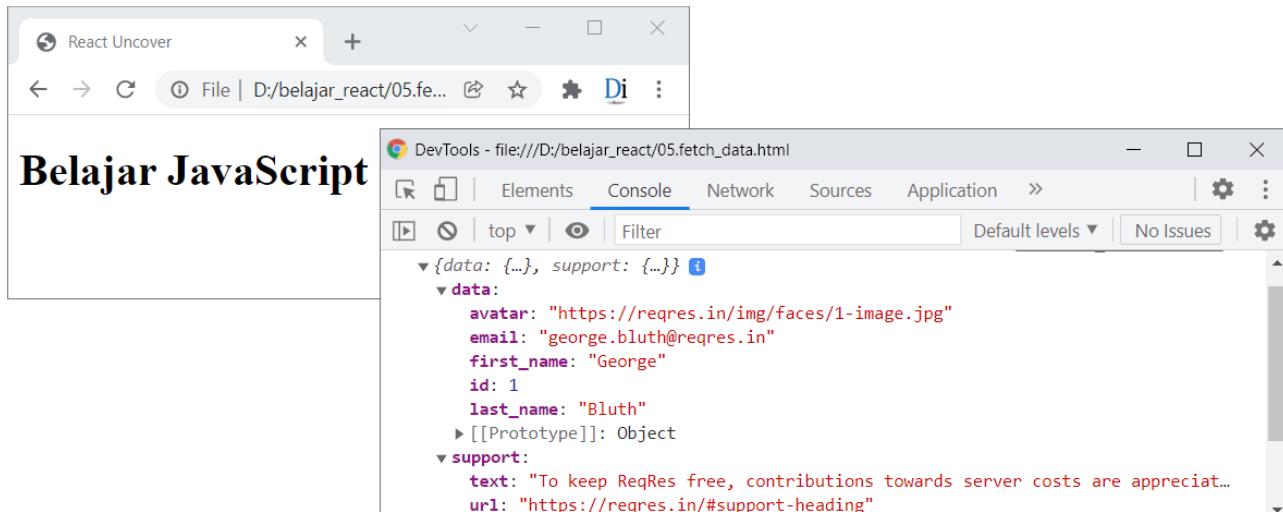


Gambar: Hasil dari fungsi fetch() dan json()

Namun jika ditulis seperti ini, hasil dari `response.json()` ternyata juga berbentuk `promise`, maka kita harus sambung kembali dengan method `then()` sekali lagi:

05.fetch_data.html

```
1 <script>
2   fetch('https://reqres.in/api/users/1')
3     .then(response => response.json())
4     .then(data => console.log(data));
5 </script>
```



Gambar: Data JSON sukses diakses

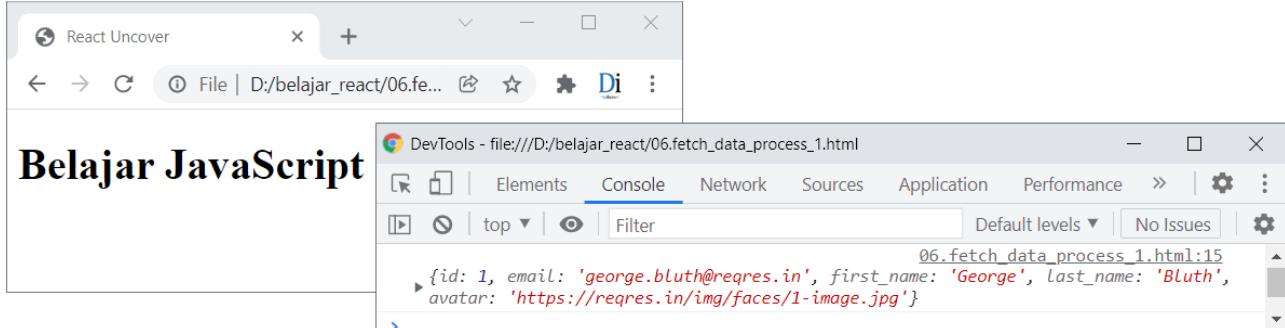
Akhirnya data JSON dari `https://reqres.in/api/users/1` sudah berhasil di akses. Method `.json()` juga otomatis mengkonversi string JSON menjadi JavaScript object, yang dalam contoh di atas tersimpan dalam variabel `data`.

Pada jendela console bisa terlihat isi dari variabel `data`, yakni sebuah object yang berisi property `data` dan `support`. Property `data` ternyata juga berbentuk object dengan 5 property: `id`, `email`, `first_name`, `last_name`, dan `avatar`. Data ini sama seperti tampilan JSON yang kita tes menggunakan [hoppscotch.io](#).

Sedikit catatan, data API dari web `reqres.in` tersimpan di dalam property `data`, yang kebetulan sama dengan nama variabel yang saya pakai untuk mengakses JSON object. Maka jika kita ingin mengakses object yang tersimpan di dalam property `data`, bisa dengan cara berikut:

06.fetch_data_process_1.html

```
1 <script>
2   fetch('https://reqres.in/api/users/1')
3     .then(response => response.json())
4     .then(data => console.log(data.data));
5 </script>
```



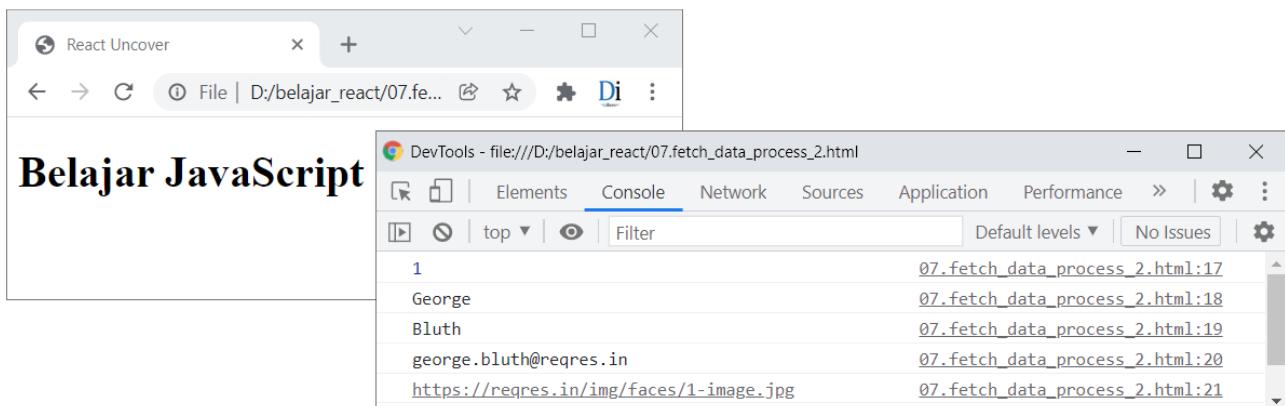
Gambar: Mengakses property data

Bagaimana jika kita ingin mengakses setiap property data secara terpisah? Tidak masalah, caranya sama seperti mengakses property dari object JavaScript biasa:

07.fetch_data_process_2.html

```

1 <script>
2   fetch('https://reqres.in/api/users/1')
3     .then(response => response.json())
4     .then(data => {
5       let user = data.data;
6       console.log(user.id);
7       console.log(user.first_name);
8       console.log(user.last_name);
9       console.log(user.email);
10      console.log(user.avatar);
11    });
12 </script>
```



Gambar: Mengakses data API reqres.in secara terpisah

Sip, kita sudah berhasil mengakses API dari reqres.in menggunakan fetch API bawaan JavaScript. Untuk uji coba, silahkan ganti id user yang diakses seperti `fetch('https://reqres.in/api/users/4')`, atau `fetch('https://reqres.in/api/users/10')`, maka data yang didapat juga akan berbeda.

Pemrosesan API membuka banyak ide program. Sedikit tantangan kecil, bisa coba buat 5 tombol yang ketika diklik akan menampilkan data user yang berbeda-beda (user dengan id 1

sampai 5). Ini semua bisa dibuat dengan kode JavaScript native.

Error Handling Fetch API

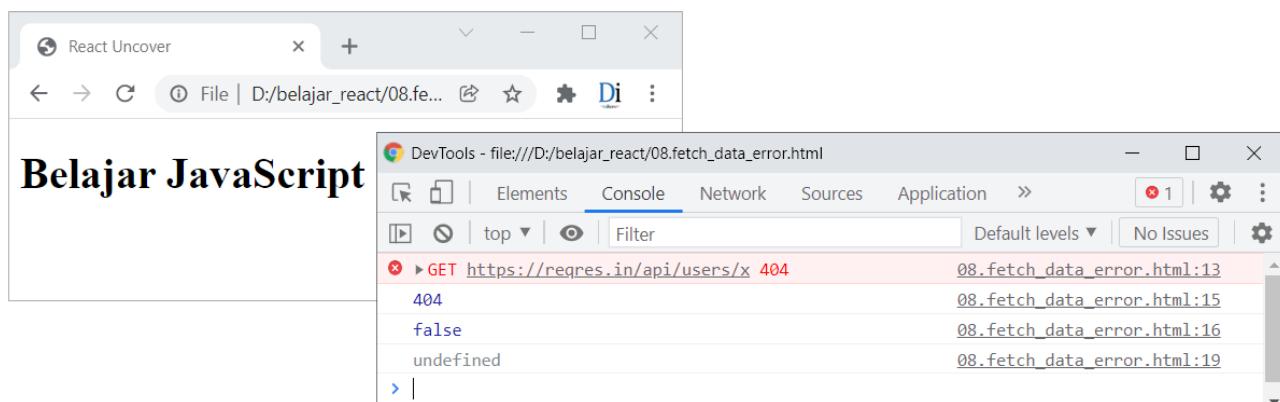
Mengakses data dari API memang praktis, akan tetapi karena kita bergantung ke server lain, selalu ada kemungkinan error. Bagaimana jika web `reqres.in` tiba-tiba mengalami gangguan dan tidak bisa di akses? Atau bagaimana jika alamat URL yang diketik salah?

Terdapat beberapa cara untuk mengidentifikasi kesalahan koneksi API. Salah satunya dengan memeriksa nilai property **status** dan **ok** dari response object:

`08.fetch_data_error.html`

```

1 <script>
2   fetch('https://reqres.in/api/users/x')
3     .then(response => {
4       console.log(response.status);
5       console.log(response.ok);
6       return response.json()
7     })
8     .then(data => console.log(data.data));
9 </script>
```



Gambar: Simulasi salah ketik alamat URL API

Alamat URL API sengaja saya buat salah dengan mengisi nomor id user sebagai 'x'. Ini tentu menjadi error karena `reqres.in` mewajibkan nomor id user dengan angka saja.

Response object sebagai hasil dari fungsi `fetch()` memiliki berbagai property, diantaranya `response.status` untuk memeriksa kode HTTP response (baris 4). Dalam contoh ini, hasilnya adalah 404 yang berarti halaman tidak ditemukan.

Selain 404, masih banyak kode HTTP response lain yang menandakan ada sesuatu yang salah. Gambaran umum, jika kodennya selain 200, maka terjadi gangguan. Daripada memeriksa setiap kode, response object juga menyediakan property `response.ok` yang berisi boolean `true` atau `false` tergantung apakah koneksi dengan API berhasil atau tidak. Dalam contoh kita, hasilnya adalah `false` karena koneksi gagal (baris 5).

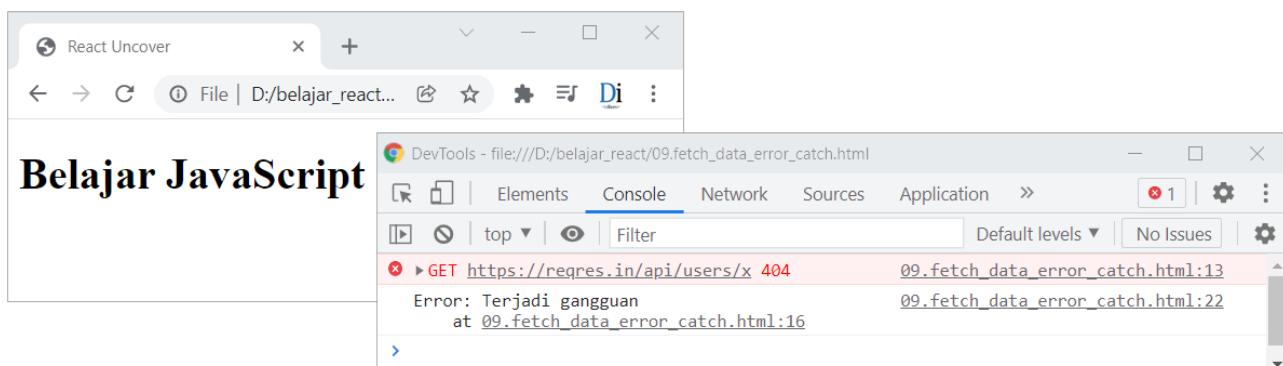
Jika anda penasaran kenapa kali ini ada perintah `return` untuk `response.json()` di baris 6, ini karena isi dari method `.then()` sudah tidak 1 baris lagi. Sebelumnya kita tidak perlu perintah `return` karena memakai penulisan singkat dari arrow notation.

Hasil dari `response.ok` bisa kita pakai untuk membuat percabangan kode program:

09.fetch_data_error_catch.html

```

1 <script>
2   fetch('https://reqres.in/api/users/x')
3     .then(response => {
4       if (!response.ok) {
5         throw new Error('Terjadi gangguan');
6       }
7       return response.json()
8     })
9     .then(data => console.log(data.data))
10    .catch(error => {
11      console.log(error);
12    });
13 </script>
```



Gambar: Error handling dengan `response.ok`

Di dalam block `.then()` antara baris 4-7 terdapat percabangan `if`, yakni `if(!response.ok)`. Seperti yang dijelaskan sebelumnya, `response.ok` nantinya berisi boolean `false` jika koneksi bermasalah. Maka dengan tambahan tanda "!", logika ini dibalik. Perintah `!response.ok` akan menjadi `true` pada saat koneksi bermasalah. Jika ini terjadi, buat sebuah `Error` object di baris 5 yang nantinya akan "ditangkap" oleh percabangan `catch()` di baris 11.

Dengan struktur kode seperti ini, block `.then()` di baris 9 hanya akan berjalan hanya jika koneksi API berhasil diakses. Jika ternyata API gagal, kode program akan langsung lompat ke block `catch()`. Ini akan menghindari error lain yang mungkin terjadi jika web browser tetap menjalankan kode program utama yang mengakses data API.

Akan tetapi ketika tab console di buka, tetap tampil error `GET https://reqres.in/api/users/x 404`. Ini berasal dari pengaturan Google Chrome dan [tidak bisa dihilangkan](#)²⁶.

26. <https://stackoverflow.com/questions/4500741/suppress-chrome-failed-to-load-resource-messages-in-console>

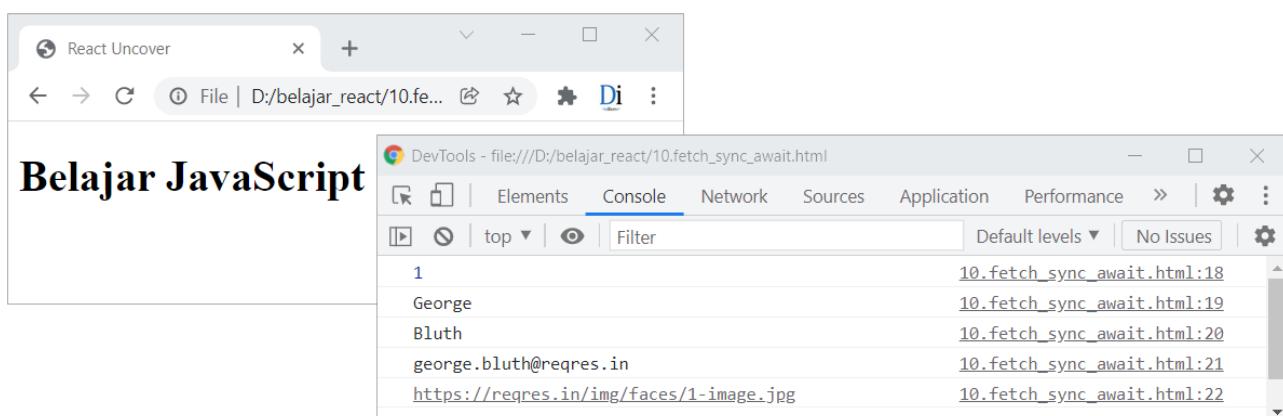
Mengakses Fetch API dengan Async Await

Selain memakai method `.then()`, kita juga bisa memakai penulisan **async await** untuk pemrosesan `promise`. Dalam kebanyakan kasus, kode programnya menjadi lebih sederhana:

10.fetch_sync_await.html

```

1 <script>
2   const myFetch = async () => {
3     let response = await fetch('https://reqres.in/api/users/1');
4     let data = await response.json();
5     let user = data.data;
6
7     console.log(user.id);
8     console.log(user.first_name);
9     console.log(user.last_name);
10    console.log(user.email);
11    console.log(user.avatar);
12  }
13
14  myFetch();
15 </script>
```



Gambar: Proses fetch API dengan async await

Syarat dari penulisan `async await` adalah, perintah yang menghasilkan `promise` harus dijalankan dari dalam function, yang dalam contoh di atas ada di fungsi `myFetch()` antara baris 2-12. Keyword `async` juga harus ditulis sebelum argument seperti di baris 2.

Di baris 3, hasil dari fungsi `fetch()` ditampung ke dalam variabel `response`. Karena perintah ini menghasilkan `promise`, maka harus diawali dengan keyword `await`. Begitu juga di baris 4, dimana `response.json()` juga mengembalikan `promise` dan disimpan ke dalam variabel `data`.

Setelah 2 kali pemrosesan `promise`, isi `data.data` disimpan ke dalam variabel `user` di baris 5 . Sesampainya di sini, variabel `user` sudah berisi data 1 user dan bisa langsung diakses melalui perintah `user.id`, `user.first_name`, dst.

Terakhir, fungsi `myFetch()` dipanggil di baris 14.

Error Handling Fetch API dengan Async Await

Proses pendektsian error juga bisa dilakukan dalam bentuk *async await*:

11.fetch_sync_await_error.html

```

1 <script>
2   const myFetch = async () => {
3     let response = await fetch('https://reqres.in/apis/users/1');
4     if (!response.ok) {
5       throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
6     }
7     let data = await response.json();
8     let user = data.data;
9
10    console.log(user.id);
11    console.log(user.first_name);
12    console.log(user.last_name);
13    console.log(user.email);
14    console.log(user.avatar);
15  }
16
17  myFetch().catch(error => console.log(error));
18 </script>
```

Pemeriksaan koneksi API ada di baris 4, yang jika terpenuhi (artinya proses `fetch` gagal), "lempar" sebuah **Error** object dengan pesan yang ingin ditampilkan. Object ini kemudian ditangkap oleh block `catch()` setelah pemanggilan fungsi `myFetch()` di baris 17.

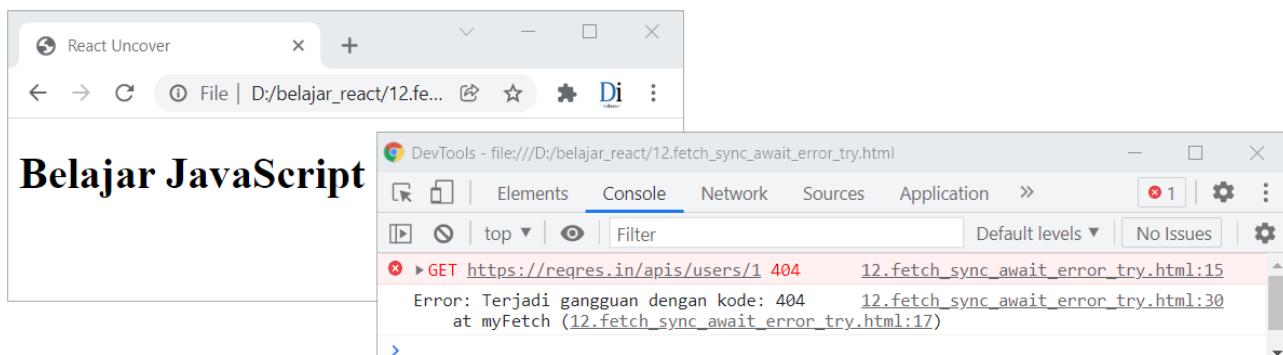
Alternatif penulisan lain bisa juga membuat block try-catch di dalam fungsi `async`:

12.fetch_sync_await_error_try.html

```

1 <script>
2   const myFetch = async () => {
3     try {
4       let response = await fetch('https://reqres.in/apis/users/1');
5       if (!response.ok) {
6         throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
7       }
8
9       let data = await response.json();
10      let user = data.data;
11
12      console.log(user.id);
13      console.log(user.first_name);
14      console.log(user.last_name);
15      console.log(user.email);
16      console.log(user.avatar);
17    }
18    catch (error) {
19      console.log(error);
20    }
21  }
```

```
22
23     myFetch();
24 </script>
```



Gambar: Error handling dengan try-catch

Kali ini semua kode program milik fungsi `myFetch()` ada di antara block `try` pada baris 3-17. Jika terdapat perintah yang membuat exception (menjalankan `throw new Error`), pemrosesan akan berhenti dan kode program langsung lompat ke block `catch` di baris 18-20.

Dengan cara ini, fungsi `myFetch()` cukup dipanggil dengan cara biasa di baris 23.

18.6. Memproses Hasil API di React

Sekarang kita sampai ke bagian paling penting dari bab ini, yaitu cara memproses hasil API menggunakan React.

React tidak memiliki perintah khusus untuk mengakses API, sehingga `fetch` API bawaan JavaScript sudah mencukupi. Alternatif lain bisa juga menggunakan library terpisah seperti **jQuery AJAX** atau **Axios**.

Yang menjadi perhatikan adalah dimana posisi perintah `fetch()` sebaiknya di tulis?

Posisi paling pas ada di dalam method `componentDidMount()`, yakni tepat setelah semua element HTML tampil di web browser. Ini agar halaman bisa tampil terlebih dahulu sembari menunggu proses pemanggilan API. Untuk server API yang sibuk, bisa saja butuh waktu beberapa detik sebelum data JSON sampai ke React. Selama menunggu kita bisa menampilkan animasi loading agar user mendapatkan *feedback* bahwa data sedang di proses.

Setelah web tampil, jika kita ingin melakukan pemanggilan API lanjutan, perintah `fetch()` bisa ditempatkan ke dalam method `componentDidUpdate()`.

Untuk versi functional component, perintah `fetch()` akan berada di dalam **useEffect** hook. Terlebih pemanggilan API bukanlah tugas utama React, sehingga itu menjadi sebuah "side effect".

Sebagai contoh kode program saya akan langsung memakai versi functional component:

13.react_fetch.html

```
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8  </head>
9
10 <body>
11     <div id="root"></div>
12
13     <script src="js/react.development.js"></script>
14     <script src="js/react-dom.development.js"></script>
15     <script src="js/babel.js"></script>
16     <script type="text/babel">
17
18     const User = () => {
19         const [user, setUser] = React.useState([]);
20
21         React.useEffect(() => {
22             const myFetch = async () => {
23                 try {
24                     let response = await fetch('https://reqres.in/api/users/2');
25                     if (!response.ok) {
26                         throw new Error(`Terjadi gangguan dengan kode:
27                                         ${response.status}`);
28                     }
29                     let data = await response.json();
30                     setUser(data.data);
31                 }
32                 catch (error) {
33                     console.log(error);
34                 }
35             }
36             myFetch();
37         }, []);
38
39         return (
40             <div>
41                 <h1>Data User</h1>
42                 <ul>
43                     <li>{user.id}</li>
44                     <li>{user.first_name}</li>
45                     <li>{user.last_name}</li>
46                     <li>{user.email}</li>
47                     <li>{user.avatar}</li>
48                 </ul>
49             </div>
50         )
51     }
52 }
```

```

54     ReactDOM.createRoot(document.getElementById('root')).render(<User />);
55
56   </script>
57 </body>
58
59 </html>

```

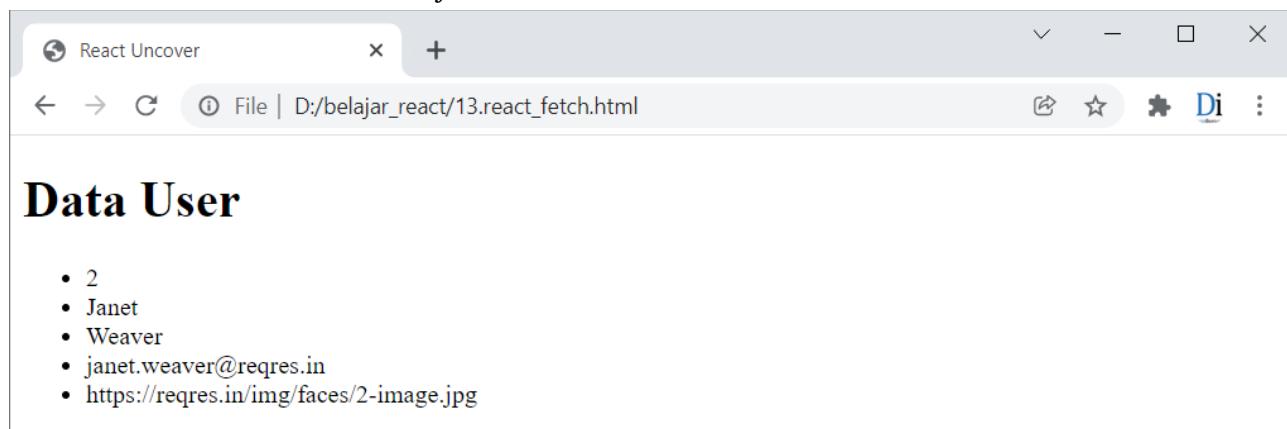
Dalam kode program ini, saya membuat komponen User antara baris 18-52. Di awal komponen, terdapat deklarasi state user yang nantinya akan diisi dengan data user dari API reqres.in.

Pemanggilan API dilakukan di dalam `useEffect()` antara baris 21-37. Di sini saya menggunakan teknik `async-await` untuk memproses fungsi `fetch()`, sehingga perlu mendeklarasikan fungsi `myFetch()` di dalam `useEffect` hook. Kode yang diperlukan kurang lebih sama seperti praktik kita sebelumnya, yakni menambah keyword `async` sebelum penulisan argument di baris 22, serta membuat block `try-catch` untuk mengantisipasi error API.

Yang berbeda ada di baris 30, dimana ketika API sukses berjalan, isi data tersebut ke dalam state user dengan perintah `setUser(data.data)`. Tidak lupa, fungsi `myFetch()` harus dipanggil di baris 36.

Sebagai argument kedua dari `useEffect()`, saya isi dengan array kosong " [] " di baris 37. Ini menginstruksikan React agar menjalankan `useEffect` 1 kali setelah proses loading, yang sama seperti `componentDidMount()`.

Di dalam kode JSX, data dari state user ditampilkan sebagai unordered list dengan tag `` antara baris 43-47. Silahkan tes jalankan file ini:



Gambar: Menampilkan data API reqres.in menggunakan React

Saat dijalankan pertama kali, mungkin anda bisa melihat icon bulatan list tampil kosong sekitar 1 detik, dan kemudian baru berisi data. Ini adalah jeda antara React memproses kode JSX sampai dengan data API dari `reqres.in` tiba untuk diproses.

Inilah teknik dasar pemrosesan API menggunakan React. Dengan mengubah struktur kode JSX serta sedikit bantuan CSS, tampilan data bisa dibuat lebih menarik:

14.react_fetch_style.html

```

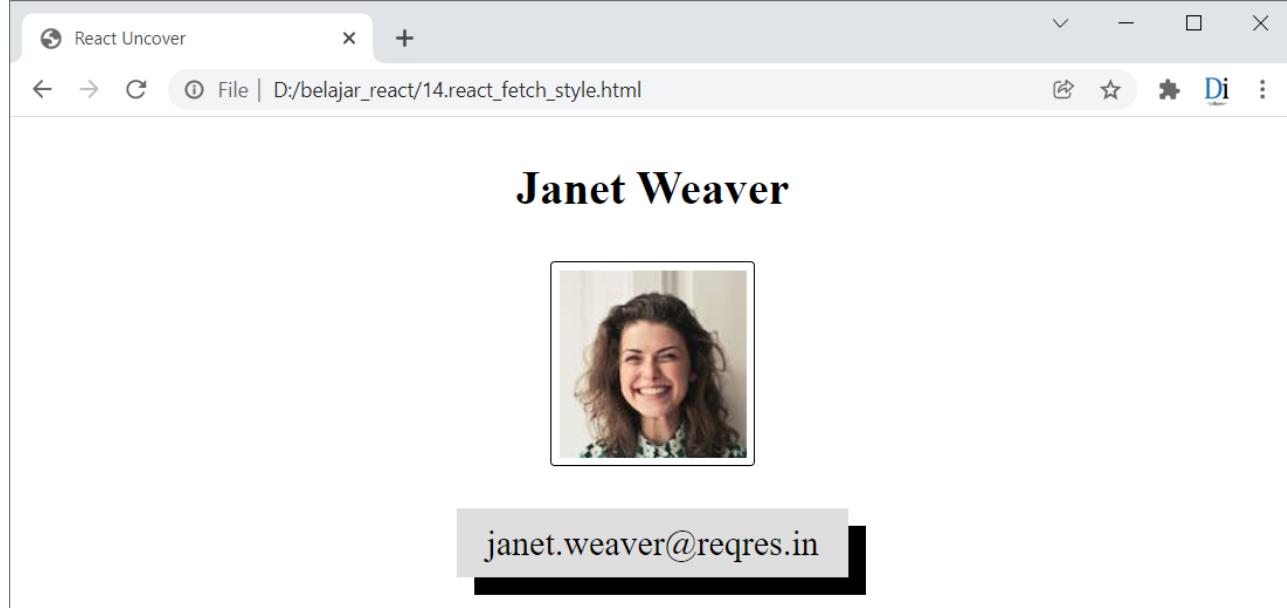
1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14          div {
15              text-align: center;
16          }
17
18          img {
19              margin: 10px;
20              border: 1px solid black;
21              padding: 5px;
22              border-radius: 3px 3px;
23          }
24
25          figcaption {
26              text-align: center;
27              background-color: gainsboro;
28              font-size: 1.5em;
29              padding: 10px 20px;
30              margin: 15px 0;
31              box-shadow: 12px 12px 0px black;
32          }
33      </style>
34  </head>
35
36  <body>
37      <div id="root"></div>
38
39      <script src="js/react.development.js"></script>
40      <script src="js/react-dom.development.js"></script>
41      <script src="js/babel.js"></script>
42      <script type="text/babel">
43
44          const User = () => {
45              const [user, setUser] = React.useState([]);
46
47              React.useEffect(() => {
48                  const myFetch = async () => {
49                      try {
50                          let response = await fetch('https://reqres.in/api/users/2');
51                          if (!response.ok) {
52                              throw new Error(`Terjadi gangguan dengan kode:
53                                              ${response.status}`);

```

```

54         }
55         let data = await response.json();
56         setUser(data.data);
57     }
58     catch (error) {
59         console.log(error);
60     }
61 }
62 myFetch();
63 ], []);
64
65 return (
66     <div>
67         <h1>`${user.first_name} ${user.last_name}`</h1>
68         <figure>
69             <img src={user.avatar} alt={`${user.first_name}`}>
70             <figcaption>{user.email}</figcaption>
71         </figure>
72     </div>
73 )
74
75 }
76 ReactDOM.createRoot(document.getElementById('root')).render(<User />);
77 </script>
78 </body>
79
80 </html>

```



Gambar: Hasil styling data API reqres.in

Di awal kode HTML terdapat tambahan kode CSS. Ini dipakai untuk mempercantik tampilan JSX antara baris 66-72. Struktur yang dipakai mirip seperti latihan menampilkan data mahasiswa di bab-bab sebelumnya. Untuk gambar user, diambil dari data `user.avatar` yang juga tersedia di server `reqres.in` (baris 69).

Silahkan tes juga dengan mengubah id user API, misalnya ganti jadi <https://reqres.in/api/users/5> atau <https://reqres.in/api/users/11>.

Error Handling Fetch API di React

Apabila URL API tidak bisa diakses, sebaiknya kita juga tampilkan pesan error dalam bentuk pemberitahuan di web browser, misalnya dengan pesan "Terjadi Gangguan..." atau pesan error lain. Ini bisa dibuat dengan tambahan state `errorStatus` seperti contoh berikut:

15.react_fetch_error.html

```

1  const User = () => {
2      const [user, setUser] = React.useState([]);
3      const [errorStatus, setErrorStatus] = React.useState(false);
4
5      React.useEffect(() => {
6          const myFetch = async () => {
7              try {
8                  let response = await fetch('https://reqres.in/api/users/2');
9                  if (!response.ok) {
10                      setErrorStatus(true);
11                      throw new Error(`Terjadi gangguan dengan kode:
12                                         ${response.status}`);
13                  }
14                  let data = await response.json();
15                  setUser(data.data);
16              }
17              catch (error) {
18                  console.log(error);
19              }
20          }
21          myFetch();
22      }, []);
23
24      return (
25          <div>
26              {errorStatus ?
27                  (
28                      <h1>Terjadi Gangguan...</h1>
29                  )
30              :
31                  (
32                      <React.Fragment>
33                          <h1>`${user.first_name} ${user.last_name}`</h1>
34                          <figure>
35                              <img src={user.avatar} alt={`${user.first_name}`} />
36                              <figcaption>{user.email}</figcaption>
37                          </figure>
38                      </React.Fragment>
39                  )
40              }
41          </div >
42      )
}

```

```

43     )
44   }
45 }
46 ReactDOM.createRoot(document.getElementById('root')).render(<User />);

```

Di baris 3 terdapat tambahan state `errorStatus` dengan nilai awal `false`. Dalam struktur JSX, state ini diperiksa di baris 26. Jika berisi `true`, maka yang akan diproses adalah block kode program antara 27-29, yang alam contoh ini menampilkan tag `<h1>Terjadi Gangguan...</h1>`.

Jika ternyata `errorStatus` berisi `false`, maka yang akan diproses adalah blok kode program di baris 31-39, yakni tampilkan semua data user seperti biasa.

Kapan `errorStatus` bisa berisi `false`? Perintah tersebut ada di baris 10, yakni pada saat property `response.ok` menghasilkan nilai `false`, yang berarti API gagal di akses.

Alamat URL API di baris 8 sengaja saya buat salah, yang seharusnya `reqres.in/api/` ditulis menjadi `reqres.in/apis/`. Berikut hasilnya di web browser:



Gambar: Pesan yang tampil saat API gagal diakses

Inilah salah satu cara untuk menampilkan pesan error API di React.

Membuat Pesan Loading

Selain pesan error, kita juga bisa membuat pesan loading yang tampil selama proses pengambilan data dari server API. Berikut salah satu teknik yang bisa dipakai:

```

16.react_fetch_loading.html

1 const User = () => {
2   const [user, setUser] = React.useState([]);
3   const [errorStatus, setErrorStatus] = React.useState(false);
4   const [loadingStatus, setLoadingStatus] = React.useState(true);
5
6   React.useEffect(() => {
7     const myFetch = async () => {
8       try {
9         let response = await fetch('https://reqres.in/api/users/5?delay=2');
10        if (!response.ok) {
11          setErrorStatus(true);
12          throw new Error(`Terjadi gangguan dengan kode:
13                           ${response.status}`);
14        }

```

```

15     let data = await response.json();
16     setUser(data.data);
17   }
18   catch (error) {
19     console.log(error);
20   }
21   finally {
22     setLoadingStatus(false);
23   }
24 }
25 myFetch();
26 }, []);
27
28 return (
29   <div>
30     {loadingStatus ?
31       (
32         <h1>Loading...</h1>
33       )
34     :
35       (
36         errorStatus ?
37           (
38             <h1>Terjadi Gangguan...</h1>
39           )
40         :
41           (
42             <React.Fragment>
43               <h1>`${user.first_name} ${user.last_name}`</h1>
44               <figure>
45                 <img src={user.avatar} alt={`${user.first_name}`}>
46                 <figcaption>{user.email}</figcaption>
47               </figure>
48             </React.Fragment>
49           )
50         )
51       }
52     </div >
53   )
54 }
55
56 ReactDOM.createRoot(document.getElementById('root')).render(<User />);

```

Sekarang terdapat tambahan 1 state lagi bernama `loadingStatus` dengan nilai awal `true` di baris 4. State inilah yang akan menjadi penanda apakah data dari server API sudah sampai atau belum. Selama nilainya `true`, maka dianggap masih proses loading. Lalu kapan nilai state ini di update? Perintah itu ada di dalam block `finally` pada baris 22, yakni `setLoadingStatus(false)`.

Di dalam struktur kondisi **try-catch-finally**, block `finally` akan selalu dijalankan setelah block `try` atau block `catch` selesai di proses. Ini menjadi posisi yang paling pas untuk mengupdate state `loadingStatus`, karena artinya proses pengambilan data ke server sudah selesai. Tidak masalah apakah proses pengambilan data berhasil (block `try` dijalankan), atau terjadi error

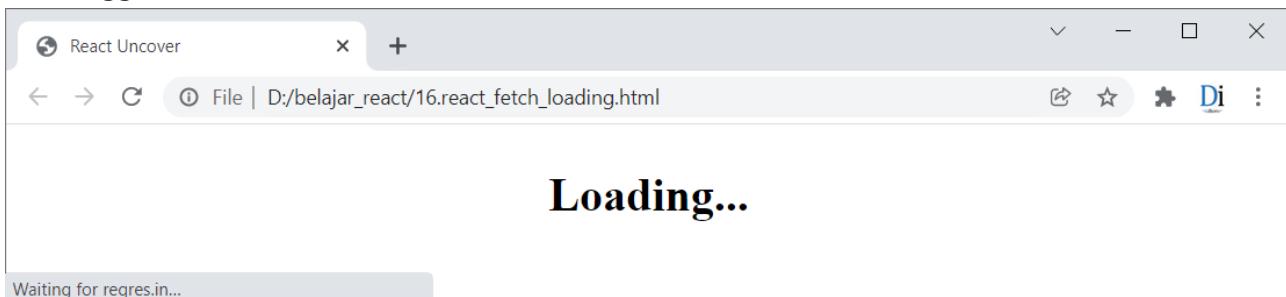
(block catch yang dijalankan).

Di dalam struktur JSX, isi state `loadingStatus` akan diperiksa di baris 30. Jika isinya `true` (yang berarti data API masih belum sampai), tampilkan blok kode program antara baris 31-33, yakni sebuah tag `<h1>Loading...</h1>`.

Ketika data API sudah sampai, isi state `loadingStatus` akan di update menjadi `false` dan kode JSX di render ulang. Karena sekarang nilainya `false`, maka yang di proses adalah block kode program antara baris 35-50, yakni proses menampilkan data API.

Sebagai alamat URL API, saya menambah query string `?delay=2` di akhir baris 9. Ini adalah fitur tambahan yang disediakan web `reqres.in` untuk simulasi proses loading. Angka 2 di sini akan membuat proses pengambilan data API berlangsung selama 2 detik.

Silahkan tes kode di atas, teks "Loading" akan tampil selama 2 detik di awal pada saat menunggu data API:



Gambar: Tampilan loading menunggu data API

Teks "Loading..." ini bisa saja diganti dengan sebuah animasi gambar .gif agar lebih menarik.

Struktur JSX dari kode program diatas terasa agak kurang rapi karena terdapat pemeriksaan bersarang (nested). Meskipun tidak salah, alternatif penulisan juga bisa seperti ini:

17.react_fetch_loading_refactor.html

```

1  const User = () => {
2      const [user, setUser] = React.useState([]);
3      const [errorStatus, setErrorStatus] = React.useState(false);
4      const [loadingStatus, setLoadingStatus] = React.useState(true);
5
6      React.useEffect(() => {
7          ...
8          ...
9      }, []);
10
11     if (loadingStatus) {
12         return <div><h1>Loading...</h1></div>
13     }
14     else if (errorStatus) {
15         return <div><h1>Terjadi Gangguan...</h1></div>
16     }
17     else {

```

```

18     return (
19       // prakteknya nanti, setiap bagian bisa dibuat menjadi komponen terpisah
20       <div>
21         <h1>`${user.first_name} ${user.last_name}`</h1>
22         <figure>
23           <img src={user.avatar} alt={`${user.first_name}`}>
24           <figcaption>${user.email}</figcaption>
25         </figure>
26       </div>
27     )
28   }
29 }
30
31 ReactDOM.createRoot(document.getElementById('root')).render(<User />);

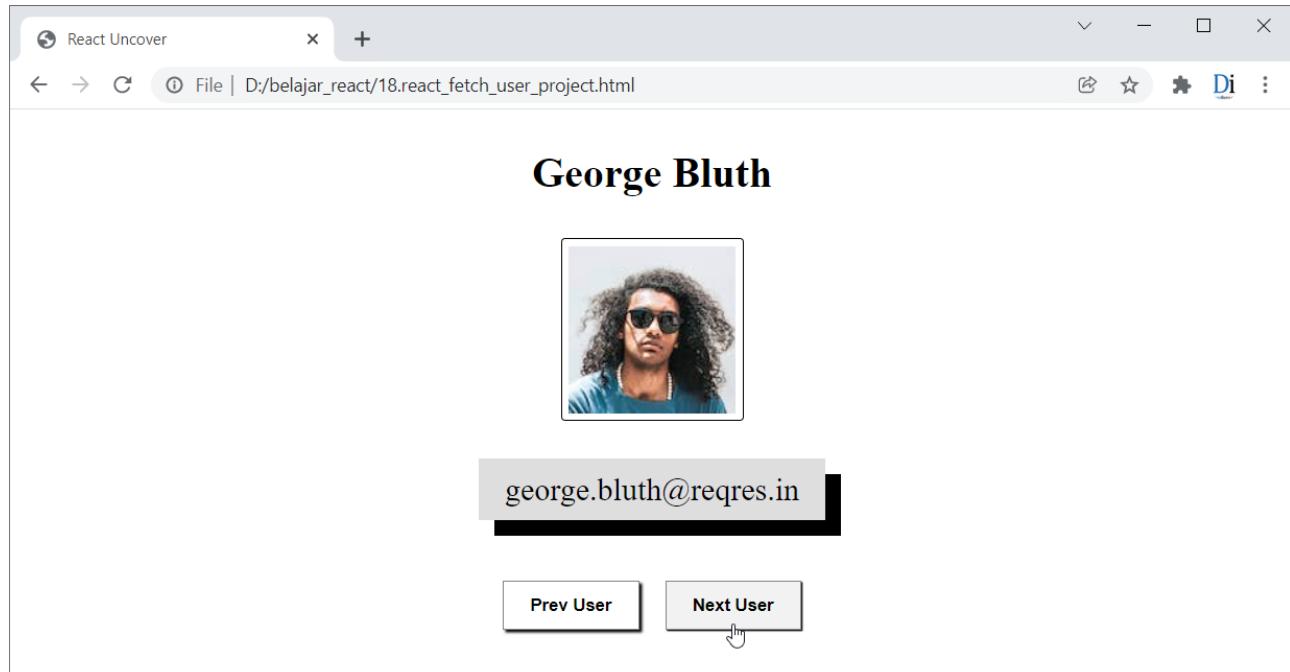
```

Proses pemeriksaan state `loadingStatus` dan `errorStatus` saya buat terpisah dengan kondisi if else biasa. Jika salah satunya terpenuhi, langsung jalankan perintah `return` dengan kode JSX yang sesuai.

Dalam prakteknya nanti, setiap bagian if else bisa saja me-return komponen React lain agar kode program menjadi lebih rapi. Misalnya jika terjadi gangguan API, tampilkan komponen React yang berisi menu navigasi agar user bisa mencari informasi lain.

18.7. Mini Project: Fetch User

Sebagai penutup materi tentang Fetch API, saya ingin membuat mini project sederhana. Kita masih menggunakan data dari `reqres.in`, dengan tambahan tombol Prev dan Next untuk beralih ke user lain. Berikut tampilan akhir yang akan dibuat:



Gambar: Tampilan mini project fetch user

Sebagian besar kode yang diperlukan sudah kita pelajari, tinggal bagaimana membuat logika untuk tombol "Prev User" dan "Next User".

Untuk berganti dari satu user ke user lain, tinggal mengubah 1 angka di belakang URL, dari `reqres.in/api/users/1` untuk user pertama, `reqres.in/api/users/2` untuk user kedua, dst.

Berangkat dari pola ini, kita bisa buat sebuah state sebagai angka counter, misalnya state `userId`. State `userId` kemudian disambung ke dalam penulisan alamat URL menjadi `reqres.in/pi/users/${userId}`. Pada saat tombol "Prev User" dan "Next User" di klik, update state `userId` dan jadikan sebagai depencedcy dari `useEffect` hook agar ketika nilai state berubah, data API baru akan diambil.

Berikut implementasi dari konsep ini:

```

1  <!DOCTYPE html>
2  <html lang="id">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>React Uncover</title>
8      <style>
9          #root {
10              display: flex;
11              justify-content: center;
12          }
13
14         div {
15             text-align: center;
16         }
17
18         button {
19             background-color: white;
20             cursor: pointer;
21             padding: 10px 20px;
22             font-weight: bold;
23             border: 1px solid gray;
24             box-shadow: 2px 2px 2px black;
25             margin: 30px 10px;
26         }
27
28         button:hover {
29             box-shadow: 1px 1px 1px black;
30             background-color: #f1f1f1;
31         }
32
33         img {
34             margin: 10px;
35             border: 1px solid black;
36             padding: 5px;
37             border-radius: 3px 3px;
38         }

```

```

39      figcaption {
40        text-align: center;
41        background-color: gainsboro;
42        font-size: 1.5em;
43        padding: 10px 20px;
44        margin: 15px 0;
45        box-shadow: 12px 12px 0px black;
46      }
47    
```

`</style>`
`</head>`
`50 <body>`
`51 <div id="root"></div>`
`53`
`54 <script src="js/react.development.js"></script>`
`55 <script src="js/react-dom.development.js"></script>`
`56 <script src="js/babel.js"></script>`
`57 <script type="text/babel">`
`58`
`59 const User = () => {
60 const [user, setUser] = React.useState([]);
61 const [errorStatus, setErrorStatus] = React.useState(false);
62 const [loadingStatus, setLoadingStatus] = React.useState(true);
63 const [userId, setId] = React.useState(1);
64
65 React.useEffect(() => {
66 const myFetch = async () => {
67 try {
68 setLoadingStatus(true)
69 let url = `https://reqres.in/api/users/${userId}?delay=1`;
70 let response = await fetch(url);
71 if (!response.ok) {
72 setErrorStatus(true);
73 throw new Error(`Terjadi gangguan dengan kode:
74 ${response.status}`);
75 }
76 let data = await response.json();
77 setUser(data.data);
78 setErrorStatus(false);
79 console.log(data.data);
80 }
81 catch (error) {
82 console.log(error);
83 }
84 finally {
85 setLoadingStatus(false);
86 }
87 }
88 myFetch();
89 }, [userId]);
90
91 if (loadingStatus) {
92 return <div><h1>Loading...</h1></div>
93 }`

```

94     else if (errorStatus) {
95         return (
96             <div>
97                 <h1>Terjadi Gangguan...</h1>
98                 <button onClick={() => setId(userId - 1)}>Prev User</button>
99                 <button onClick={() => setId(userId + 1)}>Next User</button>
100            </div>
101        )
102    }
103  else {
104    return (
105        <div>
106            {console.log('proses render')}
107            <h1>`${user.first_name} ${user.last_name}`</h1>
108            <figure>
109                <img src={user.avatar} alt={`${user.first_name}`}>
110                <figcaption>{user.email}</figcaption>
111            </figure>
112            <button onClick={() => setId(userId - 1)}>Prev User</button>
113            <button onClick={() => setId(userId + 1)}>Next User</button>
114        </div>
115    )
116  }
117}
118
119 ReactDOM.createRoot(document.getElementById('root')).render(<User />);
120 </script>
121 </body>
122
123 </html>

```

Di baris 63 saya membuat state `userId` dengan nilai awal 1. State inilah yang akan menampung nomor user id untuk API.

Penulisan alamat URL di baris 69 sekarang menjadi ``https://reqres.in/api/users/${userId}?delay=1``. Ini membuat alamat URL API menjadi dinamis karena id user bisa kita program berdasarkan state `userId`. Tambahan query string `?delay=1` sekedar untuk simulasi akses API yang lambat agar kita bisa melihat proses loading.

Tidak lupa, state `userId` dijadikan *dependency* dari `useEffect` hook di baris 89 agar ketika nilai state berubah, data API baru ikut diambil dan halaman di render ulang.

Untuk kode JSX, saya menggunakan teknik if-else seperti bahasan sebelumnya. Jika `loadingStatus` berisi nilai `true`, yang akan diproses adalah blok di baris 91-93. Namun apabila loading sudah selesai, cek lagi apakah `errorStatus` berisi nilai `true`. Jika iya, blok kode program di baris 94-102 yang akan di proses.

Perhatikan bahwa selain teks "`<h1>Terjadi Gangguan...</h1>`", juga terdapat 2 tag `<button>` dengan event `onClick` di baris 98-99. Event `onClick` ini berbentuk sebuah function yang akan menambah atau mengurangi state `userId` sebanyak 1 angka.

Saya sengaja menambah tombol "Prev User" dan "Next User" saat terjadi gangguan API karena bisa jadi dengan menambah/mengurangi isi state `userId`, API menjadi bisa diakses.

Kemudian jika loading sudah selesai dan tidak terjadi error API, barulah blok kode program di baris 104-115 akan di proses. Kode JSX ini menampilkan nama user, gambar avatar serta alamat email. Tidak lupa juga 2 buah tag `<button>` yang sama persis seperti sebelumnya.

Silahkan tes kode di atas di web browser.

Ketika halaman di akses pertama kali, state `userId` sudah berisi angka 1, sehingga alamat URL API yang diakses menjadi `https://reqres.in/api/users/1?delay=1`. Hasilnya, tampil user `george.bluth@reqres.in` di web browser.

Kemudian jika tombol "Next User" di klik, state `userId` akan naik menjadi 2. Ini akan men-trigger `useEffect` untuk menjalankan URL API baru, yakni `https://reqres.in/api/users/2?delay=1`. Hasilnya, tampil user `janet.weaver@reqres.in` di web browser.

Dengan men-klik tombol "Prev User" dan "Next User", kita bisa mengganti tampilan data user.

User id yang disediakan `reqres.in` sebenarnya tidak banyak, hanya sekitar 12 saja. Maka ketika tombol "Next User" di klik sekitar 11 kali, akan tampil halaman error berikut:



Gambar: Data tidak tampil karena alamat API tidak memberikan data JSON

Ini merupakan hasil dari blok `if (errorStatus)` di JSX. Dengan men-klik kembali tombol "Prev User", data user sebelumnya bisa tetap tampil.

Bisa memproses data API dengan React, membuka banyak peluang pembuatan aplikasi. Cukup dengan mencari web yang menyediakan data API, kita bisa membuat aplikasi-aplikasi menarik. Di tambah lagi cukup banyak layanan API gratis yang tinggal pakai, atau setidaknya cukup register saja.

Penyedia API gratis bisa di cek ke <https://github.com/public-apis/public-apis>. Di sana

tersedia daftar website penyedia API mulai dari data makanan, pekerjaan, musik, film, hingga anime. Salah satu yang akan kita praktikkan dalam mini project berikutnya adalah membuat web **Ilkoom Movie Database** menggunakan data API dari themoviedb.org.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

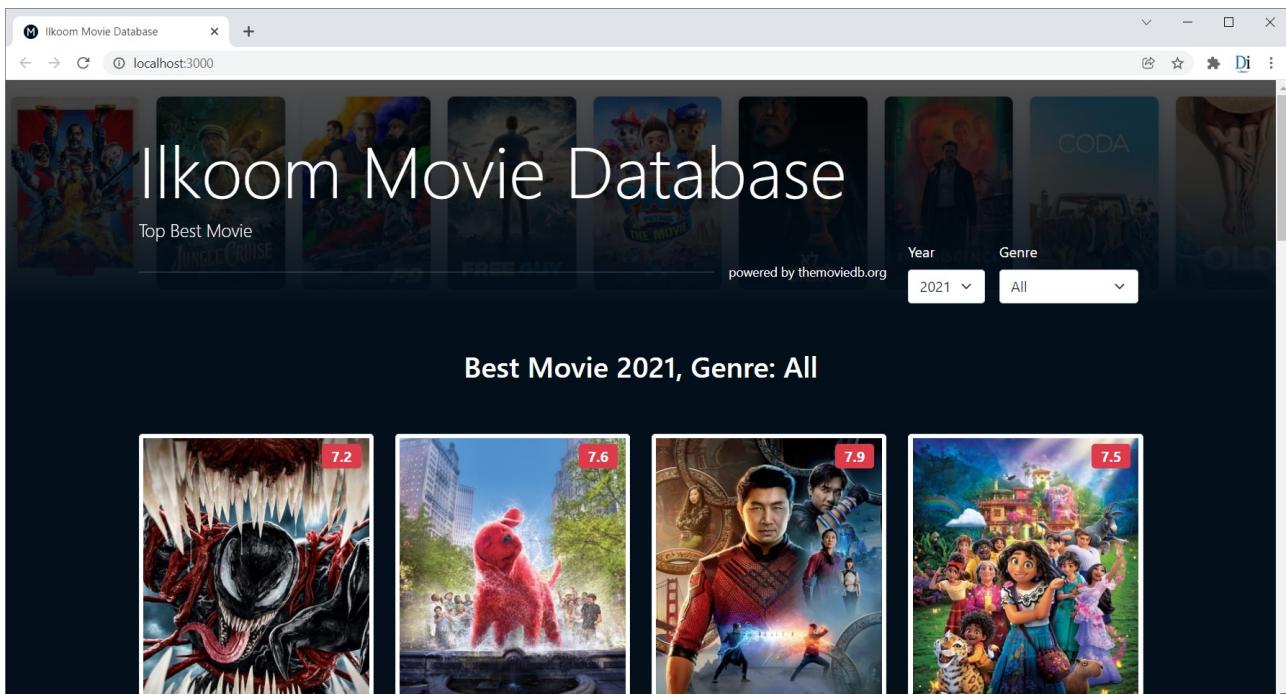
Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

19. Mini Project: Ilkoom Movie Database

Dalam bab sebelumnya kita sudah bahas cara mengambil data API dan memprosesnya di React. Materi tersebut bisa dipakai untuk membuat berbagai aplikasi web yang sangat menarik. Salah satu yang sebentar lagi akan kita rancang adalah **Ilkoom Movie Database**, sebuah aplikasi yang menampilkan daftar film terpopuler.

Web Ilkoom Movie Database mengambil data API dari www.themoviedb.org. Dengan mengolah data yang ada, kita bisa menampilkan daftar film paling populer di setiap tahun. Berikut tampilan akhir dari project ini:



Gambar: Tampilan Ilkoom Movie Database

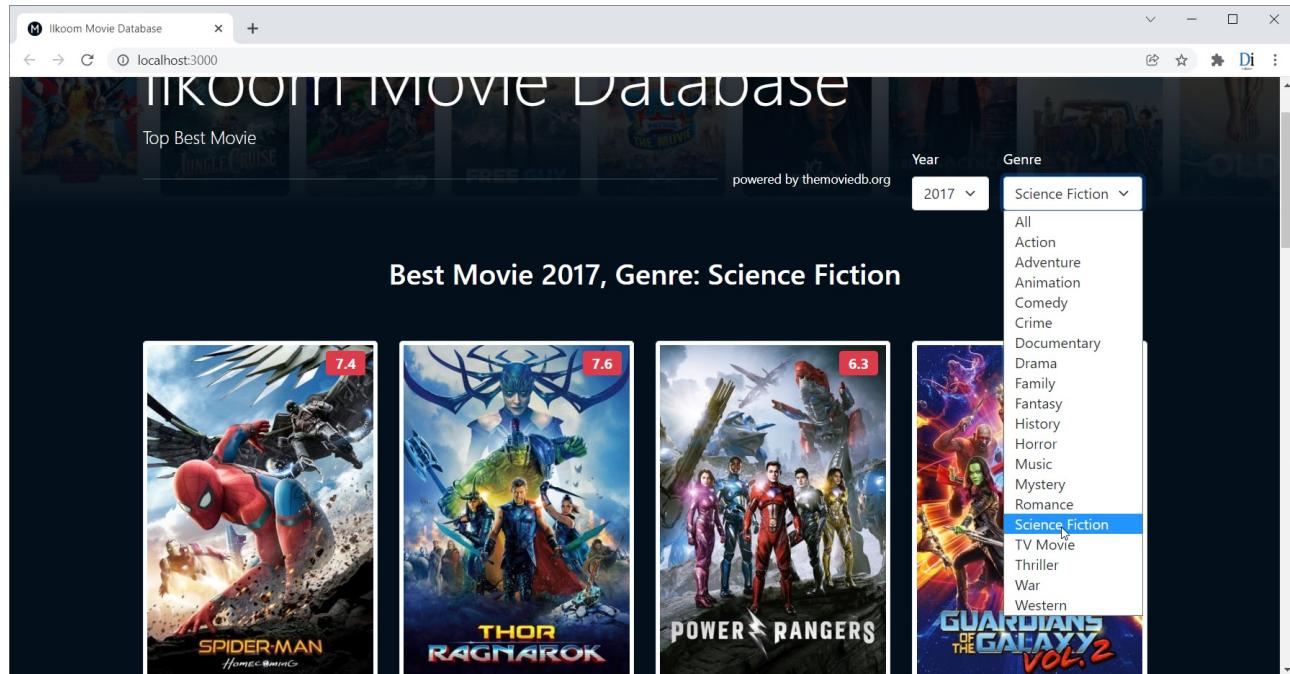
Ketika halaman di-load, akan tampil daftar 20 film terpopuler saat ini. Daftar film akan selalu update mengikuti jadwal rilis film terbaru (sesuai penilaian tim themoviedb.org).

Di sisi kanan atas terdapat menu untuk memilih tahun serta genre film. Jika menu ini dipilih, daftar film juga akan berubah. Untuk data tahun, saya batasi hingga 10 tahun ke belakang, tapi ini bisa diatur dari dalam kode program jika ingin menampilkan angka tahun yang lebih lama.

Untuk genre atau jenis film, tersedia 20 pilihan, mulai dari all (semua genre), *action*, *adventure*, *comedy*, *horror*, hingga *thriller*. Pada saat genre dipilih, halaman web otomatis bertukar dan

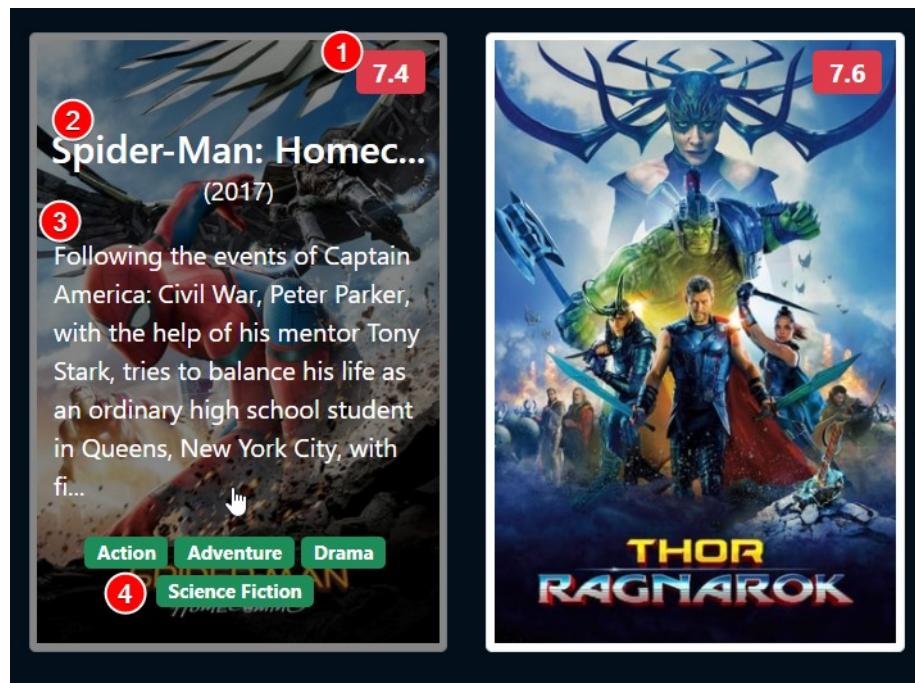
menampilkan 20 film terbaik dari genre tersebut.

Sebagai contoh, berikut tampilan film terbaik tahun 2017 dengan genre "science fiction":



Gambar: Daftar film terbaik tahun 2017, genre "Science Fiction"

Setiap film tampil dengan gambar kecil atau "card". Pada saat di hover (icon mouse berada di atas gambar), akan terlihat info tambahan dari film tersebut:



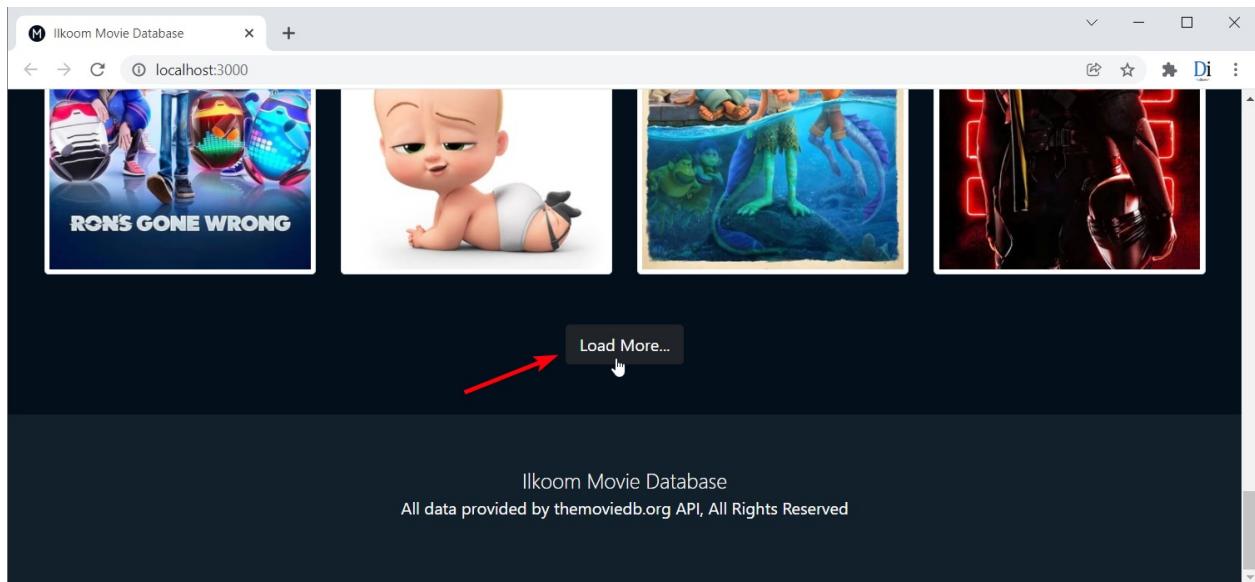
Gambar: Info film tampil saat di hover

Di sisi kanan atas terdapat angka voting dari themoviedb.org (1). Angka tertinggi tidak selalu menjadi film terpopuler karena cukup banyak faktor penilai lain. Angka voting tampil secara

default untuk semua film meskipun tidak di hover. Kemudian terdapat judul film beserta tahun rilis (2), diikuti penjelasan singkat film / sinopsis (3), dan jenis genre dari film tersebut (4).

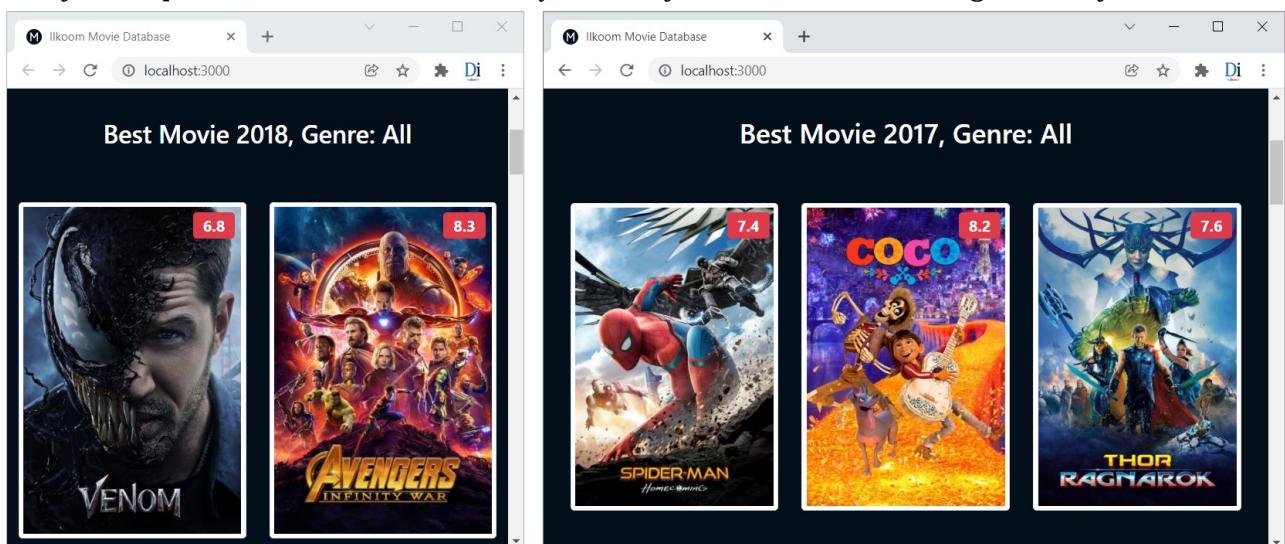
Semua data ini berasal dari API themoviedb.org, dan terserah kita sebagai programmer untuk mengolah dan menyajikan data data tersebut.

Fitur terakhir, terdapat tombol "Load More..." di bagian bawah halaman. Jika tombol ini di klik, akan tampil tambahan 20 film berikutnya sesuai dengan tahun dan genre yang dipilih:



Gambar: Tombol "Load More..." untuk menampilkan tambahan film

Dalam perancangan desain halaman, saya kembali menggunakan Bootstrap agar tampilan menjadi responsive. Ketika di buka di layar kecil, jumlah card baris mengecil menjadi 3 dan 2:



Gambar: Tampilan Ilkoom Movie Database saat lebar web browser diperkecil

19.1. Persiapan Awal

Karena project ini cukup kompleks, kita akan memakai **create react app** dalam perancangan aplikasi. Anda boleh menginstall *create react app* baru atau bisa juga menggunakan file dari mini project sebelumnya (hapus seluruh file di folder `my-app\src`).

Setelah itu edit file `public\index.html` dengan perubahan sebagai berikut:

```
public\index.html

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8" />
6      <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
7      <meta name="viewport" content="width=device-width, initial-scale=1" />
8      <meta name="theme-color" content="#000000" />
9      <meta name="description" content="Ilkoom Movie Database -
10     React Uncover DuniaIlkom" />
11     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
12     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
13     <title>Ilkoom Movie Database</title>
14 </head>
15
16 <body>
17     <noscript>You need to enable JavaScript to run this app.</noscript>
18     <div id="root"></div>
19 </body>
20
21 </html>
```

Perubahan ada di baris 9-10 untuk menukar isi atribut `content`, serta di baris 13 untuk mengganti isi tag `<title>`.

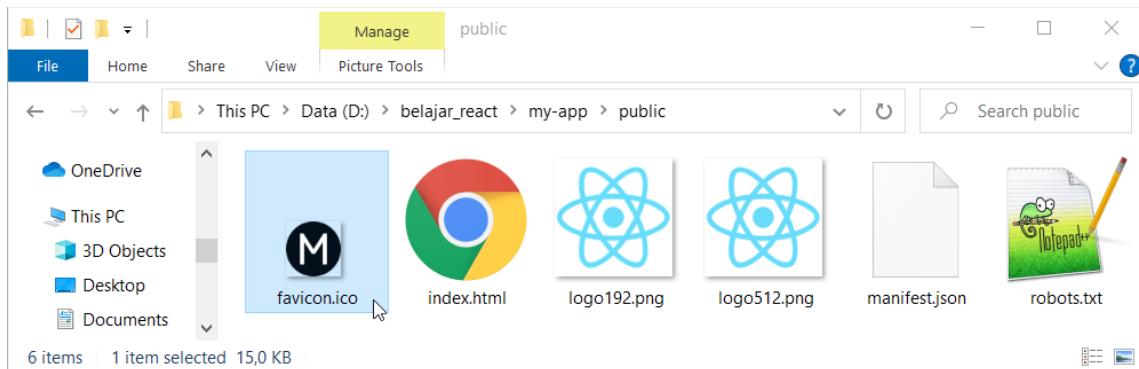
Selanjutnya kita juga butuh framework CSS Bootstrap. Jika belum tersedia, silahkan buka terminal, masuk ke folder **my-app**, lalu jalankan perintah berikut:

```
npm install bootstrap@5
```

Perintah ini akan mendownload file Bootstrap dan menyimpannya ke folder **node_modules**.

Saya juga menyiapkan file `favicon.ico` yang bisa diakses dari file `belajar_react.zip` di Google Drive. Ini tidak wajib, hanya sedikit pemanis saja:

Mini Project: Ilkoom Movie Database



Gambar: File favicon untuk aplikasi Ilkoom Movie Database

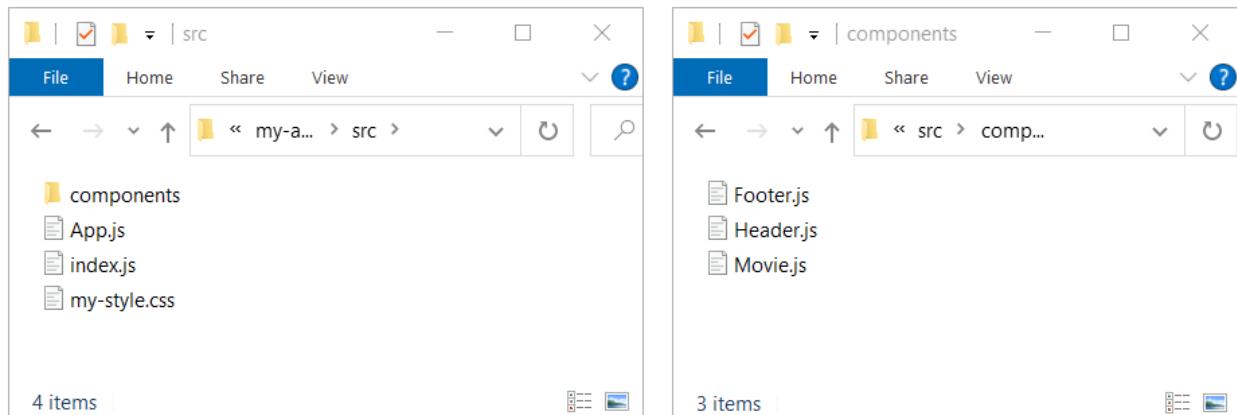
Ke dalam folder `my-app\src`, buat 3 file (isinya kosongkan saja terlebih dahulu):

- `App.js`
- `index.js`
- `my-style.css`

Kemudian buat folder **components** di dalam folder `src`, dan isi dengan 3 file berikut:

- `Movie.js`
- `Header.js`
- `Footer.js`

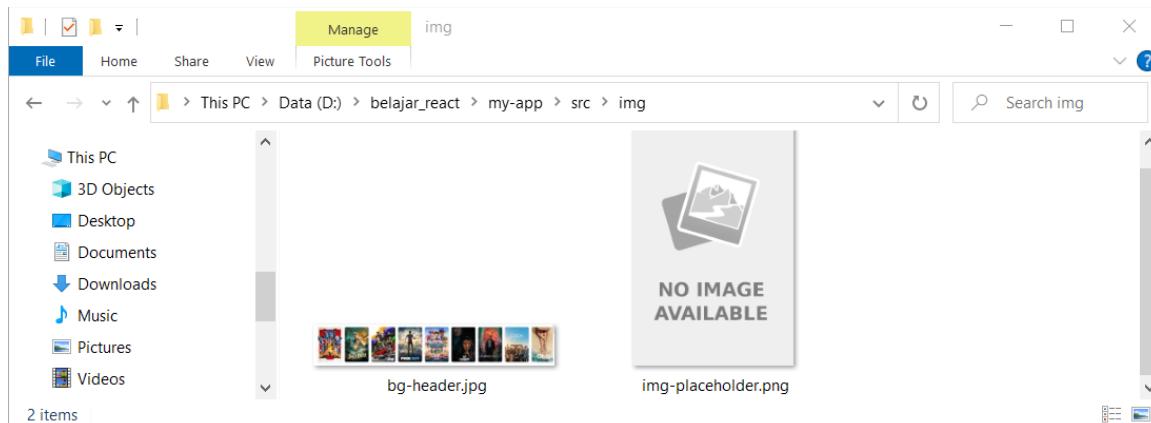
Semua file belum berisi kode apapun, hanya persiapan saja.



Gambar: Isi folder `my-app\src` dan `my-app\src\components`

Sebagai tambahan, buat juga folder **img** di dalam folder `src`. Ke dalam folder `img`, copy file `bg-header.jpg` dan `img-placeholder.png` yang tersedia di file `belajar_react.zip` Google Drive.

Mini Project: Ilkoom Movie Database



Gambar: Copy file header.jpg dan img-placeholder.png ke dalam folder src\img\

File bg-header.jpg akan dipakai sebagai gambar background halaman. Sedangkan file img-placeholder.png disiapkan sebagai gambar pengganti seandainya API dari themoviedb.org tidak menyediakan gambar film.

Setelah itu, buka file src\index.js dan isi dengan kode berikut:

src\index.js

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import 'bootstrap/dist/css/bootstrap.min.css';
5 import './my-style.css';
6
7 ReactDOM.createRoot(document.getElementById('root'))
8 .render(
9   <React.StrictMode>
10   <App />
11 </React.StrictMode>
12 );
```

Kode ini sama seperti di project mini project kita sebelumnya. Lima perintah pertama dipakai untuk proses import library react, react-dom/client, import komponen App, import file bootstrap.min.css, dan import file my-style.css.

Kemudian isi kode berikut ke dalam file my-style.css:

src\my-style.css

```
1 /* Agar container tidak terlalu lebar */
2 @media (min-width: 1200px) {
3   .container {
4     max-width: 1140px !important;
5   }
6 }
7
8 header {
9   background-image: linear-gradient(rgb(0 0 0 / 71%), rgb(2 13 24)),
```

```
10         url("./img/bg-header.jpg");
11     height: 250px;
12     background-position: center;
13     background-repeat: no-repeat;
14     background-size: cover;
15     display: flex;
16     align-items: center;
17     justify-content: center;
18 }
19
20 main {
21   background-color: #020d18;
22 }
23
24 nav {
25   background-color: transparent;
26   margin-top: -70px;
27 }
28
29 footer {
30   background-color:#0f1c27;
31 }
32
33
34 /* ===== */
35 /* movie-container */
36 /* ===== */
37
38 .movie-container {
39   position: relative;
40   cursor: pointer;
41 }
42
43 .movie-container img{
44   transition: filter 0.3s ease-in-out;
45 }
46
47 .movie-container:hover img{
48   filter:brightness(50%);
49 }
50
51 .badge.vote{
52   position: absolute;
53   top: 3%;
54   right: 9%;
55   font-size: 1rem;
56 }
57
58 /* ===== */
59 /* movie-info */
60 /* ===== */
61
62 .movie-info {
63   color: white;
64   position: absolute;
```

```
65   text-align: center;
66   padding: 0 1.5rem;
67   width: 100%;
68   top: 15%;
69   left: 0;
70   opacity: 0;
71   transition: all 0.3s ease-in-out 0s;
72 }
73
74 .movie-container:hover .movie-info{
75   opacity: 1;
76 }
77
78 .movie-info h2{
79   font-size: 1.5rem;
80   margin-bottom: 0;
81 }
82
83 .movie-info .overview {
84   text-align:left;
85   padding: 0 0.2rem;
86 }
```

Kode CSS bagian awal (baris 2-31) berguna untuk mengatur tampilan desain global, header dan footer. Diantaranya menambah gambar background header dengan property `background-image` di baris 9-10. Perhatikan alamat gambar background ditulis sebagai `url("./img/bg-header.jpg")`, ini merujuk ke gambar `bg-header.jpg` yang berada di dalam folder `src/img`.

Kode CSS bagian tengah (baris 38-56) dipakai untuk membuat efek gambar film yang sedikit menghitam saat di hover. Ini diperlukan supaya teks info yang tampil di atas gambar bisa lebih mudah terbaca. Efek kehitaman didapat dari property `filter:brightness(50%)`.

Kode CSS bagian akhir (baris 62-86) berfungsi untuk menampilkan teks info di atas gambar film. Idenya, sembunyikan teks info dengan men-set nilai `opacity` menjadi 0. Barulah saat gambar di hover, set lagi nilai `opacity` menjadi 1. Selector yang dipakai untuk proses ini adalah `.movie-container:hover .movie-info` seperti di baris 74.

Jika ingin mempelajari lebih detail maksud setiap kode CSS yang ada, bisa menunggu sampai kita masuk ke struktur kode JSX. Nanti baru terlihat lebih jelas fungsi dari setiap selector CSS dan pasangan tag HTML-nya.

19.2. Komponen Header dan Footer

Project **Ilkoom Movie Database** saya pecah menjadi beberapa komponen. Diantaranya terdapat 2 komponen yang hanya berisi kode JSX untuk tampilan saja. Komponen yang dimaksud adalah **Header** dan **Footer**. Sesuai namanya, kedua komponen bertanggung jawab untuk menampilkan bagian atas (*header*) dan bagian bawah (*footer*) halaman.

Mini Project: Ilkoom Movie Database

Silahkan buka file components\Header.js dan isi dengan kode berikut:

src\components\Header.js

```
1 const Header = () => {
2
3     return (
4         <header className="text-white py-5">
5             <div className="container">
6                 <div className="row">
7                     <div className="col-12">
8                         <h1 className="display-1">Ilkoom Movie Database</h1>
9                         <h3 className="lead mb-0 d-none d-md-block">Top Best Movie</h3>
10                    </div>
11                </div>
12            </div>
13        </header>
14    )
15 }
16
17 export default Header;
```

Kemudian buka juga file components\Footer.js dan isi dengan kode berikut:

src\components\Footer.js

```
1 const Footer = () => {
2
3     return (
4         <footer id="main-footer" className="text-white py-5">
5             <div className="container">
6                 <div className="row">
7                     <div className="col text-center">
8                         <p className="lead mb-0">Ilkoom Movie Database</p>
9                         <p>All data provided by themoviedb.org API, All Rights Reserved</p>
10                    </div>
11                </div>
12            </div>
13        </footer>
14    )
15 }
16
17 export default Footer;
```

Kedua komponen hanya berisi kode JSX dengan berbagai tag HTML. Agar hasilnya bisa terlihat, silahkan edit file App.js dan isi dengan kode berikut:

src\App.js

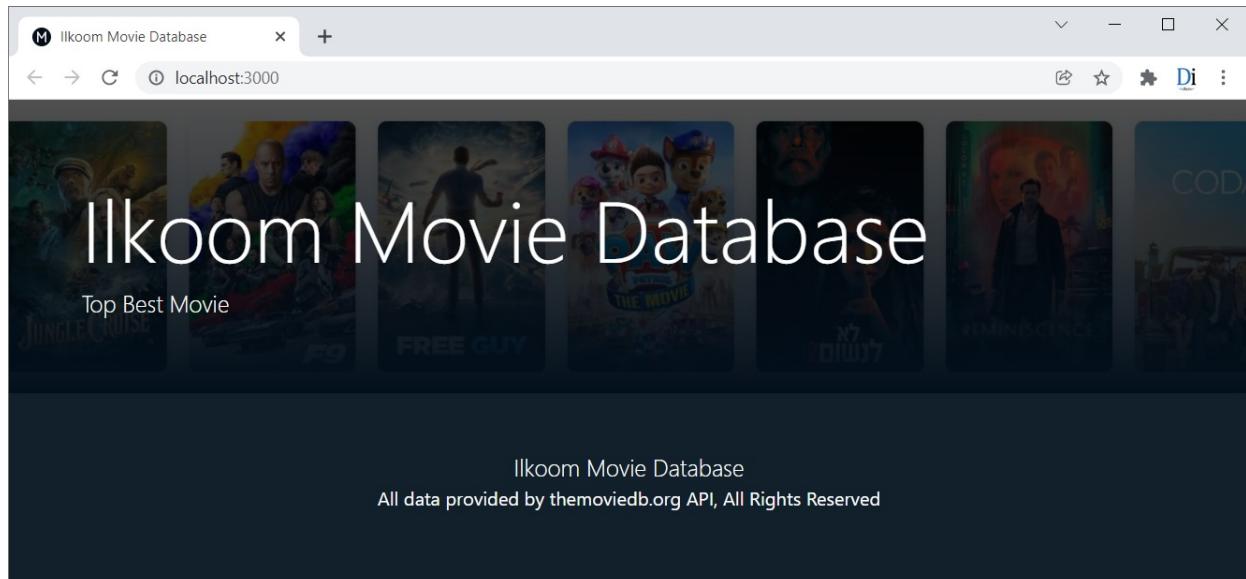
```
1 import React from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4
5 const App = () => {
```

```
6  return (
7    <React.Fragment>
8      <Header />
9      <Footer />
10     </React.Fragment>
11   )
12 }
13
14 export default App;
```

Komponen App menjadi komponen utama dari aplikasi kita. Komponen ini akan di update secara berkala seiring pembahasan materi.

Untuk saat ini komponen App masih sangat dasar. Komponen Header dan Footer di-import di baris 2-3, lalu ditampilkan ke dalam kode JSX di baris 8-9. Tag `<React.Fragment>` dipakai sebagai container dari kedua komponen.

Save file di atas, jalankan `npm start` dan cek hasilnya di web browser:



Gambar: Tampilan bagian header dan footer Ilkoom Expense Tracker

Efek visual yang terlihat berasal dari beberapa class Bootstrap di dalam kode JSX serta tambahan kode CSS di file `my-style.css`.

19.3. Mendapatkan Hak Akses API

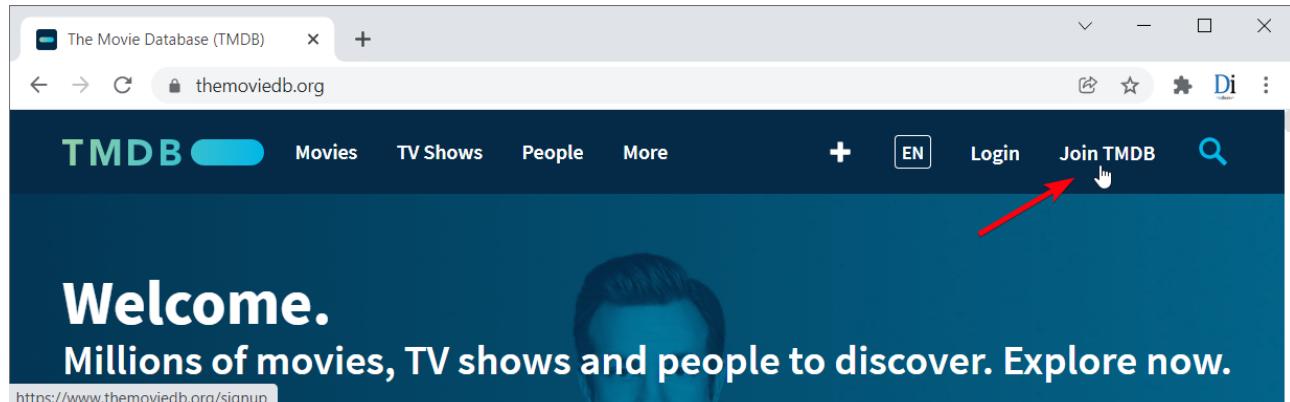
Aplikasi Ilkoom Movie Database perlu mengambil data API dari **themoviedb.org**. Berbeda dengan web **reqres.in** yang APInya bisa langsung kita akses, web themoviedb.org mewajibkan setiap user mendaftar terlebih dahulu agar mendapatkan API key.

Mayoritas web penyedia data API mewajibkan *key* sebagai penanda identitas dari setiap aplikasi. Dengan cara ini, semua request akan tercatat dan web data API bisa saja

membatasi hak akses jika sudah melebihi limit tertentu. Jika kita tetap ingin mengakses data API, maka harus ambil paket berbayar.

Untungnya, di tahun 2019 web themoviedb.org tidak lagi membatasi jumlah API request²⁷, sehingga anda bisa saja menggunakan key yang saya daftarkan. Namun tetap disarankan mendaftar sendiri agar lebih paham cara penggunaan API dari themoviedb.org.

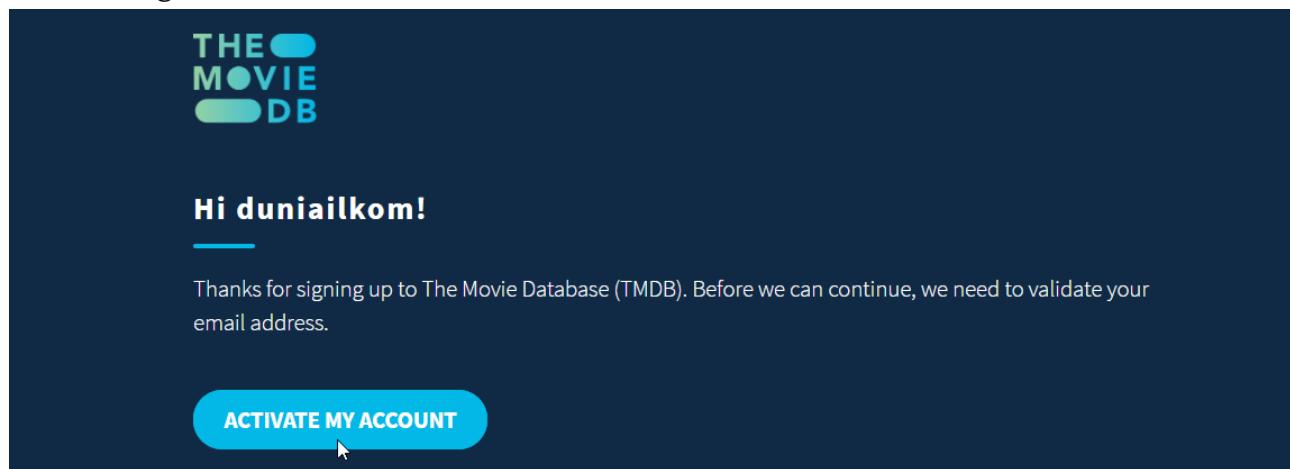
Untuk mendaftar, silahkan buka <https://www.themoviedb.org>, lalu klik menu "Join TMDB".



Gambar: Link untuk proses register themoviedb.org

Di halaman register, isi data username, password dan alamat email, kemudian klik tombol "Sign Up".

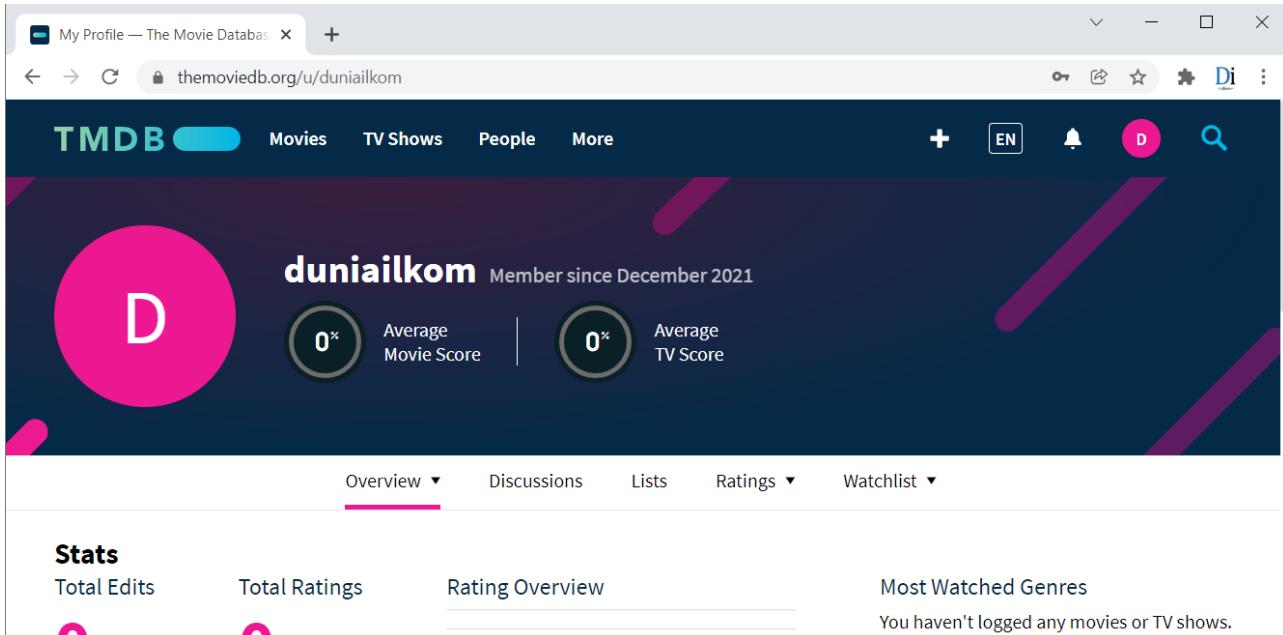
Selanjutnya cek email dan cari link aktivasi dari themoviedb.org. Klik link yang disertakan untuk mengaktifkan akun:



Gambar: Cek email untuk proses konfirmasi

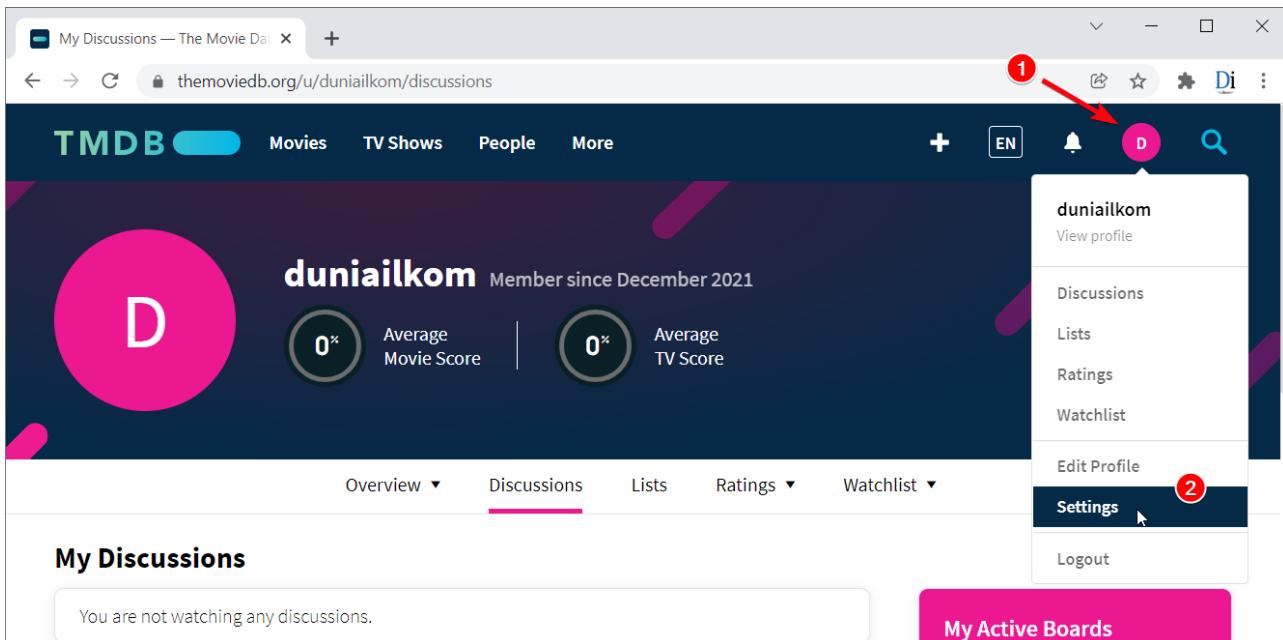
Setelah itu kita sudah bisa login dan masuk halaman dashboard themoviedb.org:

27. <https://developers.themoviedb.org/3/getting-started/request-rate-limiting>



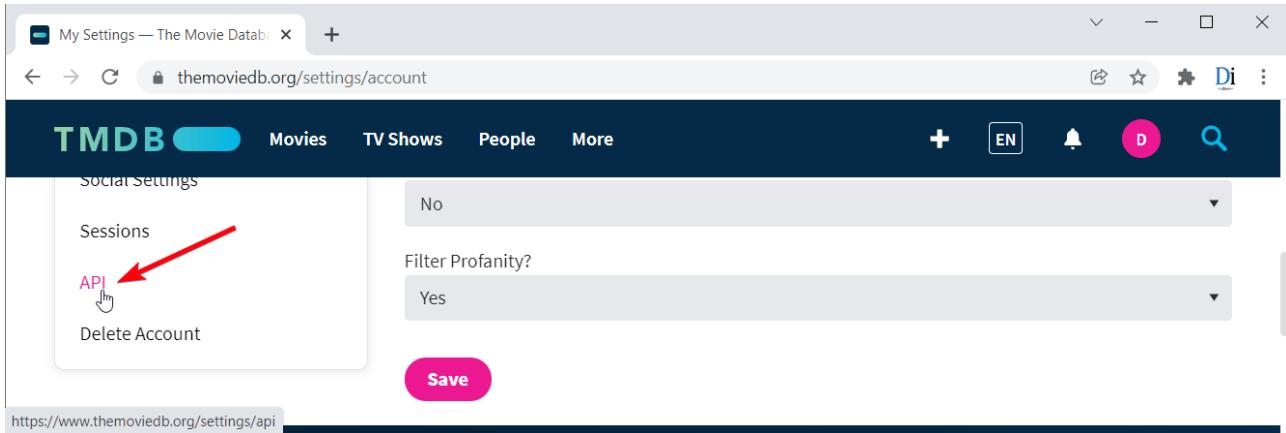
Gambar: Halaman dashboard themoviedb.org

Web themoviedb.org sebenarnya sebuah portal komunitas film, dimana kita bisa bergabung ke forum diskusi, membuat resensi film, men-voting file kesukaan, dsb. Namun tujuan kita adalah mengakses API, untuk itu silahkan klik icon inisial huruf di kanan atas (1), lalu pilih menu **Settings** (2).



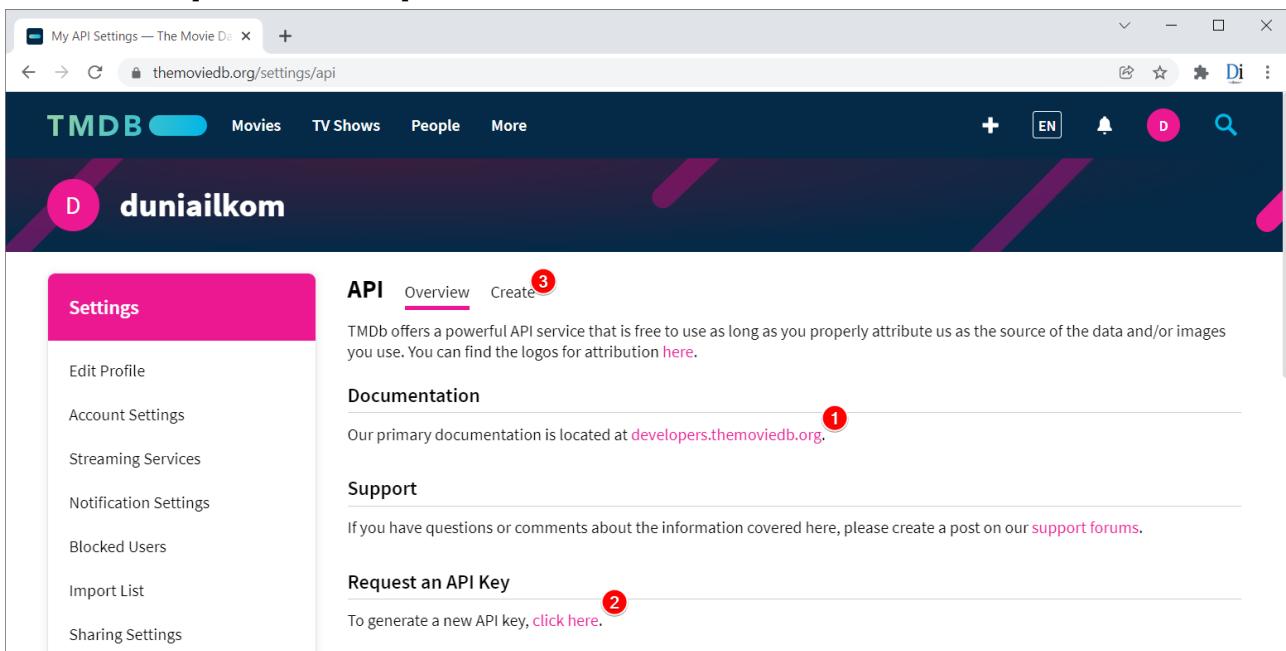
Gambar: Masuk ke menu settings themoviedb.org

Di halaman settings, lanjut klik menu API di kiri bawah:



Gambar: Klik menu API di halaman settings themoviedb.org

Dan akan tampil menu API seperti halaman berikut:

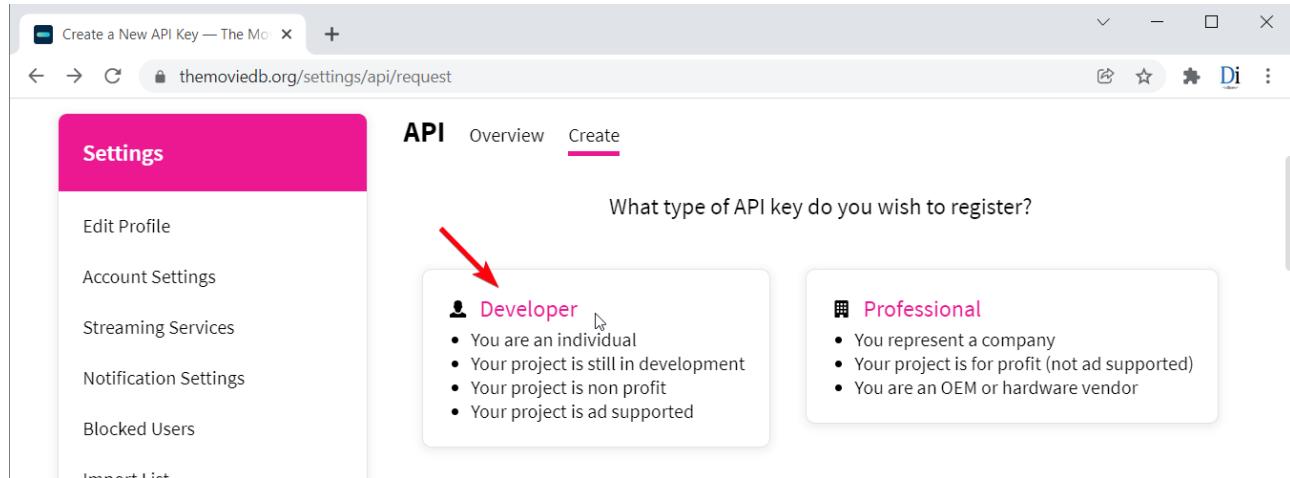


Gambar: Halaman API themoviedb.org

Halaman API ini hanya berisi informasi singkat. Yang akan sering kita akses nanti adalah halaman dokumentasi API yang berada di <https://developers.themoviedb.org/3>, linknya juga tersedia di bagian (1). Untuk sementara, link ini kita lewati terlebih dahulu

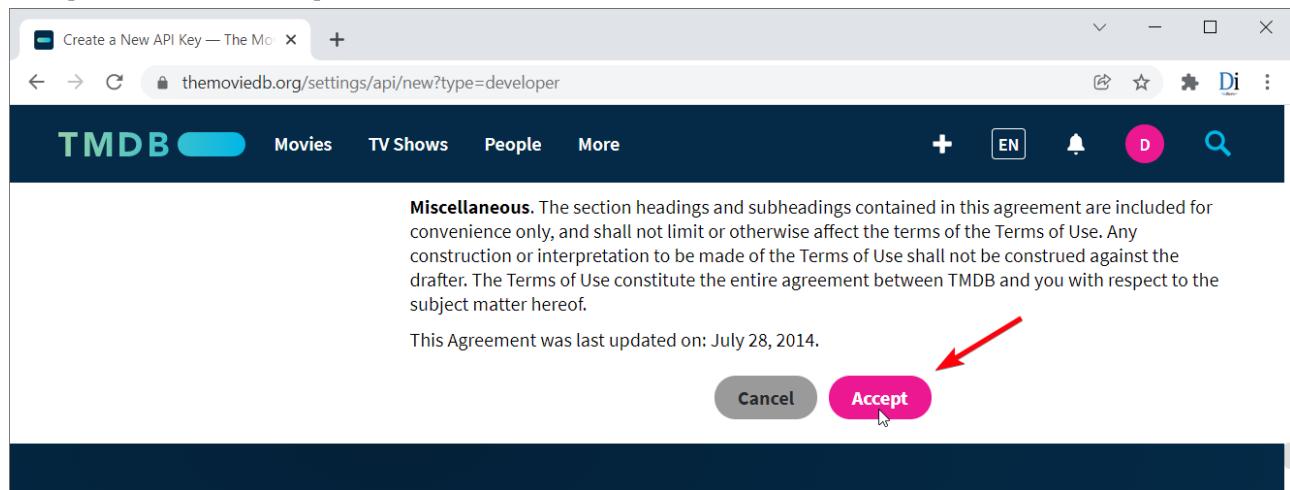
Untuk membuat key API, silahkan klik link yang ada di bagian "Request an API Key" (2), atau bisa juga klik tab "Create" (3).

Di halaman "Create", tersedia 2 pilihan, apakah kita ingin menggunakan API untuk "Developer" atau "Professional". Pilih saja **Developer**:



Gambar: Memilih tipe API themoviedb.org

Setelah itu akan tampil halaman "Approve Terms of Use". Langsung saja scroll hingga ke bawah dan pilih tombol "Accept":



Gambar: Klik tombol Accept untuk menyetujui terms of use

Di halaman berikutnya, kita diminta mengisi form terkait aplikasi yang akan mengakses API. Data ini hanya untuk administrasi saja. Sebagai contoh, saya mengisinya dengan data berikut:

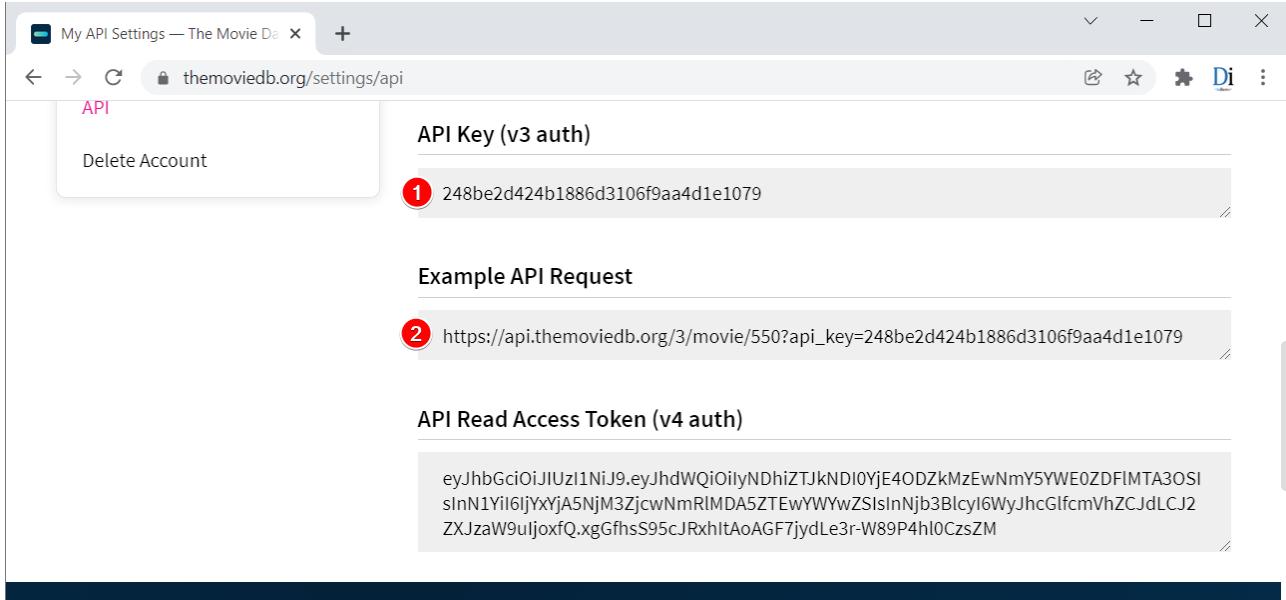
The screenshot shows a web form for requesting an API key. At the top, there are tabs for 'API', 'Overview', and 'Create'. The 'API' tab is selected. Below the tabs, there is a dropdown menu for 'Type of Use' set to 'Website'. The 'Application Name' field contains 'Ilkoom Movie Database'. The 'Application URL' field contains 'http://localhost:3000'. The 'Application Summary' field contains the text: 'Ilkoom Movie Database is an exercise project to learn React API'. Below this summary, there are two rows of input fields. The first row contains 'First Name' (Ilkoom) and 'Last Name' (Movie Database). The second row contains 'Email Address' (duniailkom01@gmail.com) and 'Phone Number (no parenthesis or dashes)' (62858422564523). The third row contains 'Address 1' (Indonesia) and 'Address 2' (empty). The fourth row contains 'City' (Jakarta) and 'State' (Indonesia). The fifth row contains 'Zip Code' (10997) and a 'Country' dropdown menu set to 'Indonesia'. At the bottom of the form is a pink 'Submit' button.

Gambar: Isi form permintaan API key dari themoviedb.org

Untuk inputan "Application Name", bisa dibedakan sedikit dengan mengubah nama aplikasi seperti *Rudi Movie Database*. Untuk "Application Summary" juga bisa diisi: *Rudi Movie Database is an exercise project to learn React API*.

Untuk inputan pribadi seperti *first name*, *address*, *phone number* dst, bisa diisi dengan data dummy, atau tidak masalah juga jika anda ingin mengisinya dengan data asli. Lalu akhiri dengan klik tombol "Submit".

Setelah itu kita kembali dibawa ke halaman settingan API. Scroll sedikit ke bawah dan API key dari themoviedb.org sudah tersedia:



Gambar: API key dari themoviedb.org sudah tersedia

Terdapat 2 jenis API yang disediakan themoviedb.org, yakni v3 dan v4. Untuk project kita, saya akan memakai v3 saja agar lebih praktis (1).

Perhatikan kolom "Example API Request" (2). Itu adalah contoh cara penulisan API key, dimana key yang kita dapat perlu ditulis sebagai *query string*. Dalam contoh ini, API yang saya dapat adalah:

248be2d424b1886d3106f9aa4d1e1079

Sehingga ketika mengakses API, perlu disambung menjadi:

https://api.themoviedb.org/3/movie/550?api_key=248be2d424b1886d3106f9aa4d1e1079

Pastinya nilai key ini akan berbeda untuk setiap user.

Karena pihak themoviedb.org tidak memberikan batas limit API, API key di atas bisa saja anda pakai, khususnya bagi yang tidak ingin repot registrasi API ke themoviedb.org.

19.4. Mengakses Data API

Agar bisa mengakses data API dari themoviedb.org, kita harus pelajari bagaimana cara penulisan URL serta data apa saja data yang bisa diakses. Informasi mengenai hal ini tersedia di dokumentasi API themoviedb.org di alamat <https://developers.themoviedb.org/3/>:

The screenshot shows the homepage of the Movie Database API documentation. The URL is <https://developers.themoviedb.org/3/getting-started/introduction>. The page features the TMDB logo and navigation links for OAS, RAML, and Support. On the left, there's a sidebar with sections like 'Select a different version', 'Filter sections...', 'GETTING STARTED' (with 'Introduction' highlighted), 'Authentication', 'Daily File Exports', 'Languages', 'Images', 'Image Languages', and 'Regions'. The main content area is titled 'Getting Started' and 'Introduction'. It welcomes users to version 3 of the API and provides instructions for registering an API key.

Halaman awal dokumentasi API dari themoviedb.org

Bisa membaca dokumentasi merupakan salah satu skill dasar yang harus dikuasai bagi kita sebagai programmer. Di kasus ideal, dokumentasi API sudah tersaji rapi beserta contoh-contoh URL, namun ini tidak selalu. Jika mendapati web yang pelit dokumentasi, itu bisa jadi perjuangan tersendiri.

Dokumentasi API dari themoviedb.org juga tidak terlalu jelas, apalagi karena data yang tersedia sangat banyak, penjelasannya juga lumayan panjang. Mengingat keterbatasan tempat, kita akan bahas 1 fitur saja. Jika berminat, anda bisa explore lebih lanjut.

Dari menu di sisi kiri, silahkan klik menu "DISCOVER":

The screenshot shows the 'Discover' documentation page. The URL is <https://developers.themoviedb.org/3/discover/movie-discover>. The left sidebar lists 'COLLECTIONS' (Companies, Configuration, Credits, Discover, Find, Genres, Guest Sessions) with 'Discover' highlighted. A red arrow points to the 'Discover' link. The main content area contains notes about using the 'certification' parameter, details about the 'region' parameter, and examples of filter syntax. It also links to examples of what can be done with the discover feature.

Gambar: Pilih menu Movies di sisi kanan

Discover adalah fitur untuk mencari daftar film berdasarkan tingkat popularitas, rata-rata rating, jumlah vote, genre dsb. Dalam praktek ini, saya akan memakai tingkat popularitas agar daftar film kita selalu update dengan film-film terbaru. Jika menggunakan aspek lain seperti

rating, film lama bisa muncul di hasil teratas.

Dokumentasi API themoviedb.org lebih banyak menyebut film dengan istilah "movie". Sehingga dalam pembahasan nanti saya juga banyak menggunakan kata "movie". Dari segi pengertian, keduanya merujuk ke maksud yang sama.

Di halaman Discover, scroll sedikit ke bawah sampai ke bagian tab "Definition" (1):



The screenshot shows the 'Discover' endpoint documentation. The 'Definition' tab is highlighted with a red circle containing the number 1. Below it, there's a 'Try it out' button. The main content area is titled 'Authentication' and includes an 'API Key' checkbox. The 'Query String' section (2) is expanded, showing a table of properties:

Property	Type	Description	Requirement
api_key	string	default: <>api_key>>	required
language	string	Specify a language to query translatable fields with. minLength: 2 pattern: ([a-z]{2})-([A-Z]{2}) default: en-US	optional
region	string	Specify a ISO 3166-1 code to filter release dates. Must be uppercase. pattern: ^[A-Z]{2}\$	optional
sort_by	string	Choose from one of the many available sort options. Allowed Values: , popularity.asc, popularity.desc, release_date.asc, release_date.desc, revenue.asc, revenue.desc, primary_release_date.asc, primary_release_date.desc, original_title.asc, original_title.desc, vote_average.asc, vote_average.desc, vote_count.asc, vote_count.desc default: popularity.desc	optional
certification_country	string	Used in conjunction with the certification filter, use this to specify a country with a valid certification.	optional

Below the table, there's a link "...show 32 more properties" (3).

Gambar: Penjelasan dari setiap query string

Di dalam tab Definition, terdapat penjelasan dari semua query string yang bisa kita tambah (2). Misalnya di baris pertama terdapat `api_key` yang wajib disertakan (`required`).

Lalu di baris kedua terdapat `language` jika kita ingin men-translate penjelasan movie ke dalam bahasa tertentu. Di kolom tengah terdapat format penulisan, yakni dengan pola `([a-z]{2})-([A-Z]{2})`. Pilihan `language` ini opsional, yang jika tidak disertakan, nilai default adalah `en-US`.

Di baris selanjutnya terdapat `region`, jika kita ingin memilih movie yang dirilis pada daerah tertentu, lalu diikuti dengan `sort_by` dimana kita bisa mengatur urutan hasil berdasarkan berbagai kriteria. Secara default movie akan disusun menurun berdasarkan popularitas, yakni nilai `popularity.desc`.

Selain 4 query ini, terdapat 32 query string lain yang bisa kita atur. Yup, jumlah yang sangat banyak, daftar lengkapnya bisa terlihat dengan men-klik link "...show 32 more properties" (3).

Untuk melihat contoh penulisan URL, silahkan klik tab "Try it out" (1):

The screenshot shows the 'Try it out' tab selected (1). Below it, there's a section for 'Variables' (2) where 'api_key' is set to '248be2d424b1886d3106f9aa4d1e1079'. Then, under 'Query String' (3), there's a table with the following parameters:

Parameter	Value	Type	Required
api_key	248be2d424b1886d3106f9aa4d1e1079		required
language	en-US		optional
region	String		optional
sort_by	popularity.desc		optional
certification_country	String		optional

Gambar: Input query string URL

Di tab ini, terdapat form inputan untuk semua query string. Di bagian "Variables", silahkan isi API key yang kita dapat sebelumnya (2). Jika anda tidak mendaftar, boleh diisi dengan API key berikut:

248be2d424b1886d3106f9aa4d1e1079

Di bagian "Query String" (3), kita bisa mengisi dan memilih nilai awal untuk semua query. Sementara ini, biarkan saja di pilihan default.

Scroll ke baris paling bawah, dan akan terlihat URL yang diperlukan:

The screenshot shows the URL being constructed. On the left, there are two parameters: 'with_watch_monetization_types' set to 'flatrate' (optional) and 'without_companies' set to 'String' (optional). On the right, a button labeled 'SEND REQUEST' (1) is shown. Below the parameters, the resulting URL is displayed: https://api.themoviedb.org/3/discover/movie?api_key=248be2d424b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate. A red arrow points from the 'SEND REQUEST' button to the URL.

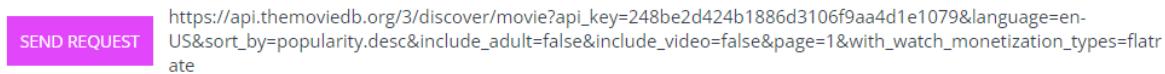
Gambar: Contoh penulisan URL API dari themoviedb.org

Dalam contoh ini, URL yang saya dapat adalah:

[https://api.themoviedb.org/3/discover/movie?
api_key=248be2d424b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate](https://api.themoviedb.org/3/discover/movie?api_key=248be2d424b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate)

Di sisi kanan, terdapat tombol "Send Request" (1) yang bisa di klik untuk melihat hasil dari URL

ini. Akan tetapi dari beberapa kali percobaan, hasilnya selalu error:



Response 503 Service Unavailable.

Body 0 Headers 0 Cookies
Pretty Raw

```
1 Error: Network Error
2
3 Check the developer tools console, it might have more information on the error.
4
5 If you are using an Ad blocker, it is possible your Ad blocker is blocking the request.
```

Gambar: Error saat mencoba URL API

Error ini bukanlah karena URL API yang salah, akan tetapi mungkin sedikit bug dari fitur testing milik themoviedb.org.

Sebagai alternatif, silahkan copy seluruh URL tersebut dan input ke web debugging API yang pernah kita coba di bab sebelumnya, yakni [hoppscotch.io](#):

HOPPSOTCH Open source API [+](#) hoppsotch.io Star 34.375 Save My Workspace Login

GET https://api.themoviedb.org/3/discover/movie?api_key=248be2d424b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate Send Save

Parameters Body Headers Authorization Pre-request Script Tests

Query Parameters Parameter 1 Value 1

Status: 200 Time: 818 ms Size: 12155 B

JSON Raw Headers Test Results

Response Body

```
1 {
2   "page": 1,
3   "results": [
4     {
5       "adult": false,
6       "backdrop_path": "/lNyLSOKMMeUPr1RsL4KcRuIXwHt.jpg",
7       "genre_ids": [
8         878,
9         ...
10      ]
11    }
12  ]
13}
```

Gambar: Testing URL API themoviedb.org di hoppscotch.io

Sip, data API sudah berhasil di akses dan tampil di bagian bawah. Berikut hasil yang saya dapat dengan sedikit modifikasi agar tidak terlalu panjang:

```
1 {
2   "page": 1,
```

```

3   "results": [
4     {
5       "adult": false,
6       "backdrop_path": "/lNyLSOKMMeUPr1RsL4KcRuIXwHt.jpg",
7       "genre_ids": [
8         878,
9         28,
10        12
11      ],
12      "id": 580489,
13      "original_language": "en",
14      "original_title": "Venom: Let There Be Carnage",
15      "overview": "After finding a host body in investigative reporter Eddie
16                  Brock, the alien symbiote must face a new enemy, Carnage,
17                  the alter ego of serial killer Cletus Kasady.",
18      "popularity": 7615.279,
19      "poster_path": "/rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg",
20      "release_date": "2021-09-30",
21      "title": "Venom: Let There Be Carnage",
22      "video": false,
23      "vote_average": 7.2,
24      "vote_count": 4236
25    },
26    {
27      "adult": false,
28      "backdrop_path": "/5uVhMGsps81CN0S4U9NF0Z4tytG.jpg",
29      "genre_ids": [
30        28,
31        35,
32        80,
33        53
34      ],
35      "id": 512195,
36      "original_language": "en",
37      "original_title": "Red Notice",
38      "overview": "An Interpol-issued Red Notice is a global alert to hunt
39                  and capture the world's most wanted. But when a daring
40                  heist brings together the FBI's top profiler and two rival
41                  criminals, there's no telling what will happen.",
42      "popularity": 3574.397,
43      "poster_path": "/wdE6ewaKZHr62bLqCn7A2DiGShm.jpg",
44      "release_date": "2021-11-04",
45      "title": "Red Notice",
46      "video": false,
47      "vote_average": 6.8,
48      "vote_count": 1886
49    },
50    {
51      "adult": false,
52      "backdrop_path": "/mFbS5TwN95BcSEfiztdchLgTQ0v.jpg",
53      "genre_ids": [
54        28,
55        18,
56        36
57      ],

```

```

58     "id": 617653,
59     "original_language": "en",
60     "original_title": "The Last Duel",
61     "overview": "King Charles VI declares that Knight Jean de Carrouges
62             settle his dispute with his squire, Jacques Le Gris, by
63             challenging him to a duel.",
64     "popularity": 2972.977,
65     "poster_path": "/zjrJE0fpzPvX8saJXj8VNfcjBoU.jpg",
66     "release_date": "2021-10-13",
67     "title": "The Last Duel",
68     "video": false,
69     "vote_average": 7.6,
70     "vote_count": 778
71   },
72   ...
73   ...
74 ],
75   "total_pages": 500,
76   "total_results": 10000
77 }

```

Data JSON yang diperoleh berbentuk *nested object* (object yang di dalamnya juga terdapat object). Object terluar terdiri dari 5 property: `page`, `results`, `total_pages` dan `total_results`. Berikut penjelasannya:

- `page`: posisi halaman saat ini.
- `results`: berisi 20 daftar movie dalam bentuk array object.
- `total_pages`: total halaman dari hasil discover.
- `total_results`: total jumlah movie untuk hasil discover.

Secara default, [themoviedb.org](https://www.themoviedb.org) akan mengirim data untuk 20 movie di setiap halaman. Jika kita ingin minta halaman 2, maka bisa menambah query string "`&page=2`". Pilihan `page` ini ada di dalam pengaturan saat pengambilan data URL:

<code>include_adult</code>	<code>false</code>	optional
<code>include_video</code>	<code>false</code>	optional
<code>page</code>	2	optional
<code>primary_release_year</code>	Integer	optional

Gambar: Mengubah query string `page` menjadi 2

Pemecahan data menjadi halaman berguna agar data tidak terlalu panjang. Selain itu ini sangat berguna untuk membuat fitur *pagination*, dimana user bisa men-klik tombol next jika ingin melihat 20 movie selanjutnya.

Data untuk 20 movie tersebut ada di dalam property `result` yang tersusun dalam bentuk array. Ini sangat memudahkan kita karena nanti tinggal di looping dalam perulangan `.map()` untuk mengakses setiap movie. Sebagai contoh, berikut data untuk 1 object movie:

```
1  {
2      "adult": false,
3      "backdrop_path": "/lNyLSOKMMeUPr1RsL4KcRuIXwHt.jpg",
4      "genre_ids": [
5          878,
6          28,
7          12
8      ],
9      "id": 580489,
10     "original_language": "en",
11     "original_title": "Venom: Let There Be Carnage",
12     "overview": "After finding a host body in investigative reporter Eddie
13             Brock, the alien symbiote must face a new enemy, Carnage,
14             the alter ego of serial killer Cletus Kasady.",
15     "popularity": 7615.279,
16     "poster_path": "/rjkmN1dniUHVYAtwuV3Tji7FsDO.jpg",
17     "release_date": "2021-09-30",
18     "title": "Venom: Let There Be Carnage",
19     "video": false,
20     "vote_average": 7.2,
21     "vote_count": 4236
22 }
```

Berikut penjelasan singkat dari setiap property:

- **adult**: apakah termasuk kategori movie dewasa atau bukan (*false*, berarti bukan).
- **backdrop_path**: *path* untuk file gambar backdrop (gambar latar belakang).
- **genre_ids**: kumpulan id dari genre movie.
- **id**: nomor id dari movie ini.
- **original_language**: bahasa asli movie.
- **original_title**: judul asli movie.
- **overview**: resensi singkat movie.
- **popularity**: nilai popularitas movie berdasarkan perhitungan internal themoviedb.org.
- **poster_path**: *path* dari file gambar utama movie.
- **release_date**: tanggal movie di rilis.
- **title**: judul movie (misalnya jika sudah diterjemahkan).
- **video**: apakah termasuk kategori video yang bukan movie (seperti wawancara tv, dsb)
- **vote_average**: angka voting rata-rata movie.
- **vote_count**: jumlah user yang memberikan voting untuk movie.

Untuk aplikasi **Ilkoom Movie Database**, saya tidak memakai semua data ini, hanya sebagian saja, yaitu: **genre_ids**, **overview**, **poster_path**, **release_date**, **title**, dan **vote_average**. Kita akan bahas beberapa nilai yang perlu penjelasan lebih lanjut.

Nilai **genre_ids** berisi angka id yang merujuk ke **id** genre movie. Setelah penelusuran lebih lanjut, **id** setiap genre bisa di dapat dari menu "GENRES" (1):



Gambar: URL API untuk melihat id genre

Ternyata id genre berada di dalam data API. Berikut alamat yang harus diakses setelah memasukkan API key (2):

```
https://api.themoviedb.org/3/genre/movie/list?
api_key=248be2d424b1886d3106f9aa4d1e1079&language=en-US
```

Data JSON dari URL di atas bisa kita lihat menggunakan hoppscotch.io. Berikut hasilnya dengan sedikit modifikasi agar lebih singkat:

```
1  {
2      "genres": [
3          { "id": 28, "name": "Action" },
4          { "id": 12, "name": "Adventure" },
5          { "id": 16, "name": "Animation" },
6          { "id": 35, "name": "Comedy" },
7          { "id": 80, "name": "Crime" },
8          { "id": 99, "name": "Documentary" },
9          { "id": 18, "name": "Drama" },
10         { "id": 10751, "name": "Family" },
11         { "id": 14, "name": "Fantasy" },
12         { "id": 36, "name": "History" },
13         { "id": 27, "name": "Horror" },
14         { "id": 10402, "name": "Music" },
15         { "id": 9648, "name": "Mystery" },
16         { "id": 10749, "name": "Romance" },
17         { "id": 878, "name": "Science Fiction" },
18         { "id": 10770, "name": "TV Movie" },
19         { "id": 53, "name": "Thriller" },
20         { "id": 10752, "name": "War" },
21         { "id": 37, "name": "Western" }
22     ]
23 }
```

Sebagai contoh, untuk movie "Venom: Let There Be Carnage", property `genre_ids` berisi array [878, 28, 12], maka genre untuk movie itu adalah ["Science Fiction", "Action", "Adventure"]. Nantinya kita perlu memikirkan cara membuat kode program untuk proses konversi ini.

Nilai lain yang perlu dibahas adalah `poster_path`. Property `poster_path` berisi `path` gambar, akan tetapi isinya hanya potongan `path` seperti "/rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg". Apa

awalan yang harus ditambah? Penjelasan tersebut ada di menu "Image" (1):

Getting Started

Images

You'll notice that movie, TV and person objects contain references to different file paths. In order to generate a fully working image URL, you'll need 3 pieces of data. Those pieces are a `base_url`, a `file_size` and a `file_path`.

The first two pieces can be retrieved by calling the `/configuration` API and the third is the file path you're wishing to grab on a particular media object. Here's what a full image URL looks like if the `poster_path` of `/kqjL17yufvn90VLyXYpvtyrFFak.jpg` was returned for a movie, and you were looking for the `w500` size:

`https://image.tmdb.org/t/p/w500/kqjL17yufvn90VLyXYpvtyrFFak.jpg`

Gambar: Penjelasan terkait alamat gambar (images)

Di situ ditulis bahwa alamat gambar terdiri dari 3 bagian: `base_url`, `file_size` dan `file_path`.

Informasi terkait alamat `base_url` dan `file_size` ada di link "`/configuration`" (2), sedangkan untuk `file_path` berasal dari property API seperti nilai `poster_path` yang sudah kita dapat sebelumnya. Contoh bentuk akhir dari URL ada di link bagian (3).

Sekarang kita buka link "`/configuration`" (2), yang ternyata itu link ke halaman yang mirip seperti melihat daftar id genre. Informasi mengenai `base_url` dan `file_size` harus diakses dari alamat URL pada bagian bawah.

Gambar: URL API configuration untuk melihat `base_url` dan `file_size`

Berikut alamat yang harus diakses setelah memasukkan API key:

`https://api.themoviedb.org/3/configuration?api_key=248be2d424b1886d3106f9aa4d1e1079`

Menggunakan `hoppscotch.io`, berikut hasil JSON yang didapat (dimodifikasi sedikit agar tampilannya lebih singkat):

```
1  {
2    "images": {
```

```

3     "base_url": "http://image.tmdb.org/t/p/",
4     "secure_base_url": "https://image.tmdb.org/t/p/",
5     "backdrop_sizes": [
6       "w300", "w780", "w1280", "original"
7     ],
8     "logo_sizes": [
9       "w45", "w92", "w154", "w185", "w300", "w500", "original"
10    ],
11    "poster_sizes": [
12      "w92", "w154" , "w185", "w342", "w500",
13      "w780", "original"
14    ],
15    "profile_sizes": [
16      "w45", "w185", "h632", "original"
17    ],
18    "still_sizes": [
19      "w92", "w185", "w300", "original"
20    ]
21  },
22  "change_keys": [
23    ...
24 ]

```

Di baris 3-4 terdapat property `base_url` dan `secure_base_url`. Inilah alamat URL dasar untuk pengambilan gambar. Karena saat ini `https` sudah sangat umum digunakan, saya akan pakai versi `secure_base_url` saja, yakni dengan alamat `path` di "`https://image.tmdb.org/t/p/`".

Untuk `file_size`, tersedia berbagai pilihan tergantung jenis gambar. Data yang ingin kita pakai nanti adalah `poster_path`, sehingga ukuran gambarnya ada di pilihan `poster_sizes`.

Ukuran gambar untuk `poster_sizes` ada 7, mulai dari `w95` dengan lebar gambar 95 pixel hingga ukuran "original" yang bisa jadi memiliki lebar hingga 2000 pixel. Semakin lebar gambar, semakin besar juga ukuran file yang harus di download oleh pengunjung web. Untuk keperluan ini, saya akan pakai "`w342`" saja, dimana lebar gambar 324 pixel sudah dirasa cukup untuk aplikasi kita.

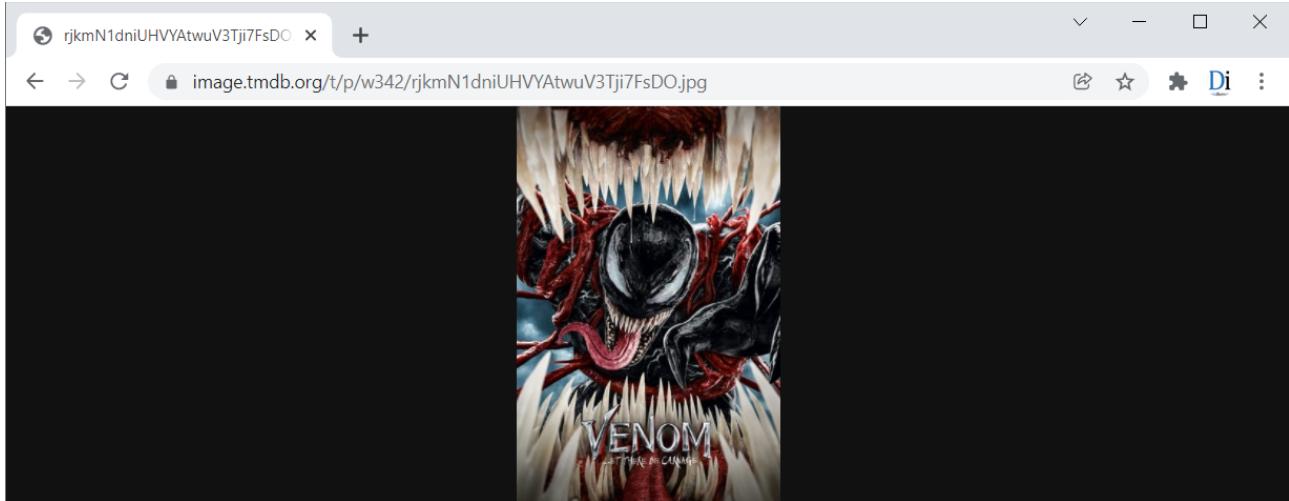
Dengan demikian, ketiga data sudah tersedia:

- ◆ `secure_base_url: https://image.tmdb.org/t/p/`
- ◆ `file_size: w342`
- ◆ `file_path: /rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg`

Sehingga alamat akhir path file gambar menjadi:

`https://image.tmdb.org/t/p/w342/rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg`

Berikut tampilannya di web browser:



Gambar: File gambar movie "Venom: Let There Be Carnage" berhasil di akses

Sip, kita sudah berhasil mendapatkan alamat lengkap gambar movie. Di dalam kode program nanti, URL <https://image.tmdb.org/t/p/w342/> tinggal di sambung dengan nilai poster_path.

Kembali ke API movie, selain genre_ids dan poster_path, property lain yang akan kita akses adalah overview, release_date, title, dan vote_average. Untuk overview dan title, itu bisa langsung dipakai karena sudah berisi teks resensi dan judul movie.

Untuk release_date, perlu sedikit pemrosesan karena nilai bawaan adalah tanggal lengkap seperti "2021-09-30". Untuk aplikasi Ilkoom Movie Database, saya ingin mengambil data tahun saja. Ini bisa dilakukan dengan bantuan method getFullYear() dari Date object JavaScript.

Terakhir untuk vote_average, nilai asli dari themoviedb.org sudah berisi angka desimal seperti 7.2. Ini juga bisa langsung dipakai.

19.5. Mengakses Data API dari React

Setelah selesai mempelajari URL untuk mengakses data API themoviedb.org, kita sudah bisa kembali ke kode program React.

Hal pertama yang akan saya rancang adalah mencoba mengakses URL "discover" yang kita bahas sebelumnya. Untuk sementara, hasil JSON ini cukup ditampilkan ke tab console.

Silahkan buka file `src\App.js`, lalu modifikasi dengan kode berikut:

```
src\App.js

1 import React, { useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4
5 const App = () => {
6   useEffect(() => {
7     const myFetch = async () => {
```

```

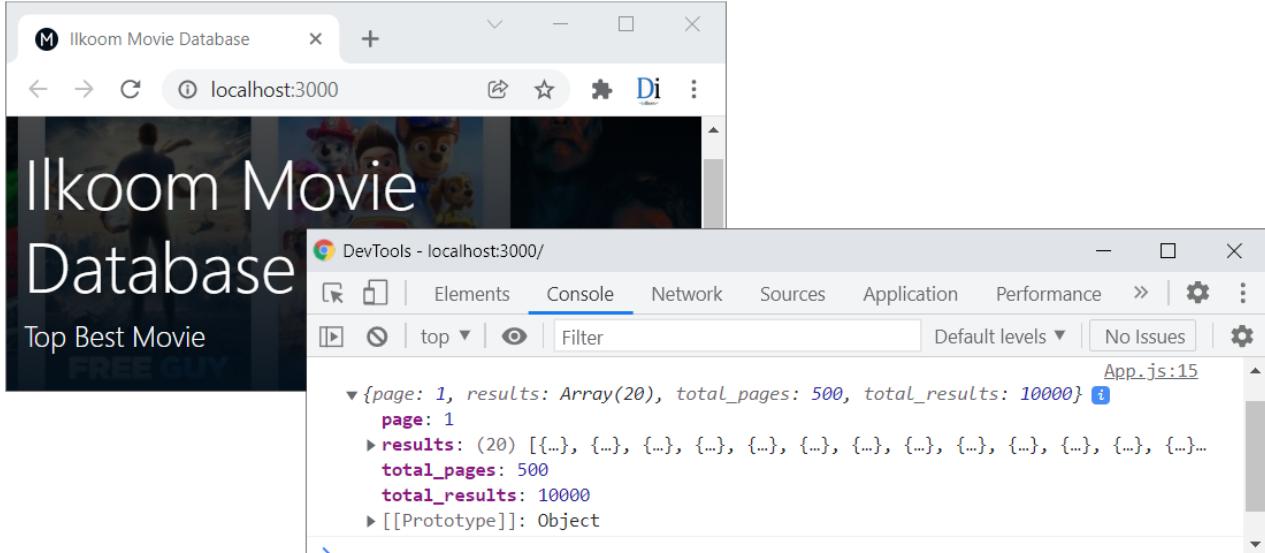
8   try {
9     let url = "https://api.themoviedb.org/3/discover/movie?api_key=248be2d4
10    24b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.
11      desc&include_adult=false&include_video=false&page=1&with_
12      watch_monetization_types=flatrate";
13     let response = await fetch(url);
14     if (!response.ok) {
15       throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
16     }
17     let data = await response.json();
18     console.log(data);
19   }
20   catch (error) {
21     console.log(error);
22   }
23 }
24 myFetch();
25 ], []);
26
27 return (
28   <React.Fragment>
29     <Header />
30     <Footer />
31   </React.Fragment>
32 )
33 }
34
35 export default App;

```

Pada saat mengimport 'react' di baris 1, terdapat tambahan alias `useEffect`. Ini diperlukan karena pengaksesan data API dilakukan dari `useEffect` hook antara baris 6-25. Teknik yang dipakai kurang lebih sama seperti bahasan kita dalam bab sebelumnya.

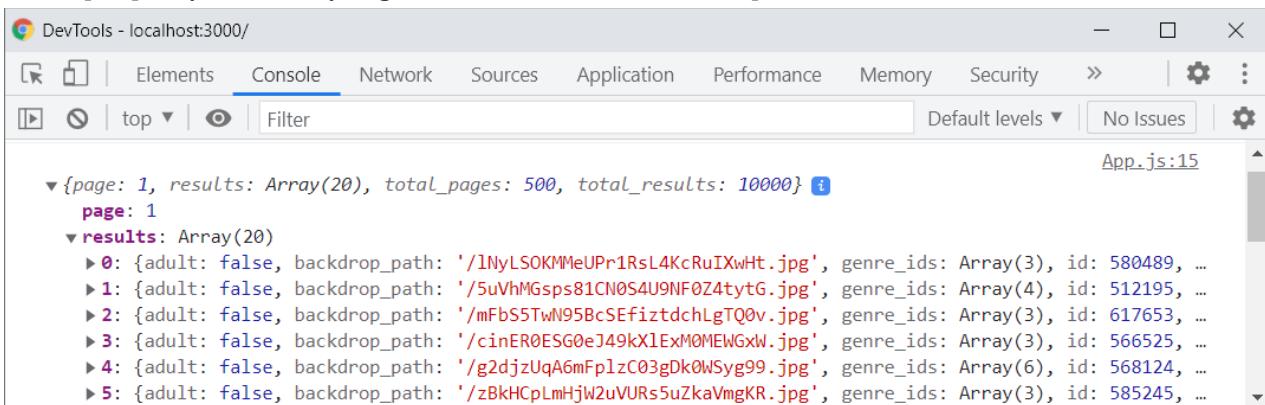
Sebagai catatan, alamat URL API di baris 10-13 harus di tulis dalam 1 baris panjang. Namun karena keterbatasan lebar buku, terpaksa saya pecah ke dalam beberapa baris. Jika anda ingin copy-paste kode di atas dari buku ini, pastikan untuk menghapus spasi dan menulisnya dalam 1 baris saja.

Hasil data JSON dari pemanggilan API ditampung oleh variabel `data` di baris 14, yang langsung diinput ke dalam perintah `console.log(data)` di baris 15. Save kode program di atas, dan lihat hasilnya ke dalam tab console:



Gambar: Tampilan hasil data dari API themoviedb.org

Sama seperti percobaan kita menggunakan hoppscotch.io, data API yang didapat terdiri dari 4 property, yakni `page`, `results`, `total_pages` dan `total_results`. Data movie tersimpan di dalam property `results` yang ketika diklik akan menampilkan data detail untuk ke-20 film:



Gambar: Data movie ada di dalam property results

Selanjutnya, saya ingin membuat state `movie` yang akan menampung semua data movie. Silahkan modifikasi ulang file `App.js` dengan tambahan kode berikut:

src\App.js

```
1 import React, { useState, useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4
5 const App = () => {
6   const [movies, setMovies] = useState([]);
7   useEffect(() => {
8     const myFetch = async () => {
9       try {
10         let url = "https://api.themoviedb.org/3/discover/movie?api_key=248be2d4
11                                     24b1886d3106f9aa4d1e1079&language=en-US&sort_by=popularity.
```

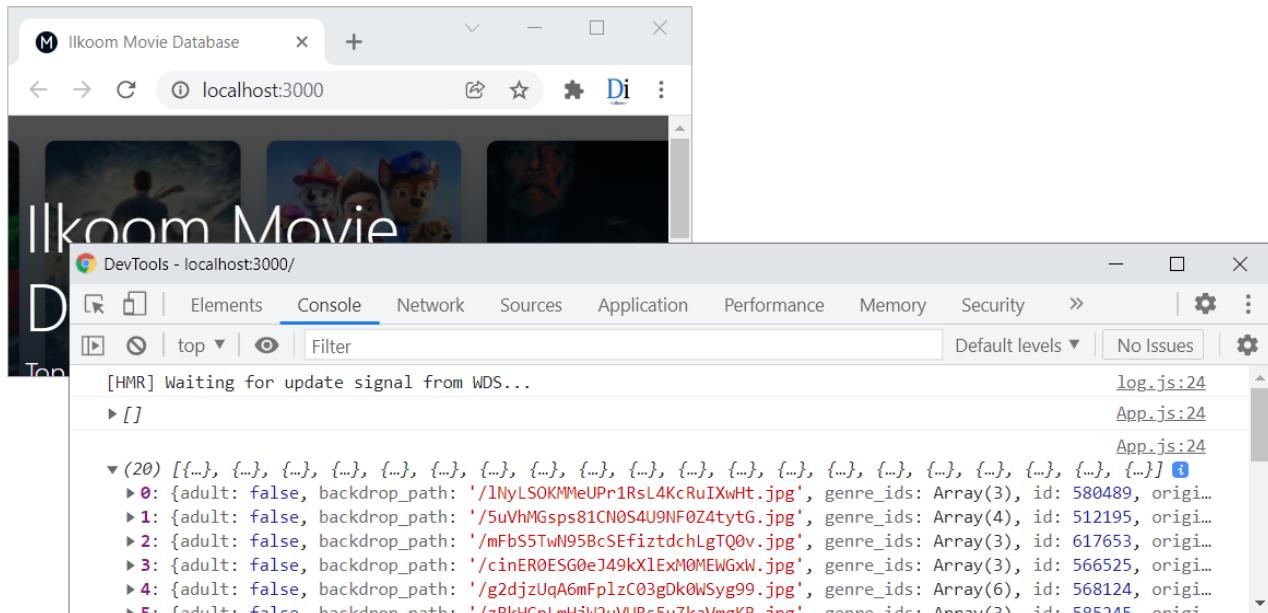
```

12             desc&include_adult=false&include_video=false&page=1&with_
13                 watch_monetization_types=flatrate";
14         let response = await fetch(url);
15         if (!response.ok) {
16             throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
17         }
18         let data = await response.json();
19         setMovies(data.results);
20     }
21     catch (error) {
22         console.log(error);
23     }
24 }
25 myFetch();
26 }, []);
27 console.log(movies);
28
29 return (
30     <React.Fragment>
31         <Header />
32         <Footer />
33     </React.Fragment>
34 )
35 }
36
37 export default App;

```

Tambahan kode program ada di baris 1, 6, 19 dan 27.

Di baris 1, perlu import alias useState karena kita akan membuat state `movies` di baris 6. Ke dalam state `movies`, diinput hasil dari `data.results` seperti di baris 19. Artinya, kita langsung ingin mengambil isi property `results` yang berisi array 20 data movie. Isi dari state `movies` selanjutnya di tampilkan dengan perintah `console.log(movies)` di baris 27. Berikut hasilnya:



Gambar: Tampilan hasil data property results

Sesaat lagi kita akan rancang kode program untuk menampilkan semua data yang tersimpan di dalam state `movies`.

Namun perhatikan hasil yang tampil di tab console. Di baris awal terlihat array kosong [], dan baru diikuti array (20) yang berisi data API movie. Bisakah anda menebak kenapa tampil array kosong [] ini terlebih dahulu?

Hal tersebut terjadi karena isi state `movies` baru di update pada tahap `componentDidMount`, yakni di dalam `useEffect` hook, sedangkan perintah `console.log(movies)` ada di baris 27.

Pada saat halaman di render pertama kali, perintah `console.log(movies)` sudah dijalankan. Akibatnya, akan tampil array kosong di tab console karena state `movies` memang belum berisi data apapun. Barulah setelah semua komponen di render, React menjalankan `useEffect` pada tahap `componentDidMount`, yang kemudian mengupdate isi state `movies`.

Pemahaman seputar React lifecycle sedikit banyak akan membantu kita untuk tidak kaget saat melihat hasil kode program yang di luar harapan.

Sebelum lanjut, saya ingin memodifikasi sedikit alamat URL API yang dipakai. Sampai saat ini, kita masih menggunakan URL awal yang didapat dari halaman "DISCOVER". Dengan memecah query string di setiap baris, berikut rincian dari URL tersebut:

```
1 https://api.themoviedb.org/3/discover/movie
2 ?api_key=248be2d424b1886d3106f9aa4d1e1079
3 &language=en-US
4 &sort_by=popularity.desc
5 &include_adult=false
6 &include_video=false
7 &page=1
8 &with_watch_monetization_types=flatrate";
```

Sebagian besar query string memiliki nilai default, sehingga string di baris 3-8 boleh saja tidak ditulis karena semuanya merupakan nilai default. Dengan demikian, URL API yang kita pakai bisa lebih singkat:

```
1 https://api.themoviedb.org/3/discover/movie
2 ?api_key=248be2d424b1886d3106f9aa4d1e1079
```

Akan tetapi saya ingin menambah 2 pengaturan berikut:

```
3 &certification_country=US
4 &certification.lte=PG-13
```

Dua query di atas berfungsi untuk mengatur tingkatan sensor dari gambar movie. Meskipun secara default sudah ada query string `include_adult=false`, efeknya nyaris tidak ada karena tim themoviedb.org memiliki kriteria yang sangat longgar terkait apa saja yang termasuk kategori "film dewasa".

Sebagai alternatif sensor, kita bisa mengatur hanya menampilkan film dengan rating PG-13 ke

bawah saja. Sebuah film dengan kategori **PG-13** boleh ditonton oleh remaja berumur minimal 13 tahun dengan pengawasan orang tua (PG merupakan singkatan dari *Parental Guidance*). Sehingga film dengan rating dewasa seperti R dan NC-17 tidak muncul di data API.

Jika setelah percobaan nanti anda masih mendapati gambar film yang tidak pantas, bisa turunkan rating ini ke PG atau G. Sebab PG-13 di sini merujuk ke sistem sensor dari negara amerika yang tentu tidak seketal sensor di Indonesia.

Dengan tambahan query string tersebut, maka alamat URL kita menjadi:

```

1 https://api.themoviedb.org/3/discover/movie
2 ?api_key=248be2d424b1886d3106f9aa4d1e1079
3 &certification_country=US
4 &certification.lte=PG-13

```

Silahkan edit kembali file App.js dengan perubahan ini:

```

src\App.js

1 import React, { useState, useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4
5 const App = () => {
6   const [movies, setMovies] = useState([]);
7   useEffect(() => {
8     const myFetch = async () => {
9       try {
10         let url = `https://api.themoviedb.org/3/discover/movie`;
11         url += `?api_key=03ba3f08aa74bf56d5c7f2aa868aad00`;
12         url += `&certification_country=US`;
13         url += `&certification.lte=PG-13`;
14
15         let response = await fetch(url);
16         if (!response.ok) {
17           throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
18         }
19         let data = await response.json();
20         setMovies(data.results);
21       }
22       catch (error) {
23         console.log(error);
24       }
25     }
26     myFetch();
27   }, []);
28   console.log(movies);
29
30   return (
31     <React.Fragment>
32       <Header />
33       <Footer />
34     </React.Fragment>

```

```
35      )
36  }
37
38 export default App;
```

Penulisan URL di baris 10-13 saya pecah agar lebih mudah dibaca dan kodennya tidak harus dalam 1 baris. Nantinya URL ini akan ditambah dengan query string lain untuk pilihan tahun, genre dan nomor halaman.

19.6. Komponen Movie

Data film yang tersimpan di dalam state `movies` akan ditampilkan oleh komponen terpisah, yakni **Movie**. File untuk komponen ini sudah kita siapkan di `src\components\Movie.js`.

Idenya, array di state `movies` akan "dibuka" menggunakan method `.map()`. Dalam setiap iterasi, kirim data 1 film sebagai `props` ke dalam komponen **Movie** untuk diproses lebih lanjut. Teknik ini sudah beberapa kali kita pakai.

Proses pengiriman data dilakukan dari komponen `App` dengan kode program berikut:

```
src\App.js
1 import React, { useState, useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4 import Movie from './components/Movie';
5
6 const App = () => {
7   const [movies, setMovies] = useState([]);
8   useEffect(() => {
9     ...
10    ...
11    }
12    myFetch();
13  }, []);
14
15  return (
16    <React.Fragment>
17      <Header />
18
19      <main className="pb-5">
20        <div className="container">
21          <h2 className="py-5 text-white text-center">Best Movie</h2>
22          <div className="row">
23            {
24              movies.map((movie) => <Movie key={movie.id} movie={movie} />)
25            }
26          </div>
27        </div>
28      </main>
29
```

```
30      <Footer />
31    </React.Fragment>
32  )
33 }
34
35 export default App;
```

Di baris 4 terdapat perintah untuk import komponen `Movie`. Kemudian di dalam kode JSX saya menambah tag `<main>` antara baris 19-28. Tag ini menjadi bagian dari konten utama Ilkoom Movie Database, tempat gambar film akan ditampilkan.

Di baris 20, terdapat tag `<div className="container">` sebagai penanda untuk membuat *grid system* bawaan Bootstrap, kemudian diikuti tag `<h2>` dengan judul teks "Best Movie".

Perulangan `.map()` ada di baris 24. Dalam setiap iterasi, panggil tag `<Movie />` dan kirim data `props` dengan nilai `movie={movie}`. Karena state `movies` berisi 20 object film, maka komponen `Movie` akan dipanggil sebanyak 20 kali pula. Tambahan atribut `key={movie.id}` berguna sebagai penanda nilai unik untuk React.

Tambahan lain, saya juga menghapus perintah `console.log(movies)` yang sebelumnya kita pakai untuk melihat isi state `movies`. Baris ini berada tepat setelah `useEffect` hook.

Sekarang saatnya masuk ke kode program untuk komponen `Movie`:

```
src\components\Movie.js

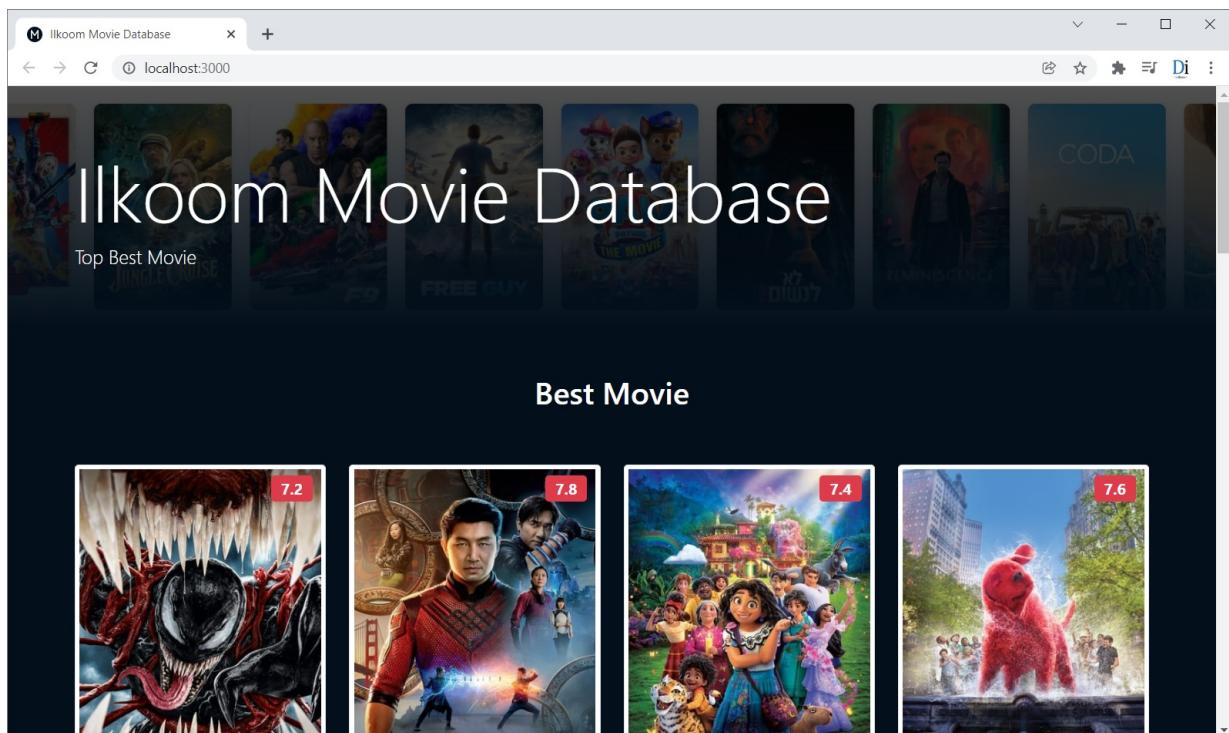
1 import imgPlaceholder from '../img/img-placeholder.png';
2
3 const Movie = (props) => {
4
5   // Ambil gambar poster untuk setiap movie
6   const getImage = () => {
7     if (props.movie.poster_path) {
8       return `https://image.tmdb.org/t/p/w342${props.movie.poster_path}`;
9     }
10    else {
11      return imgPlaceholder;
12    }
13  }
14
15  // Ambil 4 digit tahun
16  const getYear = () => {
17    return new Date(props.movie.release_date).getFullYear();
18  }
19
20  // Potong judul movie jika lebih dari 17 karakter
21  const getTitle = () => {
22    if (props.movie.title.length >= 17) {
23      return props.movie.title.substring(0, 17) + "..."
24    }
25    else {
26      return props.movie.title;
```

```

27     }
28   }
29
30   // Potong keterangan movie jika lebih dari 200 karakter
31   const getOverview = () => {
32     if (props.movie.overview.length >= 200) {
33       return props.movie.overview.substring(0, 200) + "..."
34     }
35     else {
36       return props.movie.overview;
37     }
38   }
39
40   // Ambil nama genre dari setiap movie
41   const getGenre = () => {
42     const genres = [
43       { "id": 28, "name": "Action" },
44       { "id": 12, "name": "Adventure" },
45       { "id": 16, "name": "Animation" },
46       { "id": 35, "name": "Comedy" },
47       { "id": 80, "name": "Crime" },
48       { "id": 99, "name": "Documentary" },
49       { "id": 18, "name": "Drama" },
50       { "id": 10751, "name": "Family" },
51       { "id": 14, "name": "Fantasy" },
52       { "id": 36, "name": "History" },
53       { "id": 27, "name": "Horror" },
54       { "id": 10402, "name": "Music" },
55       { "id": 9648, "name": "Mystery" },
56       { "id": 10749, "name": "Romance" },
57       { "id": 878, "name": "Science Fiction" },
58       { "id": 10770, "name": "TV Movie" },
59       { "id": 53, "name": "Thriller" },
60       { "id": 10752, "name": "War" },
61       { "id": 37, "name": "Western" }
62     ];
63
64     let movieGenre = [];
65     genres.forEach((genre) => {
66       if (props.movie.genre_ids.includes(genre.id)) {
67         movieGenre.push(genre.name);
68       }
69     });
70
71     return (
72       <div>
73         {movieGenre.map((genre) =>
74           <span key={genre.toString()} className="badge bg-success me-1">
75             {genre}
76           </span>)
77         }
78       )
79     }
80
81   return (

```

```
82 <div className="movie-container col-6 col-md-4 col-xl-3 mb-5">
83   {console.log(props.movie)}
84
85   <img src={getImage()} alt={props.movie.title}
86     className="w-100 img-thumbnail"></img>
87   <span className="badge bg-danger vote">{props.movie.vote_average}</span>
88
89   <div className="movie-info">
90     <h2>{getTitle()}</h2>
91     <p>({getYear()})</p>
92     <p className="overview d-none d-lg-block">
93       {getOverview()}
94     </p>
95     {getGenre()}
96   </div>
97 </div>
98 )
99 }
100
101 export default Movie;
```

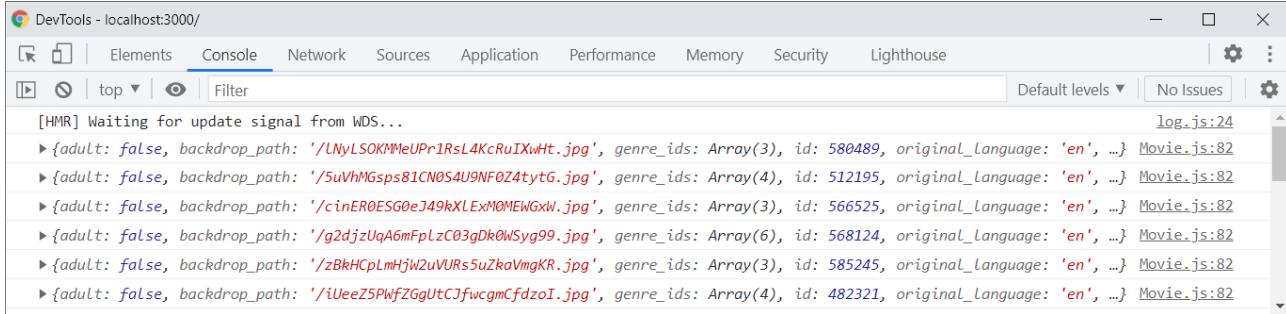


Gambar: Hasil dari penambahan komponen Movie

Kode yang dibutuhkan lumayan panjang. Kita akan bahas secara bertahap mulai dari struktur JSX terlebih dahulu.

Perhatikan kode program di baris 83, di sana terdapat perintah `console.log(props.movie)`. Ini saya tambah hanya untuk proses *debugging* saja, agar kita bisa memastikan nilai setiap film sudah sampai ke komponen Movie. Sebagai pembuktian, bisa buka tab console dan akan terlihat 20 baris object movie:

Mini Project: Ilkoom Movie Database

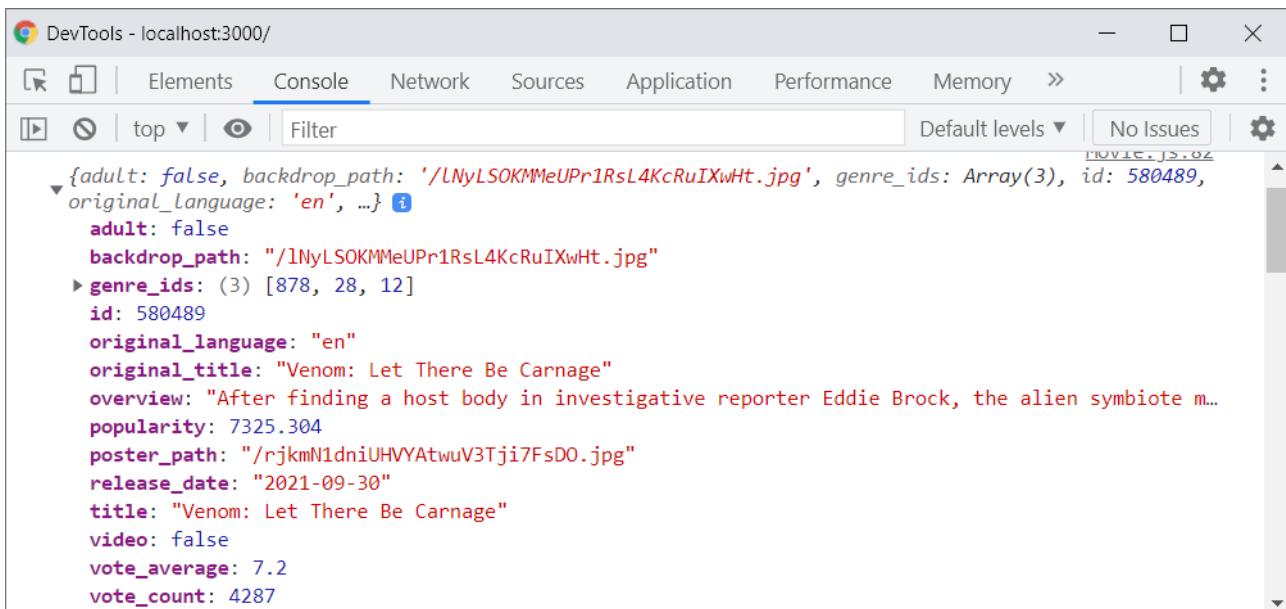


The screenshot shows the DevTools Console tab with 20 movie objects listed. Each object is an array containing a single movie. The properties of each movie include adult (false), backdrop_path (e.g., '/LNyLSOKMMMeUPr1RsL4KcRuIXwHt.jpg'), genre_ids (array of 3 or 4 integers), id (e.g., 580489, 512195, 566525, etc.), original_language ('en'), and original_title (e.g., "Venom: Let There Be Carnage").

```
[HMR] Waiting for update signal from WDS...
▶ {adult: false, backdrop_path: '/LNyLSOKMMMeUPr1RsL4KcRuIXwHt.jpg', genre_ids: Array(3), id: 580489, original_language: 'en', ...} Movie.js:82
▶ {adult: false, backdrop_path: '/SuVhMGsp81CN0S4U9NF0Z4tytG.jpg', genre_ids: Array(4), id: 512195, original_language: 'en', ...} Movie.js:82
▶ {adult: false, backdrop_path: '/cinER0ESG0eJ49kXLExM0MEWGXW.jpg', genre_ids: Array(3), id: 566525, original_language: 'en', ...} Movie.js:82
▶ {adult: false, backdrop_path: '/g2djzUqA6mPlzC03gk0Wsyg99.jpg', genre_ids: Array(6), id: 568124, original_language: 'en', ...} Movie.js:82
▶ {adult: false, backdrop_path: '/zBkHCpLmHjW2uVURs5uZkoVmKR.jpg', genre_ids: Array(3), id: 585245, original_language: 'en', ...} Movie.js:82
▶ {adult: false, backdrop_path: '/iUeeZ5PwfZGgUtCJfwcgmcfdzoI.jpg', genre_ids: Array(4), id: 482321, original_language: 'en', ...} Movie.js:82
```

Gambar: Tampilan 20 object movie di tab Console

Nilai-nilai ini menjadi patokan dari nama props yang ingin di akses. Misalnya jika ingin mengakses judul, maka itu ada di `props.movie.title`, atau untuk gambar movie, ada di `props.movie.poster_path`, dsb:



The screenshot shows the DevTools Console tab with a detailed view of a single movie object. The object has properties like adult (false), backdrop_path ('/LNyLSOKMMMeUPr1RsL4KcRuIXwHt.jpg'), genre_ids ([878, 28, 12]), id (580489), original_language ('en'), original_title ("Venom: Let There Be Carnage"), overview ("After finding a host body in investigative reporter Eddie Brock, the alien symbiote m..."), popularity (7325.304), poster_path ('/rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg'), release_date ("2021-09-30"), title ("Venom: Let There Be Carnage"), video (false), vote_average (7.2), and vote_count (4287).

```
▼ {adult: false, backdrop_path: '/LNyLSOKMMMeUPr1RsL4KcRuIXwHt.jpg', genre_ids: Array(3), id: 580489, original_language: 'en', ...} ⓘ
  ↳ adult: false
    backdrop_path: "/LNyLSOKMMMeUPr1RsL4KcRuIXwHt.jpg"
  ↳ genre_ids: (3) [878, 28, 12]
    id: 580489
    original_language: "en"
    original_title: "Venom: Let There Be Carnage"
    overview: "After finding a host body in investigative reporter Eddie Brock, the alien symbiote m..."
    popularity: 7325.304
    poster_path: "/rjkmN1dniUHVYAtwuV3Tji7FsD0.jpg"
    release_date: "2021-09-30"
    title: "Venom: Let There Be Carnage"
    video: false
    vote_average: 7.2
    vote_count: 4287
```

Gambar: Panduan nama props untuk dari 1 movie

Kembali ke kode JSX, di baris 85 terdapat tag `` dengan nilai atribut `src` yang berasal dari fungsi `getImage()`. Selain itu terdapat pula atribut `alt` dengan nilai `props.movie.title`:

```
<img src={getImage()} alt={props.movie.title} className="w-100 img-thumbnail">
```

Untuk nilai atribut `alt`, itu diakses langsung dari `props` dan tidak ada masalah. Namun untuk nilai atribut `src`, perlu sedikit penjelasan.

Sebagaimana yang pernah kita bahas pada saat menganalisis data API, nilai yang dikirim oleh themoviedb.org hanya bagian akhir dari `path` gambar saja, yang nilainya tersimpan di `poster_path`. Path tersebut harus digabung lagi dengan `base_url` dan `file_size` agar menjadi satu URL utuh.

Supaya lebih rapi, proses penyambungan `path` gambar saya pindahkan ke fungsi terpisah, yakni fungsi `getImage()` di baris 6-13:

```
// Ambil gambar poster untuk setiap movie
const getImage = () => {
  if (props.movie.poster_path) {
    return `https://image.tmdb.org/t/p/w342${props.movie.poster_path}`;
  }
  else {
    return imgPlaceholder;
  }
}
```

Di awal fungsi, terdapat tambahan kondisi if else karena ternyata tidak semua movie memiliki gambar. Untuk movie yang kurang terkenal atau movie lama, property `poster_path` tidak berisi nilai sama sekali. Jadi untuk mengantisipasi hal ini, perlu sedikit pemeriksaan.

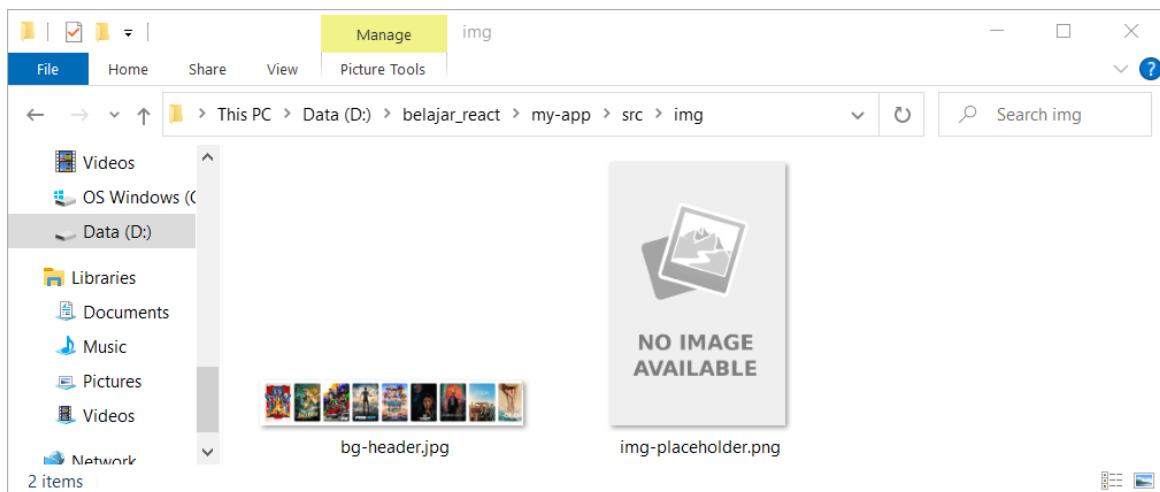
Jika movie tersebut memiliki gambar poster, yakni `if(props.movie.poster_path)` bernilai `true`, maka kembalikan string hasil penyambungan `https://image.tmdb.org/t/p/w342` dengan `props.movie.poster_path`. String ini akan di-return sebagai nilai akhir dari fungsi `getImage()`.

Namun apabila movie tidak memiliki gambar poster, maka `return imgPlaceholder`. Dari mana datangnya `imgPlaceholder`? Ini berasal dari perintah import di baris 1, yakni:

```
import imgPlaceholder from '../img/img-placeholder.png';
```

Perintah import tidak hanya terbatas untuk file CSS dan JavaScript saja, tapi juga bisa ke file lain seperti gambar. Perintah import di atas akan menyimpan path gambar yang ada di alamat `../img/img-placeholder.png` ke dalam variabel `imgPlaceholder`.

Jika anda masih ingat, gambar ini kita copy di persiapan awal bersama gambar `bg-header.jpg`. Lokasinya ada folder `src\img`:

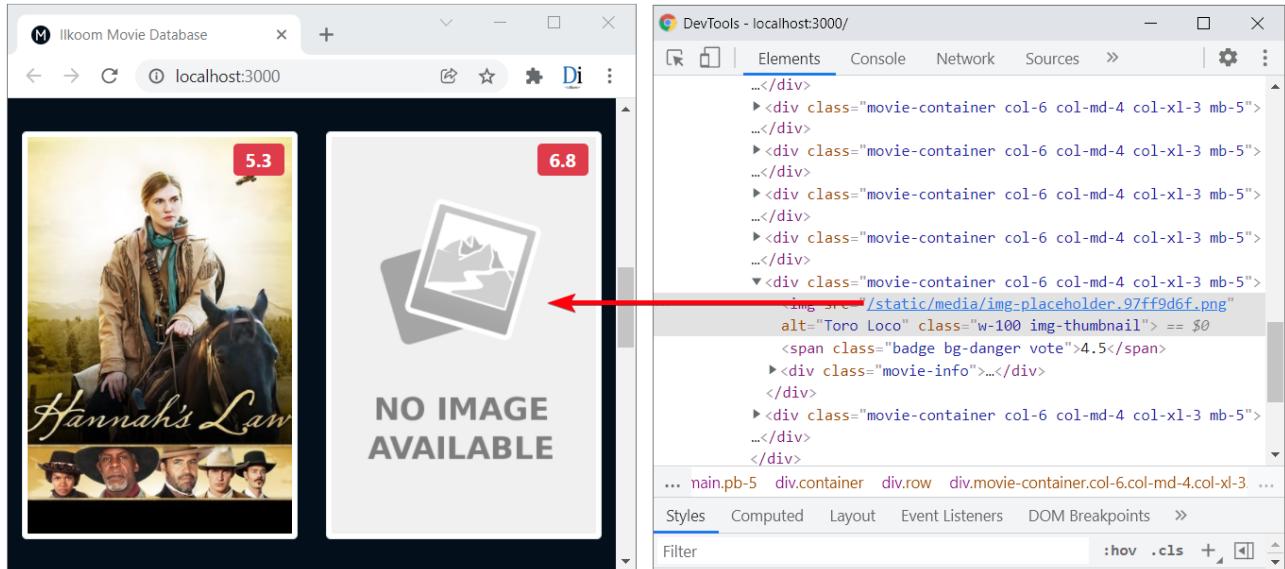


Gambar: Lokasi gambar `img-placeholder.png`

Dengan perintah import di atas, sepanjang kode program variabel `imgPlaceholder` akan berisi path ke gambar yang di-import. Teknik ini cukup penting dipahami karena menjadi cara paling praktis untuk mengakses gambar dengan URL relative dari dalam kode JSX.

Alamat gambar saya tulis sebagai '../img/img-placeholder.png' karena menjadi *path relatif* dari file saat ini. Maksudnya, karena sekarang kita ada di dalam file `components\Movie.js`, maka untuk sampai ke `img-placeholder.png`, harus naik 1 tingkat menggunakan tanda "...", lalu masuk ke folder `img`, dan baru akses file `img-placeholder.png`.

Pada saat di render, aplikasi `webpack` yang ada di dalam `create react app` secara otomatis men-bundle gambar dan memberikan alamat baru, terutama ketika project ini di publish (masih ingat perintah `npm run build?`). Berikut contoh source code yang di dapat nanti:



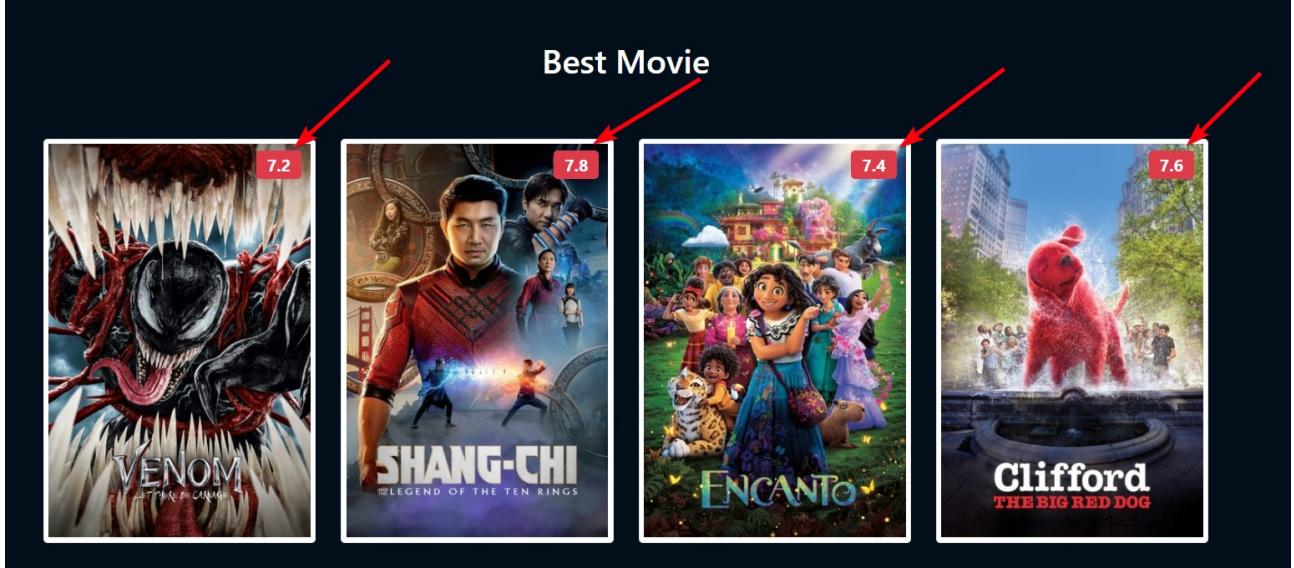
Gambar: Path gambar ada di static/media/img-placeholder.97ff9d6f.png

Sekarang alamat URL gambar memiliki tambahan path "static" yang di generate oleh `create react app`.

Kembali ke kode JSX komponen `Movie`, sekarang kita masuk ke tag `` yang ada di bawah tag `<image>`, yakni di baris 87:

```
<span className="badge bg-danger vote">{props.movie.vote_average}</span>
```

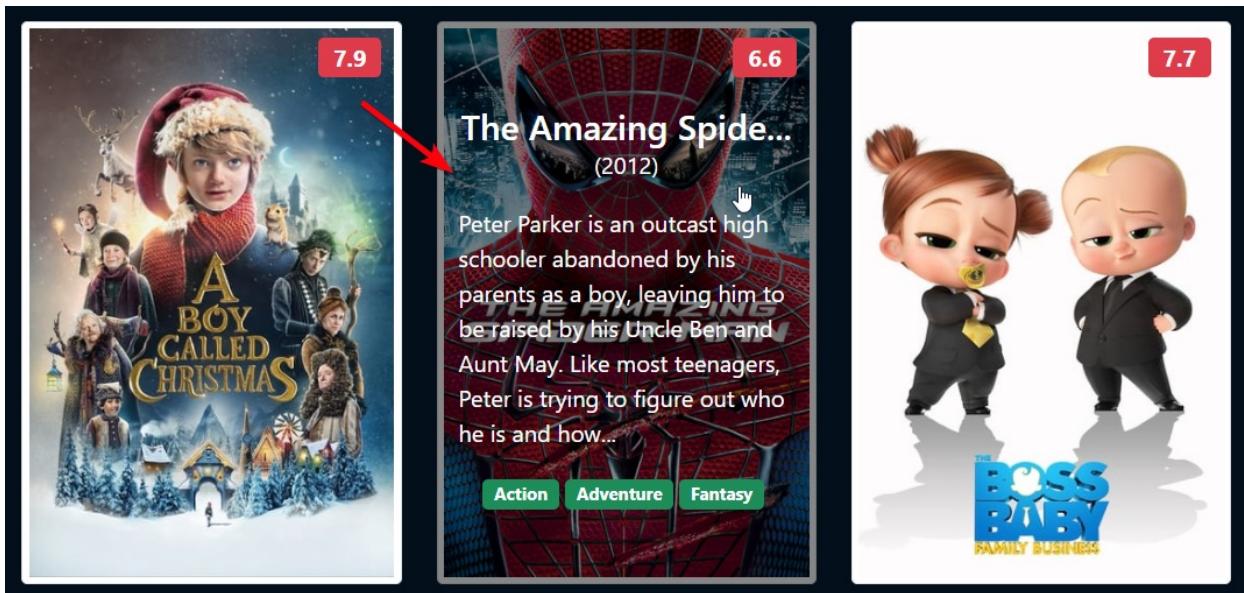
Kode ini dipakai untuk membuat tampilan angka `vote` di sudut kanan atas:



Gambar: Angka voting dari user themoviedb.org

Tidak ada masalah karena cukup langsung akses property `vote_average` sebagai `{props.movie.vote_average}`. Atribut `className="badge bg-danger vote"` adalah nama class bawaan bootstrap untuk membuat tampilan `badge`, yakni kotak kecil merah dengan sudut bulat.

Lanjut, sekarang di dalam kode JSX kita masuk ke tag `<div className="movie-info">`. Ini adalah bagian untuk menampilkan informasi movie dan baru muncul saat gambar movie di hover. Berikut tampilan yang di maksud:



Gambar: Bagian "movie-info" yang baru muncul saat gambar di hover

Struktur pertama untuk "movie-info" adalah bagian judul movie di baris 90. Ini dibuat dengan kode berikut:

```
<h2>{getTitle()}</h2>
```

Saya kembali menggunakan fungsi terpisah karena ingin melakukan sedikit pemrosesan.

Dari data API yang kita dapat, judul movie bisa diakses dari `props.movie.title`. Masalahnya, sebagian judul terlalu panjang sehingga merusak desain halaman. Oleh karena itu saya ingin membatasi panjang judul sebanyak 17 karakter saja. Jika lebih, potong dan sambung dengan tanda titik tiga kali "..." sebagai tanda bahwa judul tersebut masih bersambung.

Pemotongan judul ini dilakukan oleh fungsi `getTitle()` yang didefinisikan pada baris 20-28:

```
// Potong judul movie jika lebih dari 17 karakter
const getTitle = () => {
  if (props.movie.title.length >= 17) {
    return props.movie.title.substring(0, 17) + "..."
  }
  else {
    return props.movie.title;
  }
}
```

Di awal fungsi terdapat pemeriksaan panjang judul movie dengan perintah `if(props.movie.Title.length >= 17)`. Jika ini bernilai `true`, yang artinya judul movie lebih dari 17 karakter, potong judul tersebut dan tambah dengan string "...". Pemotongan dilakukan dengan perintah `props.movie.title.substring(0, 17)`.

Dalam contoh di atas, judul "The Amazing Spider-Man" akan dipotong menjadi "The Amazing Spide...".

Namun jika kondisi `if(props.movie.Title.Length >= 17)` bernilai `false`, yang artinya panjang judul movie kurang dari 17 karakter, langsung saja kembalikan string `props.movie.title` tanpa di potong.

Setelah judul movie, berikutnya giliran tahun rilis di baris 91 dengan kode berikut:

```
<p>{getYear()}</p>
```

Kembali, saya butuh bantuan fungsi terpisah. Alasannya, property `release_date` yang ada di `props.movie` berisi tanggal lengkap dengan format YYYY-MM-DD. Supaya lebih sederhana, saya ingin menampilkan tahun saja, bukan tanggal lengkap.

Berikut pendefinisian fungsi `getYear()` yang ada di baris 15-18:

```
// Ambil 4 digit tahun
const getYear = () => {
  return new Date(props.movie.release_date).getFullYear();
}
```

Untuk bisa mendapatkan data tahun, nilai tanggal di dalam `props.movie.release_date` perlu di konversi menjadi **Date** object bawaan JavaScript. Setelah itu, pemanggilan method `getFullYear()` dari Date object ini akan mengembalikan 4 digit angka tahun.

Sebagai contoh, jika property `props.movie.release_date` berisi string "2012-06-23", maka hasil dari `Date(props.movie.release_date).getFullYear()` akan mengembalikan string "2012".

Setelah tanggal, berikutnya giliran resensi singkat movie di baris 92-94:

```
<p className="overview d-none d-lg-block">
  {getOverview()}
</p>
```

Teks resensi juga perlu sedikit pemrosesan untuk membatasi jumlah karakter agar tidak terlalu panjang. Teknik yang dipakai mirip seperti pemotongan judul movie, hanya saja kali ini saya batasi sebanyak 200 karakter. Berikut pendefinisian fungsi `getOverview()` di baris 31-38:

```
// Potong keterangan movie jika lebih dari 200 karakter
const getOverview = () => {
  if (props.movie.overview.length >= 200) {
    return props.movie.overview.substring(0, 200) + "..."
  }
  else {
    return props.movie.overview;
  }
}
```

Rasanya tidak butuh penjelasan lebih lanjut karena sangat mirip seperti isi fungsi `getTitle()`.

Sekarang kita masuk ke info terakhir sekaligus yang paling kompleks, yakni menampilkan genre movie. Di dalam kode JSX pada baris 95, kode yang dibutuhkan cukup singkat, yakni:

```
{getGenre()}
```

Tentu saja kita perlu membahas apa isi dari fungsi `getGenre()` ini. Pendefinisian fungsi tersebut ada di baris 40-79:

```
// Ambil nama genre dari setiap movie
const getGenre = () => {
  const genres = [
    { "id": 28, "name": "Action" },
    { "id": 12, "name": "Adventure" },
    { "id": 16, "name": "Animation" },
    { "id": 35, "name": "Comedy" },
    { "id": 80, "name": "Crime" },
    { "id": 99, "name": "Documentary" },
    { "id": 18, "name": "Drama" },
    { "id": 10751, "name": "Family" },
    { "id": 14, "name": "Fantasy" },
    { "id": 36, "name": "History" },
    { "id": 27, "name": "Horror" },
    { "id": 10402, "name": "Music" },
    { "id": 9648, "name": "Mystery" },
    { "id": 10749, "name": "Romance" },
    { "id": 878, "name": "Science Fiction" },
    { "id": 10770, "name": "TV Movie" },
```

```
{ "id": 53, "name": "Thriller" },
{ "id": 10752, "name": "War" },
{ "id": 37, "name": "Western" }
];

let movieGenre = [];
genres.forEach((genre) => {
  if (props.movie.genre_ids.includes(genre.id)) {
    movieGenre.push(genre.name);
  }
});

return (
  <div>
    {movieGenre.map((genre) =>
      <span key={genre.toString()} className="badge bg-success me-1">
        {genre}
      </span>)}
  </div>
)
}
```

Di penjelasan tentang data API themoviedb.org, kita sempat bahas bahwa untuk mendapatkan informasi mengenai genre movie, datanya tersimpan di dalam property `genre_ids`. Akan tetapi property ini hanya berisi array dari id genre, seperti [28, 12, 14]. Untuk mendapatkan nama genre, harus melihat lagi daftar array genre menggunakan API.

Daftar id genre sebenarnya bersifat statis dan (seharusnya) tidak banyak berubah, sehingga akan lebih efisien jika kita copy saja ke dalam variabel. Kode tersebut ada di awal fungsi `getGenre()`, dimana konstanta `genres` saya input dengan array dari semua id genre.

Setelah itu, proses konversi dari id genre ke dalam nama genre dilakukan oleh perulangan `genres.forEach ((genre) => {...})`.

Idenya adalah, jalankan perulangan untuk semua element array yang ada di konstanta `genres`. Dalam setiap iterasi, cek apakah nilai genre itu cocok dengan salah satu id yang ada di dalam `props.movie.genre_ids`. Pemeriksaan ini dilakukan dengan perintah `if(props.movie.Genre_ids.includes(genre.id))`. Jika genre itu cocok, ambil nama genre dan input ke dalam array `movieGenre` dengan perintah `movieGenre.push(genre.name)`.

Sebagai contoh, misalkan `props.movie.genre_ids` saat ini berisi array [28, 12, 14].

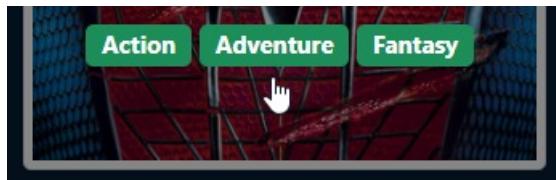
Perintah `genres.forEach()` akan memulai perulangan dari element pertama yang ada di array `genres`, yakni {"id":28,"name":"Action"}. Apakah 28 ada di dalam array [28, 12, 14]? Yup ada, maka nilai "Action" akan diinput ke dalam variabel `movieGenre`.

Lanjut, perulangan masuk ke element `genres` kedua, yakni {"id":12,"name":"Adventure"}. Apakah 12 ada di dalam array [28, 12, 14]? Juga ada, maka nilai "Adventure" akan diinput lagi ke dalam variabel `movieGenre`.

Perulangan lanjut ke element genres ketiga, yakni {"id":16, "name":"Animation"}. Apakah 16 ada di dalam array [28, 12, 14]? Tidak ada. Maka kode program langsung lanjut memeriksa apakah {"id":35, "name":"Comedy"} ada di dalam array [28, 12, 14], dan begitu seterusnya sampai element terakhir dari array genres, yakni {"id":37, "name":"Western"}.

Setelah perulangan genres.forEach() selesai, variabel movieGenre akan berisi array ["Action", "Adventure", "Fantasy"], sebagai hasil dari nilai awal [28, 12, 14].

Namun langkah kita belum selesai. Saya juga ingin menampilkan setiap genre ke dalam tag . Tujuannya agar nama genre tampil dalam kotak badge milik Bootstrap. Berikut tampilan yang dimaksud:



Gambar: Genre ditampilkan sebagai bagde Bootstrap

Untuk membuatnya, array movieGenre perlu masuk ke dalam perulangan movieGenre.map() di akhir fungsi getGenre(). Jika saat ini variabel movieGenre berisi array ["Action", "Adventure", "Fantasy"], maka hasil akhir perulangan adalah:

```
<div>
  <span className="badge bg-success me-1">Action</span>
  <span className="badge bg-success me-1">Adventure</span>
  <span className="badge bg-success me-1">Fantasy</span>
</div>
```

Inilah kode yang tampil sebagai pengganti perintah {getGenre()} di dalam struktur JSX.

Dengan semua kode ini, di halaman Ilkoom Movie Database akan tampil 20 movie terpopuler versi themoviedb.org.

Karena kode program untuk komponen Movie sudah selesai, silahkan hapus perintah {console.log(props.movie)} yang terdapat di awal struktur JSX (atau bisa juga dijadikan baris komentar).

19.7. Membuat Pilihan Tahun dan Genre

Fitur selanjutnya yang ingin saya tambah adalah membuat pilihan tahun dan genre. Dengan ini, user bisa melihat apa saja movie terpopuler di tahun dan genre tertentu, seperti movie terpopuler di tahun 2015 untuk genre Action.

Kode program pilihan tahun dan genre nantinya ada di dalam komponen App. Loh, kenapa tidak dipisah menjadi komponen tersendiri? Alasannya karena pilihan tahun dan genre ini

diperlukan oleh `useEffect`. Dari beberapa kali percobaan, kode programnya jadi lumayan rumit jika dipisah ke dalam komponen tersendiri.

Pertama, kita akan buat kode untuk menampilkan menu pilih tahun dan genre. Silahkan modifikasi file `App.js` dengan tambahan sebagai berikut:

```
src\App.js

1 import React, { useState, useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4 import Movie from './components/Movie';
5
6 // Generate genre
7 const genres = [
8     { "id": "", "name": "All" },
9     { "id": 28, "name": "Action" },
10    { "id": 12, "name": "Adventure" },
11    { "id": 16, "name": "Animation" },
12    { "id": 35, "name": "Comedy" },
13    { "id": 80, "name": "Crime" },
14    { "id": 99, "name": "Documentary" },
15    { "id": 18, "name": "Drama" },
16    { "id": 10751, "name": "Family" },
17    { "id": 14, "name": "Fantasy" },
18    { "id": 36, "name": "History" },
19    { "id": 27, "name": "Horror" },
20    { "id": 10402, "name": "Music" },
21    { "id": 9648, "name": "Mystery" },
22    { "id": 10749, "name": "Romance" },
23    { "id": 878, "name": "Science Fiction" },
24    { "id": 10770, "name": "TV Movie" },
25    { "id": 53, "name": "Thriller" },
26    { "id": 10752, "name": "War" },
27    { "id": 37, "name": "Western" }
28 ];
29
30 // Generate tahun
31 let years = [];
32 const thisYear = new Date().getFullYear();
33 for (let i = 0; i < 10; i++) {
34     years.push(thisYear - i);
35 }
36
37 const App = () => {
38     const [movies, setMovies] = useState([]);
39     const [year, setYear] = useState(thisYear);
40     const [genreId, setGenreId] = useState("");
41     const [genreName, setGenreName] = useState("All");
42
43     const handleYearChange = (event) => {
44         // Untuk mengambil tahun dari <select>
45         setYear(event.target.value);
46     }

```

```

47
48     const handleGenreChange = (event) => {
49         setGenreId(event.target.value);
50
51         // Untuk mengambil nama genre dari <select>
52         let index = event.target.selectedIndex;
53         setGenreName(event.target[index].text);
54     }
55
56     useEffect(() => {
57         const myFetch = async () => {
58             try {
59                 let url = `https://api.themoviedb.org/3/discover/movie`;
60                 url += `?api_key=03ba3f08aa74bf56d5c7f2aa868aad00`;
61                 url += `&certification_country=US`;
62                 url += `&certification.lte=PG-13`;
63
64                 let response = await fetch(url);
65                 if (!response.ok) {
66                     throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
67                 }
68                 let data = await response.json();
69                 setMovies(data.results);
70             }
71             catch (error) {
72                 console.log(error);
73             }
74         }
75         myFetch();
76     }, []);
77
78     return (
79         <React.Fragment>
80             <Header />
81
82             <nav>
83                 <div className="container text-white">
84                     <div className="row">
85                         <div className="col d-none d-md-flex align-items-center">
86                             <hr className="flex-grow-1 me-3" />
87                             <small>powered by themoviedb.org</small>
88                         </div>
89                         <div className="col col-md-3 d-flex">
90                             <div className="me-3">
91                                 <label htmlFor="year" className="form-label">Year</label>
92                                 <select className="form-select" onChange={handleYearChange}>
93                                     value={year} id="year">
94                                         {
95                                             years.map((year) =>
96                                                 <option key={year.toString()} value={year}>
97                                                 {year}
98                                                 </option>
99                                         )
100                                     </select>
101                                 </div>

```

```

102      <div>
103          <label htmlFor="genre" className="form-label">Genre</label>
104          <select className="form-select" onChange={handleGenreChange}>
105              value={genreId} id="genre">
106              {
107                  genres.map((genre) =>
108                      <option key={genre.id} value={genre.id}>
109                          {genre.name}
110                      </option>
111                  )
112              </select>
113          </div>
114      </div>
115  </div>
116      </div>
117  </nav>
118
119  <main className="pb-5">
120      <div className="container">
121          <h2 className="py-5 text-white text-center">
122              {`Best Movie ${year}, Genre: ${genreName}`}
123          </h2>
124          <div className="row">
125              {
126                  movies.map((movie) => <Movie key={movie.id} movie={movie} />)
127              }
128          </div>
129      </div>
130  </main>
131
132      <Footer />
133  </React.Fragment>
134  )
135 }
136
137 export default App;

```

Setelah perintah import, di baris 7-28 terdapat pendefinisian konstanta `genres` yang berisi array object dari pasangan id dan nama genre. Kode ini sangat mirip seperti yang kita buat di dalam komponen `Movie`. Konstanta `genres` nantinya berguna untuk membuat pilihan menu genre dan untuk proses request API ke server themoviedb.org.

Sedikit perbedaan dengan konstanta `genres` di komponen `Movie` sebelumnya, ada di object pertama, yakni `{"id": "", "name": "All"}`. Object ini sengaja saya tambah untuk membuat pilihan menu "All", yang jika dipilih akan me-reset pilihan genre. Ini dipakai untuk menampilkan movie untuk semua genre. Penjelasan lebih lanjut akan kita bahas saat pengaksesan data API sebentar lagi.

Lanjut, kode program di baris 31-35 berguna untuk men-generate array tahun seperti `[2021, 2020, 2019, ...]`. Array ini nantinya dipakai sebagai nilai dari tag `<option>` di menu tahun. Agar tidak terlalu panjang, pilihan tahun saya batasi sebanyak 10 saja.

Daripada men-generate angka tahun statis (misalnya dari 2021 - 2012), saya ingin membuat angka tahun yang dinamis dan terus update. Caranya, ambil angka tahun terbaru dari sistem komputer user menggunakan kode `const thisYear = new Date().getFullYear()`.

Sebagai contoh, jika tanggal di sistem user saat ini menunjukkan 20 Desember 2021, maka konstanta `thisYear` akan berisi angka 2021. Atau jika saat ini sudah tanggal 2 Februari 2023, maka konstanta `thisYear` juga akan berisi angka 2023.

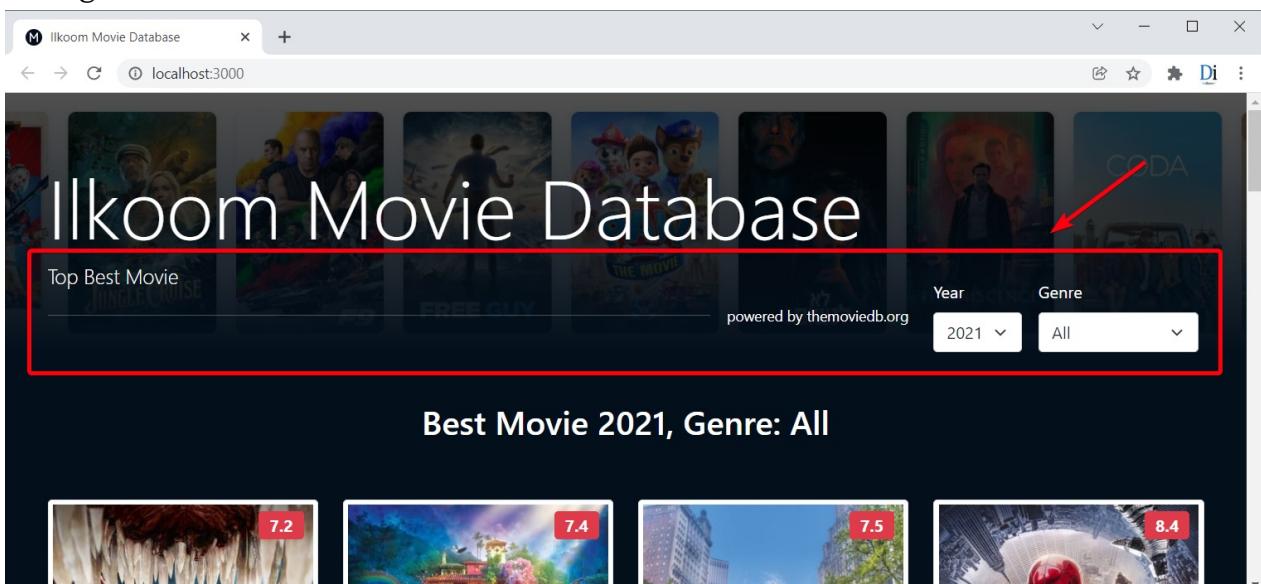
Konstanta `thisYear` selanjutnya dipakai sebagai nilai awal dari perulangan `for` di baris 33-35. Dalam setiap iterasi, kurangi nilai `thisYear` sebesar 1 angka dan simpan angka tahun tersebut sebagai element array ke dalam variabel `years`.

Membuat angka tahun yang dinamis seperti ini bukannya tanpa kelemahan. Bisa saja user sengaja menukar tanggal komputer menjadi 01 Maret 1980 sehingga pilihan tahun di aplikasi kita juga akan mulai dari 1980. Idealnya, angka tahun bisa diambil dari sistem server agar lebih aman (memakai bahasa server seperti PHP atau nodeJS)

Masuk ke dalam komponen App, di baris 39-41 terdapat tambahan 3 buah state baru, yakni `year`, `genreId`, dan `genreName`. Sesuai dengan namanya, ketiga state dipakai untuk menyimpan data tahun, genre id dan genre name.

Nilai awal dari state `year` adalah `thisYear` yang sebelumnya sudah di definisikan pada baris 32 (merujuk ke angka tahun saat ini di sistem komputer user). Untuk state `genreId`, nilai awal diisi string kosong, dan state `genreName` memiliki nilai awal berupa string "All".

Selanjutnya kita lompat dahulu ke struktur JSX. Tambahan paling banyak ada di baris 84-119, yakni di dalam tag `<nav>`. Sebagian besar berisi tag HTML beserta tambahan class Bootstrap. Untuk mengatur posisi element, saya banyak memakai class flex Bootstrap. Berikut tampilan dari tag `<nav>` ini:



Gambar: Hasil dari kode JSX tag `<nav>`

Fokus utama kita ada di kode untuk membuat pilihan **Year** dan **Genre**.

Untuk pilihan tahun (year), kodennya ada di baris 92-100. Tag `<select>` memiliki atribut `onChange={handleYearChange}` dan `value={year}` untuk membuat controlled component React, yakni sebagai penghubung inputan tahun ke dalam state year.

Jika user memilih salah satu tahun, itu akan men-trigger event `onChange={handleYearChange}` yang pendefinisiannya ada di baris 43-46. Fungsi ini hanya berisi 1 perintah `setYear(event.target.value)` untuk mengupdate state year.

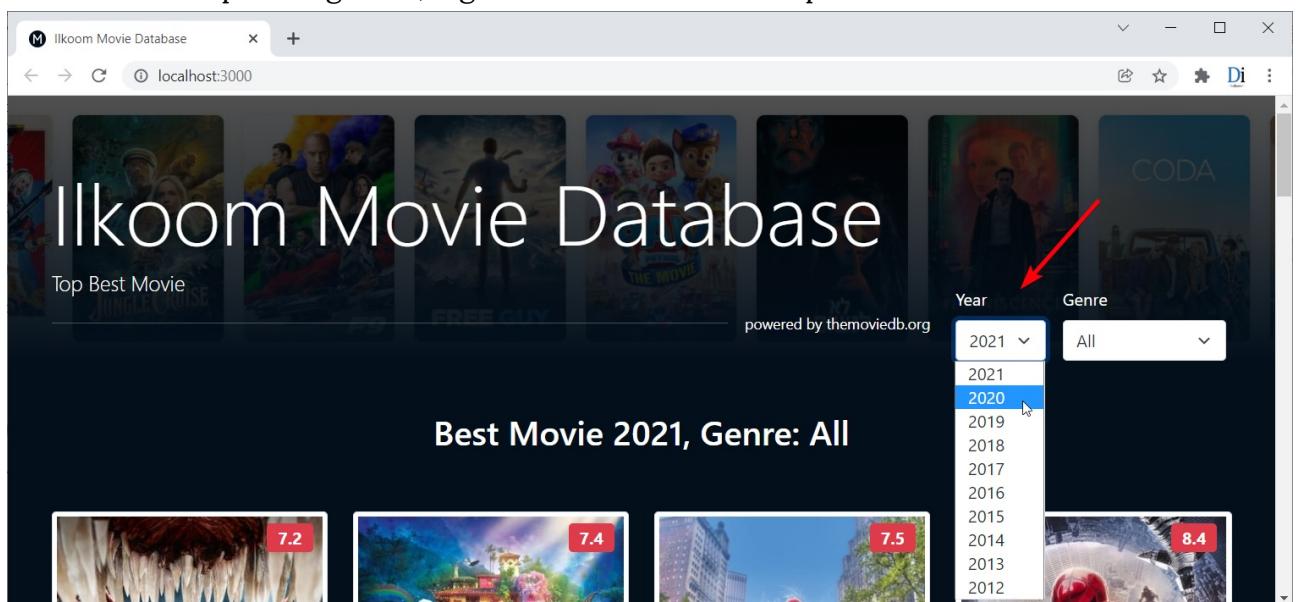
Di dalam HTML, pilihan menu dari tag `<select>` harus ditulis ke dalam tag `<option>`. Agar tidak perlu membuat manual, saya menggunakan perulangan `years.map()` antara baris 95-98 untuk membuat semua pilihan tahun.

Jika anda masih ingat, variabel `years` berisi array angka tahun yang kita generate di awal kode program. Dengan menggunakan perulangan `years.map()`, angka tahun akan di-generate secara otomatis. Dalam setiap iterasi, tampilkan tag `<option>{year}</option>` dengan atribut `key={year.toString()}` dan `value={year}`.

Sebagai contoh, jika array `years` berisi angka [2021, 2020, 2019], maka hasil perulangan `years.map()` akan membuat struktur berikut:

```
<option value="2021">2021</option>
<option value="2020">2020</option>
<option value="2019">2019</option>
```

Hasil akhir dari perulangan ini, tag `<select>` akan berisi 10 pilihan tahun:



Gambar: Pilihan tahun untuk movie

Teknik yang sama juga saya pakai untuk men-generate tag `<option>` untuk pilihan genre yang kodennya ada di baris 104-112.

Kali ini tag <select> memiliki atribut onChange={handleGenreChange} dan value={genreId}. Jika user memilih salah satu genre, itu akan men-trigger event onChange={handleGenreChange} yang pendefinisianya ada di baris 48-54. Fungsi ini akan mengupdate state genreId dengan perintah setGenreId(event.target.value), serta juga men-set genreName dengan perintah setGenreName(event.target[index].text).

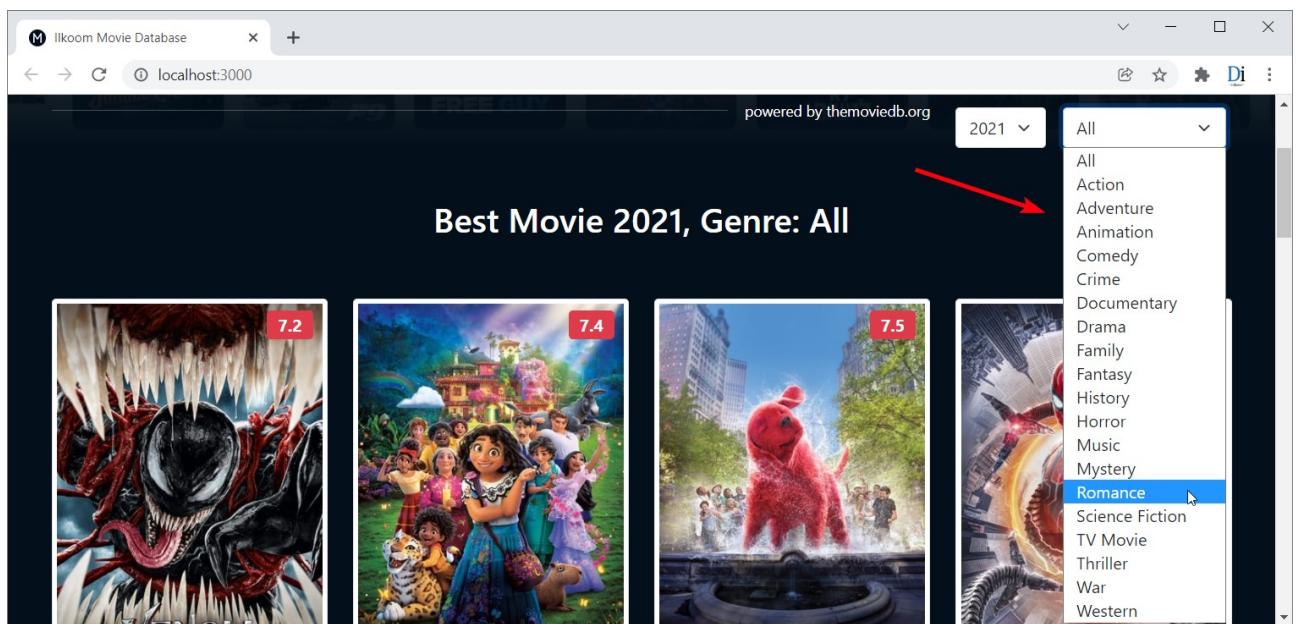
Agak berbeda dengan proses generate tag <option> untuk menu tahun, proses generate tag <option> untuk menu genre memiliki nilai berbeda antara isi atribut value dan teks yang tampil di antara tag pembuka <option> dan tag penutup </option>.

Sebagai contoh, 5 tag <option> pertama dari pilihan genre akan digenerate sebagai berikut:

```
<option value="">All</option>
<option value="28">Action</option>
<option value="12">Adventure</option>
<option value="16">Animation</option>
<option value="35">Comedy</option>
```

Di sini, nilai atribut value berisi id genre, sedangkan teks yang tampil adalah nama genre. Kenapa tidak langsung nama genre saja sebagai isi atribut value? Alasannya karena di dalam query string untuk meminta data API nanti, kita harus input nilai genre dengan kode id, tidak bisa menggunakan nama genre. Namun bagi pengguna aplikasi, tentu lebih suka memilih nama genre daripada nomor id yang tidak berarti apa-apa.

Perulangan genres.map() ini akan men-generate pilihan menu dengan tampilan sebagai berikut:



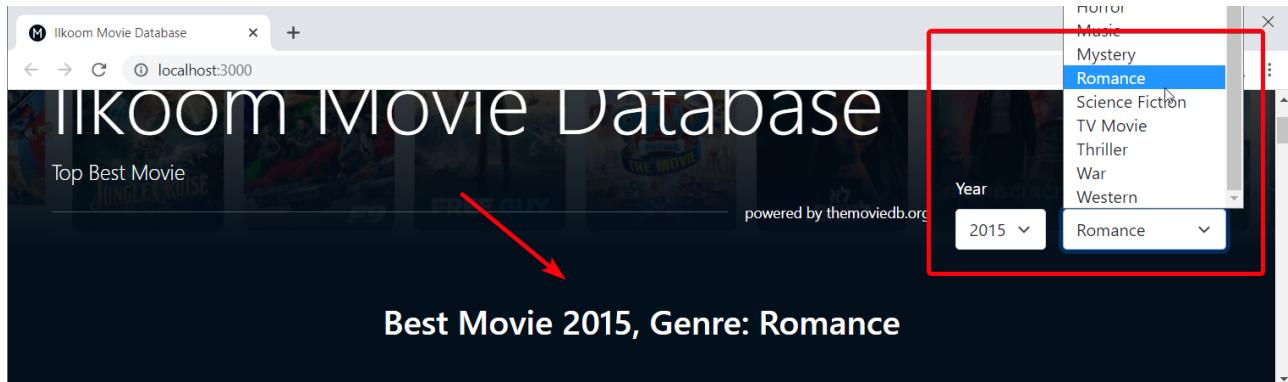
Gambar: Pilihan genre untuk movie

Total terdapat 20 pilihan genre, termasuk "All" yang menjadi pilihan default.

Tambahan kode JSX terakhir ada di baris 121-123, dimana judul tag <main> sekarang berisi teks

yang mengakses nilai state year dan state genreName. Kode ini juga sebagai uji coba apakah menu tahun dan genre movie sudah sampai ke dalam kedua state.

Silahkan tes dengan menukar angka tahun dan pilihan genre. Seharusnya judul teks ini juga akan berubah:



Gambar: Judul untuk tag <main> berubah sesuai pilihan menu

Sampai di sini, pilihan Year dan Genre sudah berhasil kita simpan ke dalam state. Sekarang saatnya tambah kode tersebut ke dalam URL API dari themoviedb.org.

Di materi tentang mengambil data API, web themoviedb.org menyediakan lebih dari 30 pilihan *discover*, termasuk diantaranya tahun dan genre. Jika kita ingin membatasi pilihan tahun rilis sebuah movie, query string yang harus ditambah adalah `primary_release_year`. Dan untuk batasan genre, menggunakan query string `with_genres`.

Sebagai contoh, jika kita ingin menampilkan daftar movie di tahun **2015** untuk genre **comedy**, harus menambah query string sebagai berikut:

```
...&primary_release_year=2015&with_genres=35
```

Nilai dari query `with_genres` adalah nomor id dari genre tersebut.

Di dalam kode program kita saat ini, angka tahun sudah tersedia di state year, dan angka id genre sudah tersimpan di dalam state genreId. Maka, langkah selanjutnya tinggal menambah query string ke dalam alamat API di dalam .

Silahkan modifikasi useEffect hook di dalam komponen App dengan tambahan kode berikut:

src\App.js

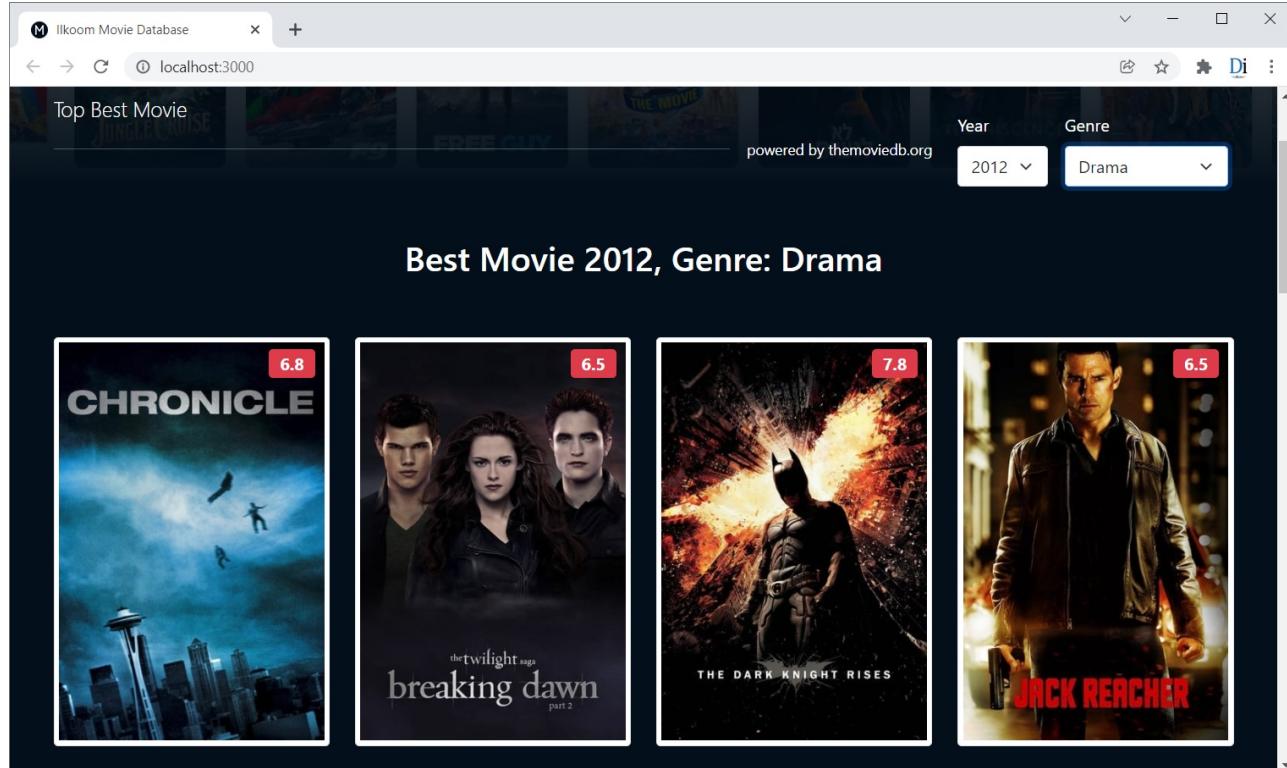
```
1  ...
2  useEffect(() => {
3      const myFetch = async () => {
4          try {
5              let url = `https://api.themoviedb.org/3/discover/movie`;
6              url += `?api_key=03ba3f08aa74bf56d5c7f2aa868aad00`;
7              url += `&certification_country=US`;
8              url += `&certification.lte=PG-13`;
9              url += `&primary_release_year=${year}`;
```

```
10     url += `&with_genres=${genreId}`;
11
12     let response = await fetch(url);
13     if (!response.ok) {
14         throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
15     }
16     let data = await response.json();
17     setMovies(data.results);
18 }
19 catch (error) {
20     console.log(error);
21 }
22 }
23 myFetch();
24 }, [year, genreId]);
25 ...
```

Tambahannya ada di baris 9-10, serta di baris 24.

Untuk di baris 9-10, itu adalah cara menyambung query string dengan tambahan pilihan tahun dan genre. Kemudian kita juga harus memasukkan state year dan genreId sebagai *dependency* dari useEffect hook di baris 24 agar ketika nilai salah satu state ini berubah, halaman ikut di render ulang.

Hasilnya, begitu pilihan Year dan/atau Genre diubah, daftar movie juga akan berubah yang didapat dari permintaan API baru ke server themoviedb.org.



Gambar: Daftar film terpopuler di tahun 2012 untuk genre drama

Perhatikan juga bahwa kode program untuk komponen Movie tidak perlu kita utak-atik sama

sekali. Bagi komponen `Movie`, yang dibutuhkan hanya data 1 movie yang didapat sebagai `props`. Selama data `props` sesuai dengan kebutuhan, komponen tersebut langsung menampilkan data movie yang diminta.

19.8. Membuat Load More

Fitur terakhir yang ingin saya buat ke dalam Ilkoom Movie Database adalah membuat tombol "load more" di bagian bawah halaman. Pada saat tombol ini di klik, akan tampil daftar movie tambahan untuk tahun dan genre yang sedang dipilih.

Sebagai contoh, jika saat ini user memilih tahun 2012 dan genre Drama, maka akan tampil daftar 20 film terpopuler untuk pilihan tersebut. Dengan men-klik tombol load more, akan tampil tambahan 20 film lagi di bagian bawah, dan begitu seterusnya (tombol load more bisa terus di klik). Teknik ini memanfaatkan fitur query string yang disediakan API themoviedb.org.

Untuk membuatnya, silahkan modifikasi kembali file `App.js` dengan tambahan sebagai berikut:

`src\App.js`

```

1 import React, { useState, useEffect } from 'react';
2 import Footer from './components/Footer';
3 import Header from './components/Header';
4 import Movie from './components/Movie';
5
6 // Generate genre
7 const genres = [
8   { "id": "", "name": "All" },
9   { "id": 28, "name": "Action" },
10  { "id": 12, "name": "Adventure" },
11  { "id": 16, "name": "Animation" },
12  { "id": 35, "name": "Comedy" },
13  { "id": 80, "name": "Crime" },
14  { "id": 99, "name": "Documentary" },
15  { "id": 18, "name": "Drama" },
16  { "id": 10751, "name": "Family" },
17  { "id": 14, "name": "Fantasy" },
18  { "id": 36, "name": "History" },
19  { "id": 27, "name": "Horror" },
20  { "id": 10402, "name": "Music" },
21  { "id": 9648, "name": "Mystery" },
22  { "id": 10749, "name": "Romance" },
23  { "id": 878, "name": "Science Fiction" },
24  { "id": 10770, "name": "TV Movie" },
25  { "id": 53, "name": "Thriller" },
26  { "id": 10752, "name": "War" },
27  { "id": 37, "name": "Western" }
28 ];
29
30 // Generate tahun
31 let years = [];

```

```

32 const thisYear = new Date().getFullYear();
33 for (let i = 0; i < 10; i++) {
34   years.push(thisYear - i);
35 }
36
37 const App = () => {
38   const [movies, setMovies] = useState([]);
39   const [year, setYear] = useState(thisYear);
40   const [genreId, setGenreId] = useState("");
41   const [genreName, setGenreName] = useState("All");
42   const [page, setPage] = useState(1);
43
44   const handleYearChange = (event) => {
45     // Untuk mengambil tahun dari <select>
46     setYear(event.target.value);
47
48     // Untuk Reset page
49     setPage(1);
50   }
51
52   const handleGenreChange = (event) => {
53     setGenreId(event.target.value);
54
55     // Untuk mengambil nama genre dari <select>
56     let index = event.target.selectedIndex;
57     setGenreName(event.target[index].text);
58
59     // Untuk Reset page
60     setPage(1);
61   }
62
63   const handleLoadMoreClick = () => {
64     setPage(prevPage => prevPage + 1);
65   }
66
67   useEffect(() => {
68     const myFetch = async () => {
69       try {
70         let url = `https://api.themoviedb.org/3/discover/movie`;
71         url += `?api_key=03ba3f08aa74bf56d5c7f2aa868aad00`;
72         url += `&certification_country=US`;
73         url += `&certification.lte=PG-13`;
74         url += `&primary_release_year=${year}`;
75         url += `&with_genres=${genreId}`;
76         url += `&page=${page}`;
77
78         let response = await fetch(url);
79         if (!response.ok) {
80           throw new Error(`Terjadi gangguan dengan kode: ${response.status}`);
81         }
82         let data = await response.json();
83         // Jika halaman 1, isi ulang state movies
84         // Jika halaman 2 atau lebih, tambahkan ke dalam state movie
85         if (page === 1) {
86           setMovies(data.results);

```

```

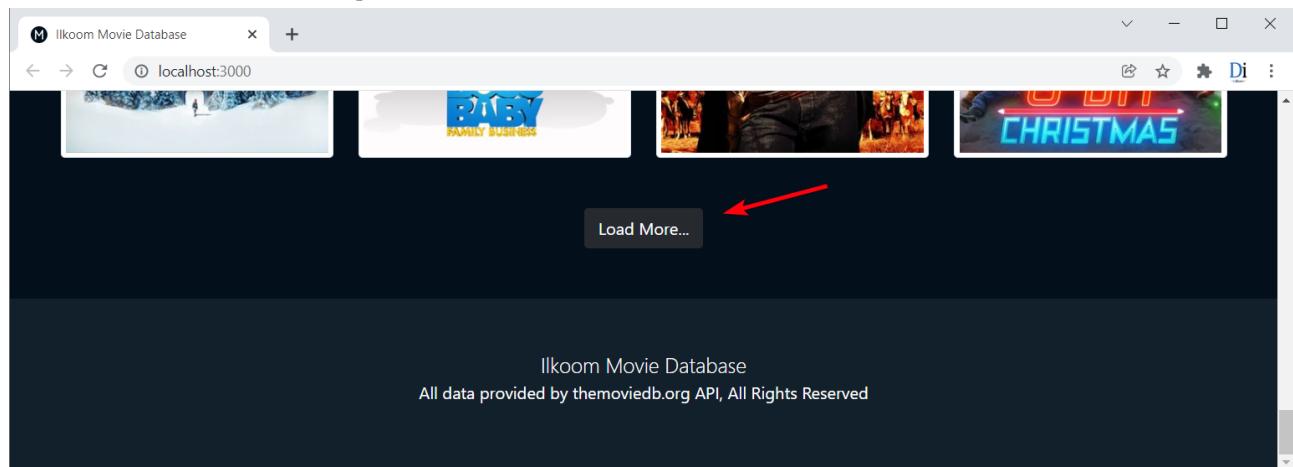
87         } else {
88             setMovies((prevMovie) => [...prevMovie, ...data.results]);
89         }
90     }
91     catch (error) {
92         console.log(error);
93     }
94 }
95 myFetch();
96 }, [year, genreId, page]);
97
98 return (
99     <React.Fragment>
100     <Header />
101
102     <nav>
103         <div className="container text-white">
104             <div className="row">
105                 <div className="col d-none d-md-flex align-items-center">
106                     <hr className="flex-grow-1 me-3" />
107                     <small>powered by themoviedb.org</small>
108                 </div>
109                 <div className="col col-md-3 d-flex">
110                     <div className="me-3">
111                         <label htmlFor="year" className="form-label">Year</label>
112                         <select className="form-select" onChange={handleYearChange}>
113                             value={year} id="year">
114                             {
115                                 years.map((year) =>
116                                     <option key={year.toString()} value={year}>
117                                         {year}
118                                     </option>)
119                             }
120                         </select>
121                     </div>
122                     <div>
123                         <label htmlFor="genre" className="form-label">Genre</label>
124                         <select className="form-select" onChange={handleGenreChange}>
125                             value={genreId} id="genre">
126                             {
127                                 genres.map((genre) =>
128                                     <option key={genre.id} value={genre.id}>
129                                         {genre.name}
130                                     </option>)
131                             }
132                         </select>
133                     </div>
134                 </div>
135             </div>
136         </div>
137     </nav>
138
139     <main className="pb-5">
140         <div className="container">
141             <h2 className="py-5 text-white text-center">
```

```
142         {`Best Movie ${year}, Genre: ${genreName}`}
143     </h2>
144     <div className="row">
145     {
146         movies.map((movie) => <Movie key={movie.id} movie={movie} />)
147     }
148     </div>
149     <div className="row">
150         <div className="col text-center">
151             <button className="btn btn-dark" onClick={handleLoadMoreClick}>
152                 Load More...
153             </button>
154         </div>
155     </div>
156     </div>
157     </main>
158
159     <Footer />
160 </React.Fragment>
161 )
162 }
163
164 export default App;
```

Penambahan kode program untuk membuat fitur *load more* ini tidak terlalu banyak.

Pertama, kita butuh sebuah state baru bernama `page` yang didefinisikan pada baris 42. State `page` dipakai untuk menampung posisi halaman yang akan di request ke server `themoviedb.org`. Sebagai nilai awal, berisi angka 1.

Lompat ke kode JSX, di akhir tag `<main>` terdapat tambahan struktur baru antara baris 149-155. Inilah kode untuk menampilkan tombol "Load More...":



Gambar: Tombol "Load More..." di akhir halaman

Tag `<button>` untuk *load more* memiliki atribut `onClick={handleLoadMoreClick}`. Sehingga pada saat tombol ini di klik, akan menjalankan fungsi `handleLoadMoreClick()` yang ada di baris 63-65. Isinya hanya 1 perintah, yaitu `setPage(prevPage => prevPage + 1)` untuk menaikkan

nilai state page sebanyak 1 angka.

Masuk ke useEffect hook, terdapat tambahan *query string* baru di baris 76 berupa `url += `&page=${page}``. Inilah cara meminta data JSON baru untuk halaman tertentu ke server themoviedb.org.

Secara default, jika *query string* page tidak ditulis, server themoviedb.org akan mengirim data JSON untuk halaman 1 yang berisi data 20 movie. Jika kita ingin meminta data 20 movie lagi, sertakan *query string* `page=2`, kemudian `page=3` untuk 20 movie selanjutnya, dst.

Setiap request data API hanya berisi 20 movie. Oleh karena itu jika kita meminta data API untuk halaman kedua, data movie yang didapat harus ditambah ke dalam state `movies` agar data movie halaman sebelumnya tidak tertimpa. Proses penambahan itu dilakukan pada baris 85-89 dengan sebuah kondisi `if (page === 1)`.

Jika saat ini state `page` berisi angka 1, maka langsung saja isi state `movie` dengan perintah `setMovies(data.results)`. Perintah ini akan menimpa semua isi state `movie` sebelumnya.

Akan tetapi jika isi state `page` bukan 1 (yang berarti tombol "load more..." sudah di klik), data movie yang datang dari API harus ditambah ke dalam state `movie` dengan perintah `setMovies((prevMovie) => [...prevMovie, ...data.results])`). Dengan cara ini, isi state `movie` awal tidak tertimpa dengan data baru.

Tidak lupa, state `page` ditambah sebagai *dependency* dari useEffect hook di baris 96. Artinya setiap kali isi state `page` berubah, halaman perlu di render ulang.

Logika program kita belum selesai, karena bagaimana jika setelah sampai di halaman ketiga dari genre Thriller di tahun 2015 user menukar genre menjadi Mystery? Itu akan me-request halaman ke 3 dari genre Mystery.

Solusinya, reset isi state `page` menjadi 1 setiap kali menu Year atau Genre ditukar. Kode yang dibutuhkan adalah `setPage(1)` di kedua *event handling*, yakni di baris 49 dan 60.

Dengan tambahan ini, selesai sudah kode program untuk aplikasi Ilkoom Movie Database. Silahkan tes semua fitur yang ada seperti ganti pilihan tahun dan genre, lalu tekan tombol load more beberapa kali.

Studi kasus aplikasi Ilkoom Movie Database memperlihatkan kemampuan React dalam memprosesan data API. Kode program yang diperlukan relatif lebih singkat daripada membuat

semuanya menggunakan JavaScript native. Pemecahan kode program menjadi komponen-komponen terpisah juga memudahkan kita dalam penambahan fitur-fitur lanjut.

Jika butuh tantangan, silahkan pelajari dokumentasi API themoviedb.org. Masih banyak query string tambahan yang bisa kita pakai. Selain itu juga ada fitur lain seperti pencarian film berdasarkan judul. Silahkan di explore lebih lanjut.

Tidak lupa juga cukup banyak web penyedia data API lain yang bisa dimanfaatkan untuk membuat aplikasi web yang tidak kalah menarik.

Dalam bab berikutnya kita akan bahas cara mengonlinekan kode program React ke **Firebase Hosting**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

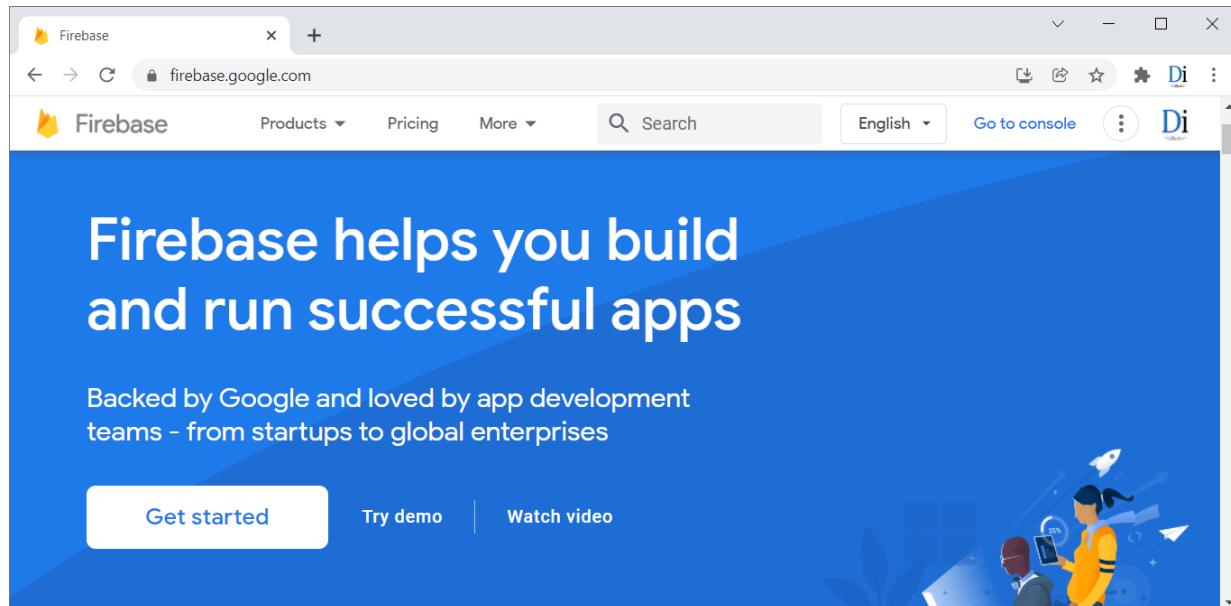
20. Mengonlinekan React App (Firebase Hosting)

Akhirnya kita sampai ke bab terakhir di buku **React Uncover**. Bab ini akan membahas cara mengonlinekan aplikasi React yang sudah kita buat. Pada dasarnya, aplikasi React hanya terdiri dari file HTML, CSS dan JavaScript, sehingga bisa langsung di upload ke web hosting.

Namun jika anda tidak memiliki hosting berbayar, tersedia beberapa layanan hosting gratis di internet. Pilihannya bisa ke 000webhost, heroku, netlify, atau firebase. Dalam praktik kali ini kita akan bahas menggunakan **Google Firebase Hosting**.

20.1. Membuat Akun Firebase

Firebase adalah salah satu layanan milik Google yang berfokus kepada layanan **BaaS (Backend as a Service)**. Maksudnya, firebase berisi kumpulan teknologi untuk mengelola sisi back-end sebuah aplikasi agar *front-end developer* bisa fokus ke alur logika kode program saja, tidak perlu pusing memikirkan konfigurasi server di back end.



Gambar: Tampilan halaman firebase.google.com

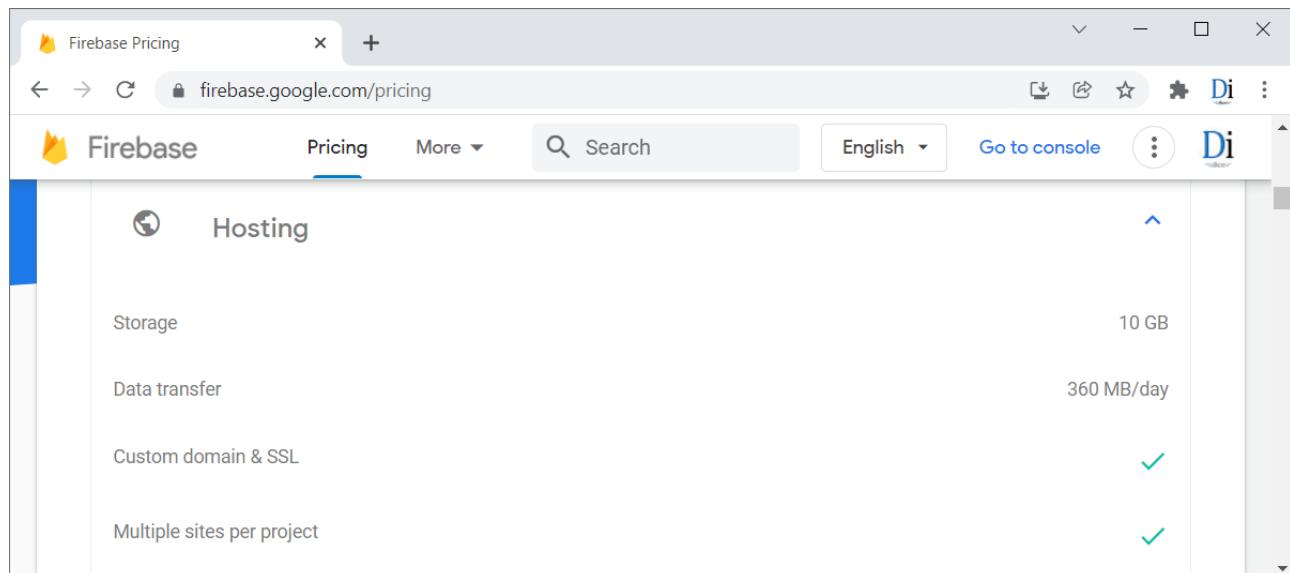
Sebagian dari kita mungkin pernah mendengar bahwa firebase dipakai sebagai database. Betul, salah satu fitur andalan firebase adalah *realtime database* yang menggunakan format NoSQL. Layanan database ini cukup populer sebagai back-end dari aplikasi JavaScript dan juga aplikasi mobile.

Mengonlinekan React App (Firebase Hosting)

Selain database, firebase juga menyediakan fitur hosting yang bisa kita pakai secara gratis (tentunya dengan batasan tertentu).

Untuk kapasitas hosting, tersedia ruang sebesar 10GB yang menurut saya sangat besar untuk sebuah hosting gratisan. Selain itu terdapat batasan *bandwidth* sebesar 360 MB/hari yang lumayan kecil untuk web serius, akan tetapi ini sudah cukup untuk keperluan testing atau halaman portfolio pribadi yang pengunjung hariannya tidak terlalu banyak.

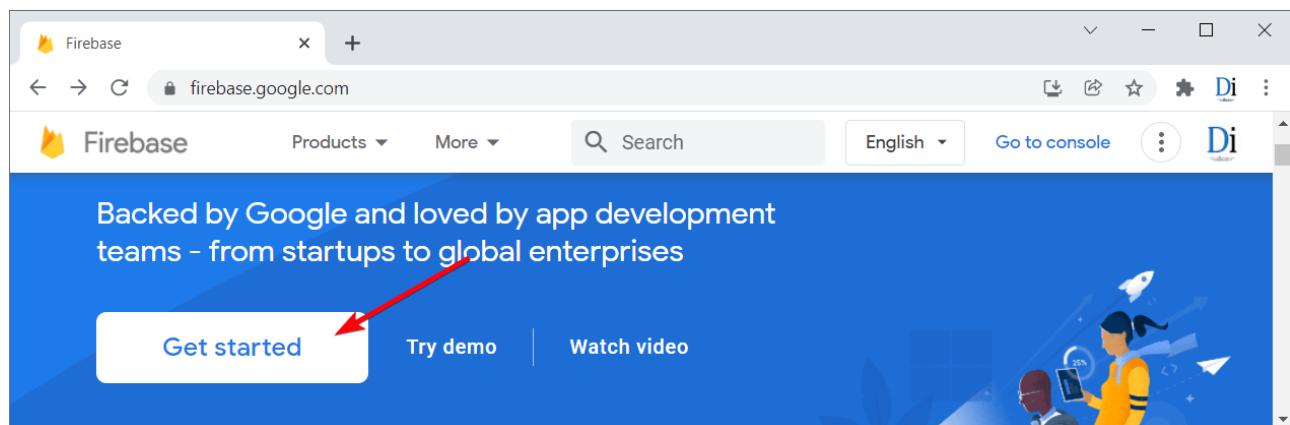
Lebih lengkap tentang batasan versi gratis dan berbayar Firebase bisa kunjungi halaman firebase.google.com/pricing.



Hosting	
Storage	10 GB
Data transfer	360 MB/day
Custom domain & SSL	✓
Multiple sites per project	✓

Gambar: Batasan kapasitas hosting untuk versi gratis firebase

Untuk memulai, silahkan buat akun Firebase di alamat firebase.google.com²⁸. Karena Firebase menjadi bagian dari ekosistem Google, bagi yang sudah punya akun gmail tinggal klik saja tombol "Get Started":



Gambar: Klik tombol "Get Started" untuk login ke halaman dashboard (console) Firebase

Setelah itu kita akan disambut ke proses pembuatan project.

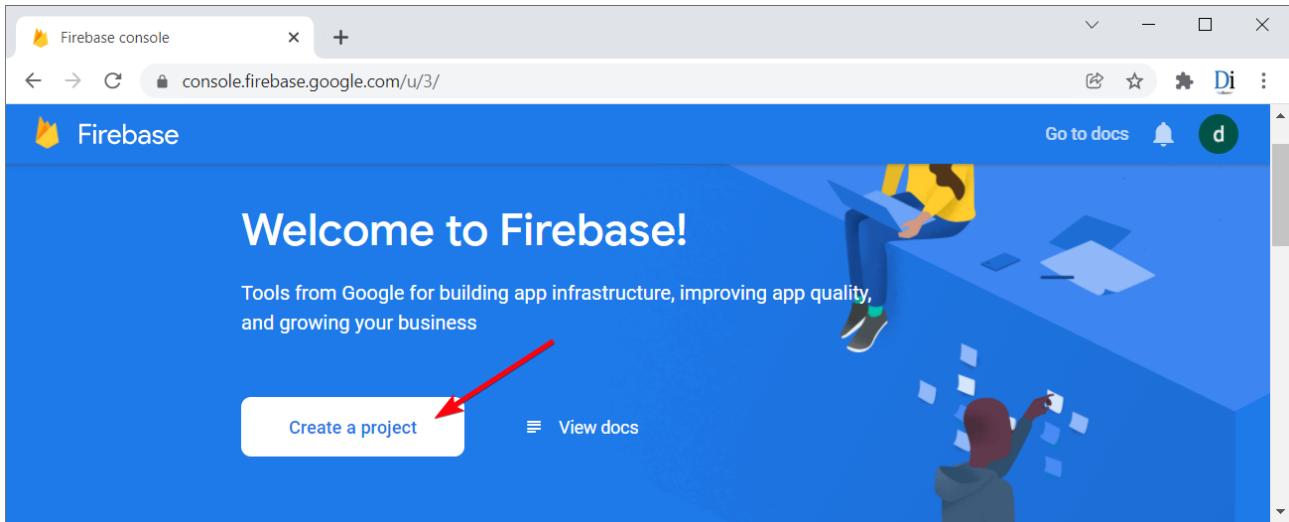
28. <https://firebase.google.com>

20.2. Membuat Firebase Project

Setelah login ke akun firebase, kita disambut halaman admin atau yang disebut firebase sebagai **halaman console**. Hal pertama yang harus kita lakukan adalah membuat project.

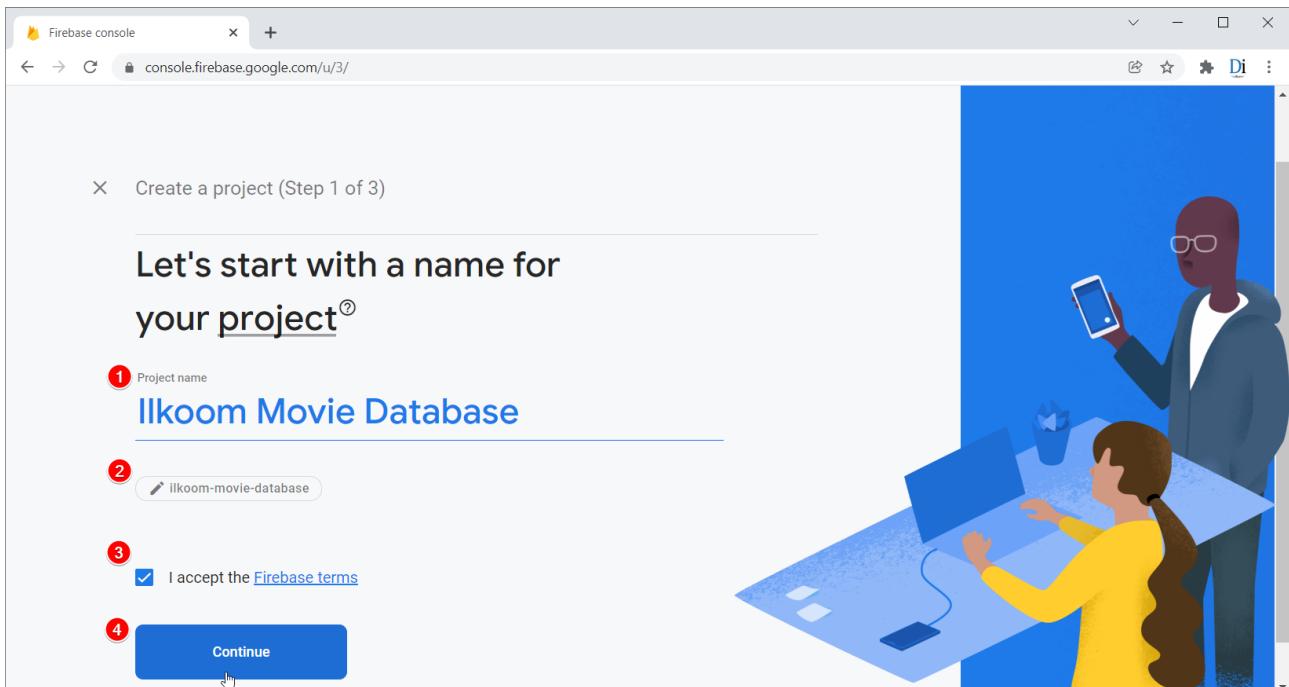
Project mewakili aplikasi yang ingin dibuat. Di satu akun firebase, bisa saja terdapat banyak project (satu untuk setiap aplikasi). Sepanjang praktek dalam bab ini, nantinya kita juga akan membuat beberapa project.

Untuk membuat project baru, silahkan klik tombol "**Create a project**".



Gambar: Klik tombol "create a project" untuk membuat project baru

Di halaman awal pembuatan project, kita diminta menginput nama project:



Gambar: Halaman awal create a project

Mengonlinekan React App (Firebase Hosting)

Nama project ini boleh bebas dan hanya untuk internal akun saja. Dalam contoh ini, saya ingin mengonlinekan aplikasi Ilkoom Movie Database yang baru saja selesai kita buat, oleh karena itu saya menginput nama project sebagai "**Ilkoom Movie Database**" (1). Anda boleh memakai nama project yang sama atau memilih nama lain.

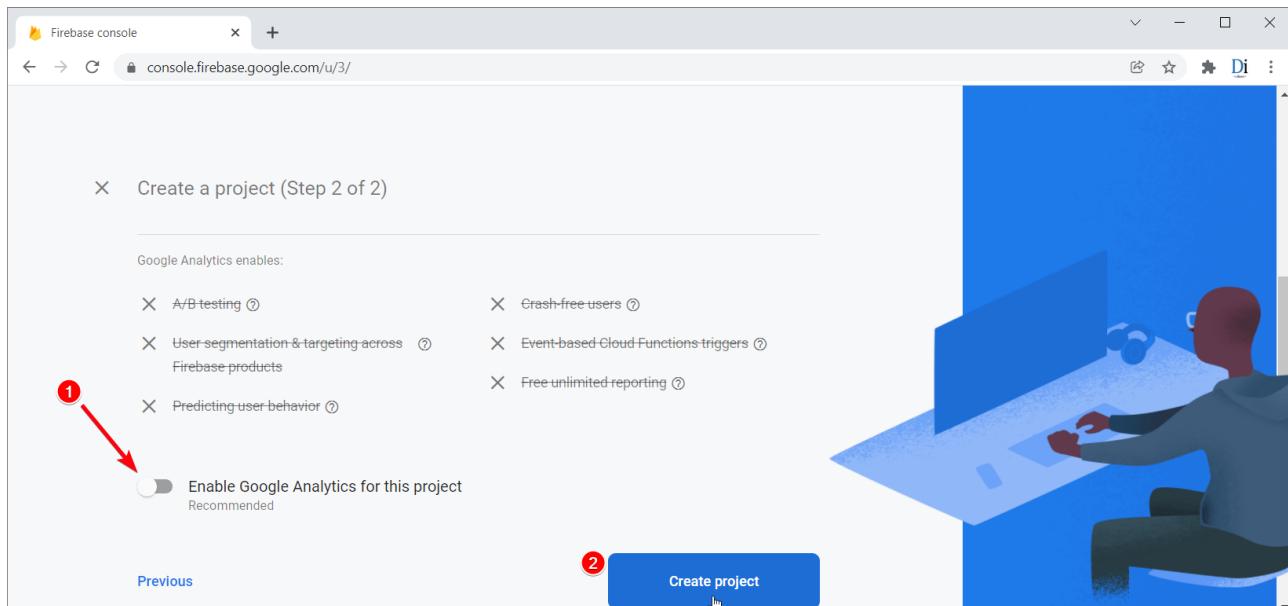
Pada saat kita mengetik nama project, di bagian bawah terdapat **project id** yang otomatis di-generate firebase (2). Dalam contoh ini, project id yang saya dapat adalah "ilkoom-movie-database". Project id bisa di-edit manual dengan mengklik icon pensil di sebelah kiri.

Project id sangat penting karena akan dipakai sebagai URL dari aplikasi kita. Sebagai contoh, karena saya menggunakan id project ilkoom-movie-database, maka nantinya alamat URL aplikasi akan berada di <https://ilkoom-movie-database.web.app>. Karena itu, pastikan project id sudah sesuai dengan keinginan.

Juga karena alamat URL bersifat public, nama project id harus unik (belum pernah dipakai orang lain). Jika id yang kita pilih sudah tidak tersedia, Firebase akan menambah beberapa angka acak di belakang id.

Inputan form selanjutnya adalah persetujuan Firebase terms yang harus di ceklist (3). Dan akhiri dengan men-klik tombol "**Continue**" (4).

Sekarang kita masuk ke halaman integrasi project dengan Google Analytics:

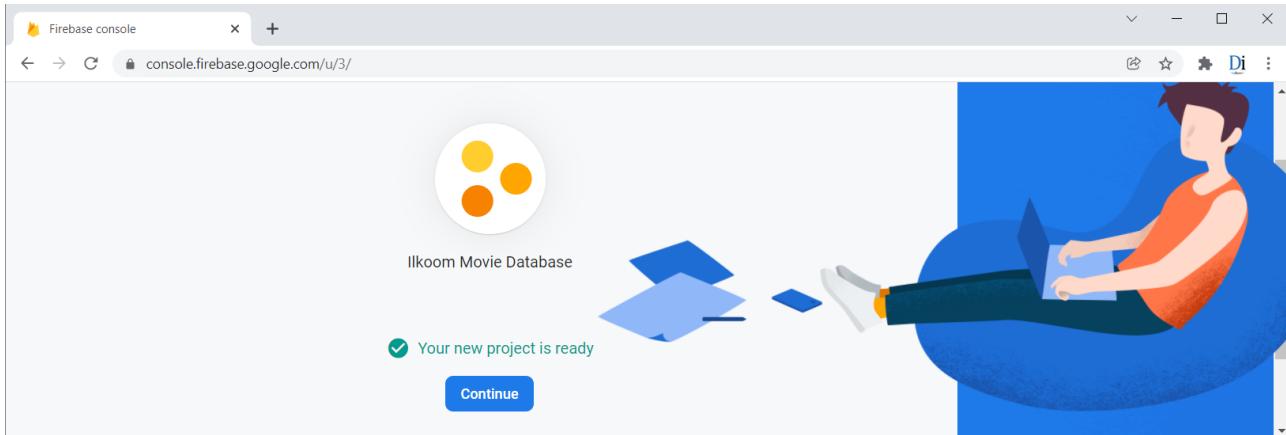


Gambar: Halaman integrasi dengan google analytics

Agar lebih simple, dalam praktik ini kita tidak akan memakai Google Analytics. Karena itu geser slider "Enable Google Analytics for this project" ke posisi off (1). Dan lanjut dengan klik tombol "**Create Project**" (2).

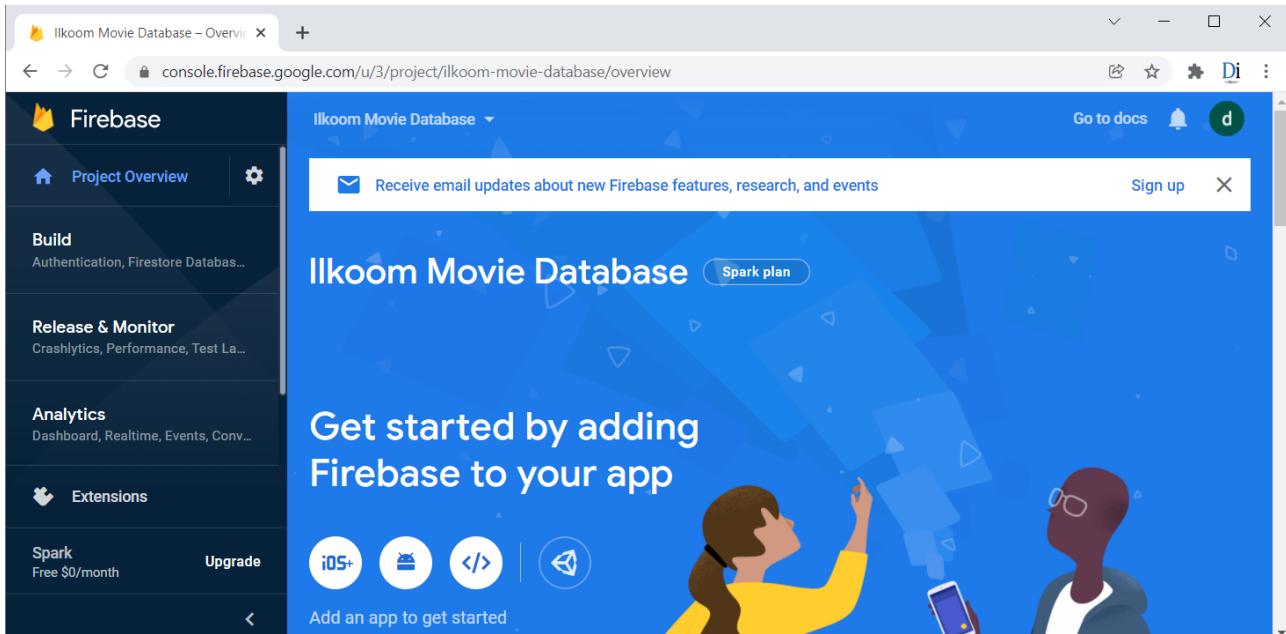
Setelah itu firebase akan menyiapkan project kita.

Mengonlinekan React App (Firebase Hosting)



Gambar: Proses pembuatan project sudah selesai

Setelah proses loading selesai, klik lagi tombol "Continue". Kita dibawa kembali ke halaman console yang sekarang tampilannya menjadi lebih ramai:



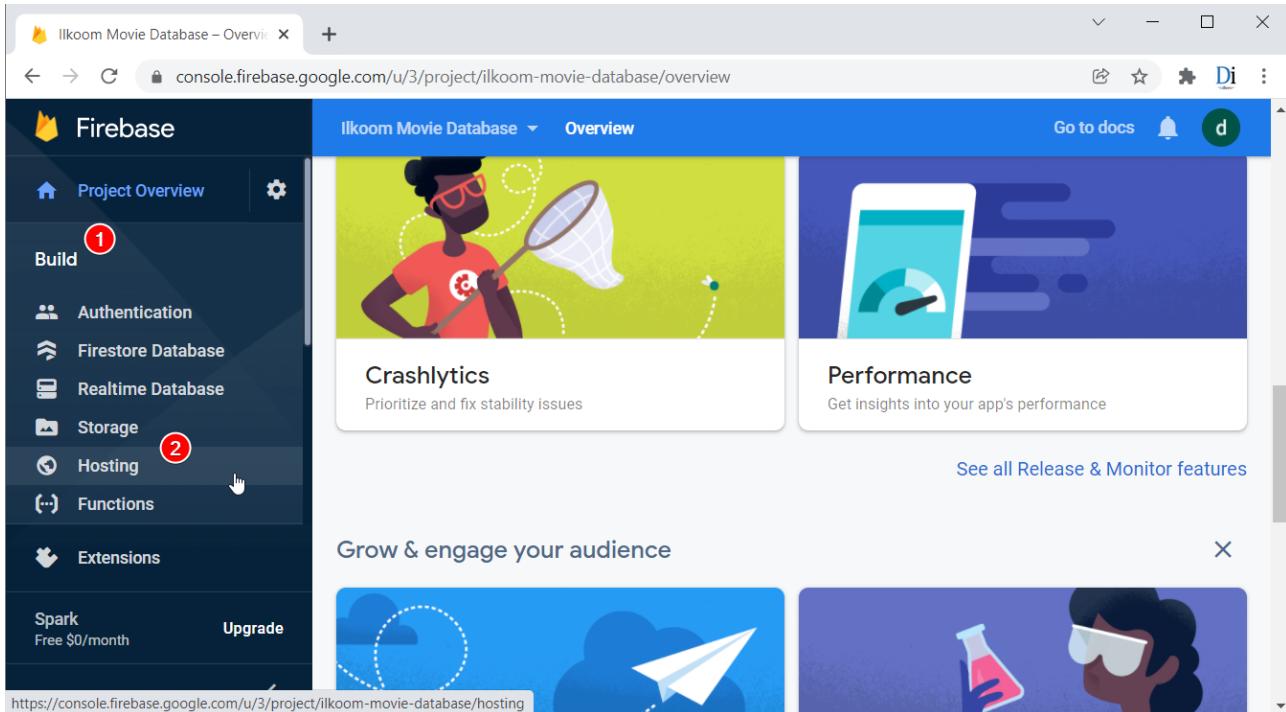
Gambar: Halaman firebase console setelah project selesai dibuat

Di halaman ini kita bisa pelajari dan mendaftar di berbagai layanan yang disediakan firebase. Tapi sebelum "tersesat" (karena banyaknya pilihan menu), sebaiknya ikuti panduan berikutnya.

20.3. Instalasi Firebase CLI

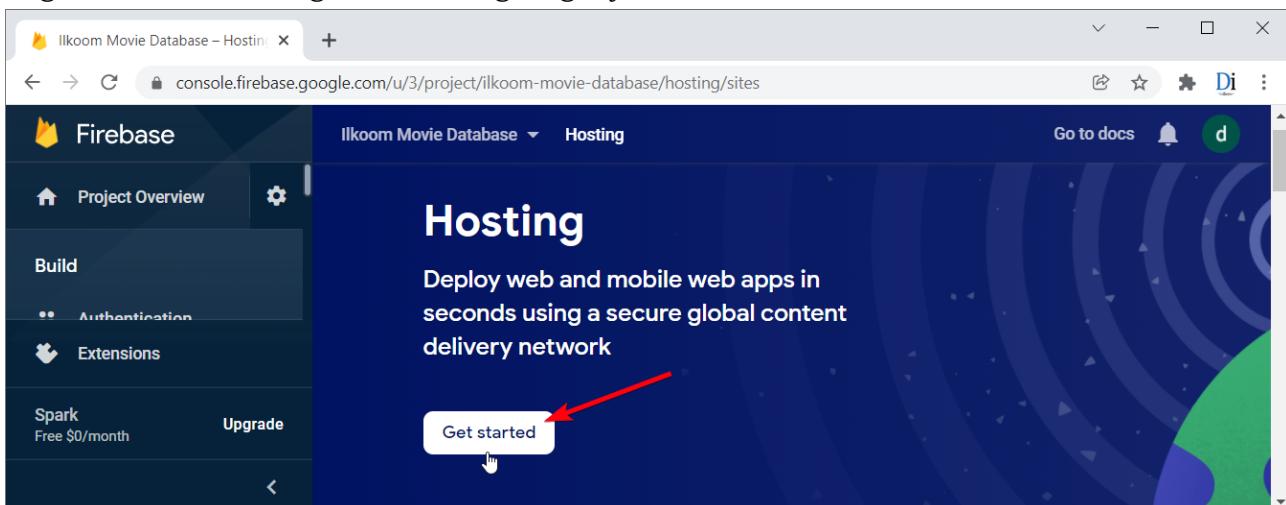
Tujuan utama kita adalah mengonlinekan aplikasi React menggunakan Firebase Hosting. Untuk membukanya, silahkan klik menu "**Build**" di kiri atas (1), lalu pilih "**Hosting**" (2):

Mengonlinekan React App (Firebase Hosting)



Gambar: Pilih menu Build -> Hosting di halaman console firebase

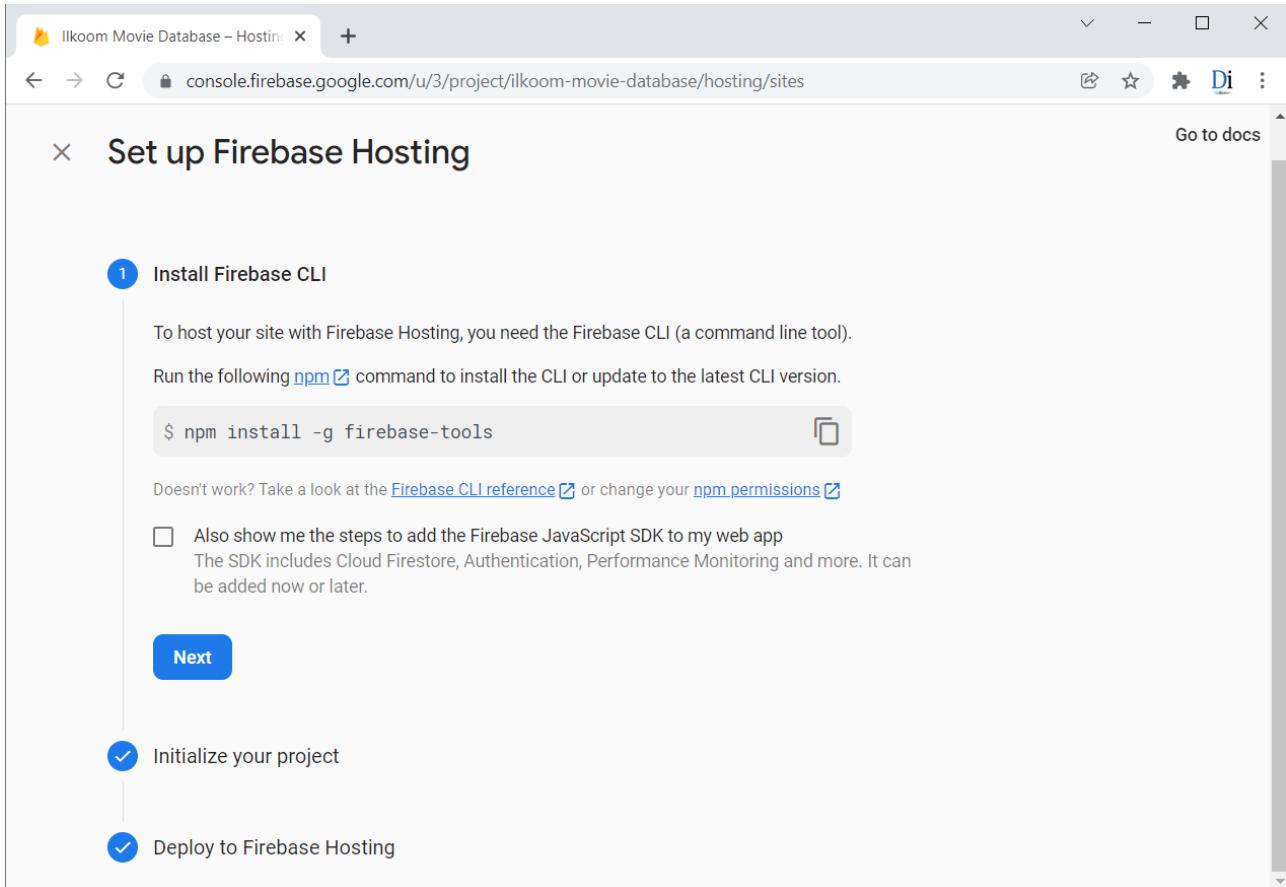
Begitu halaman hosting terbuka, langsung saja klik tombol "**Get started**":



Gambar: Klik tombol "Get started"

Halaman ini berisi instruksi bagaimana cara mengupload file ke firebase hosting:

Mengonlinekan React App (Firebase Hosting)

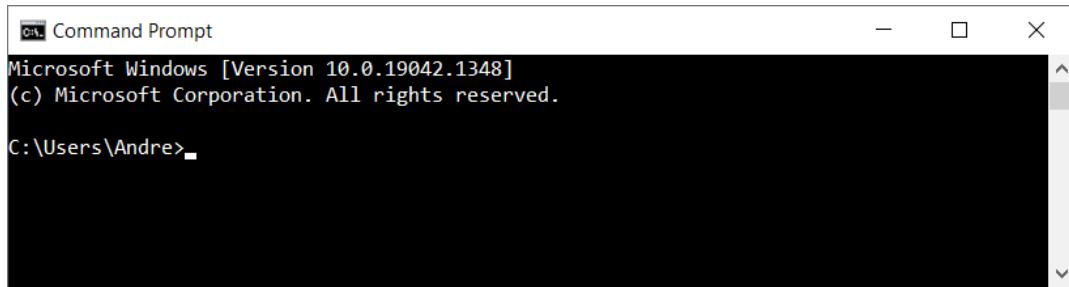


Gambar: Petunjuk cara mengupload file ke firebase hosting

Untuk proses upload file, ternyata harus menginstall **Firebase CLI** terlebih dahulu. Yup, kita perlu mengetik perintah menggunakan terminal atau command prompt. CLI sendiri merupakan singkatan dari **Command Line Interface**.

Meskipun rasanya lebih repot (kenapa tidak langsung drag file saja di web browser?), Firebase CLI sangat praktis dalam proses pengembangan aplikasi. Perintah yang diketik pun tidak terlalu banyak dan instruksinya bisa dibaca dalam halaman ini.

Baik, silahkan minimize web browser dan buka terminal atau command prompt. Untuk Windows 10 bisa dengan klik icon kaca pembesar di sebelah kanan start menu, ketik "cmd" dan tekan tombol Enter:



Gambar: Buka jendela cmd

Mengonlinekan React App (Firebase Hosting)

Secara default jendela cmd akan terbuka di alamat C:\Users\<nama_user>. Ini tidak masalah karena proses instalasi Firebase CLI bisa dilakukan dari mana saja. Syarat lain, di komputer sudah harus sudah terinstall Node.js.

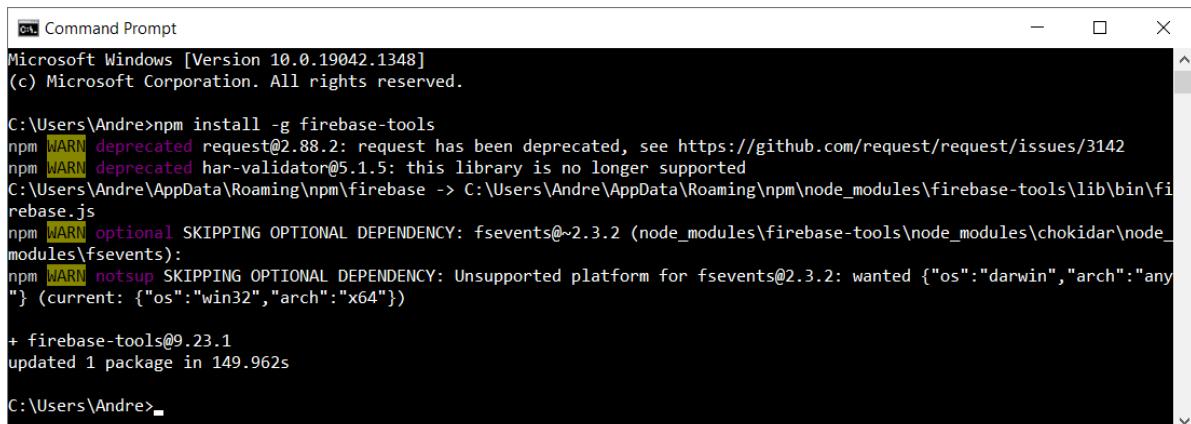
Agar tidak bermasalah dengan pembatasan hak akses ExecutionPolicy, saya sarankan untuk menjalankan perintah CLI dari cmd, bukan terminal bawaan VS Code yang secara default menggunakan PowerShell.

Jika tetap ingin menjalankan perintah CLI di PowerShell, silahkan update pengaturan ExecutionPolicy seperti bahasan [di akhir bab Create React App](#).

Untuk memulai instalasi Firebase CLI, ketik perintah berikut (akhiri dengan tombol enter):

```
npm install -g firebase-tools
```

Proses instalasi akan berlangsung beberapa saat:



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>npm install -g firebase-tools
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
C:\Users\Andre\AppData\Roaming\npm\firebase -> C:\Users\Andre\AppData\Roaming\npm\node_modules\firebase-tools\lib\bin.firebaseio.js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.2 (node_modules\firebase-tools\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ firebase-tools@9.23.1
updated 1 package in 149.962s

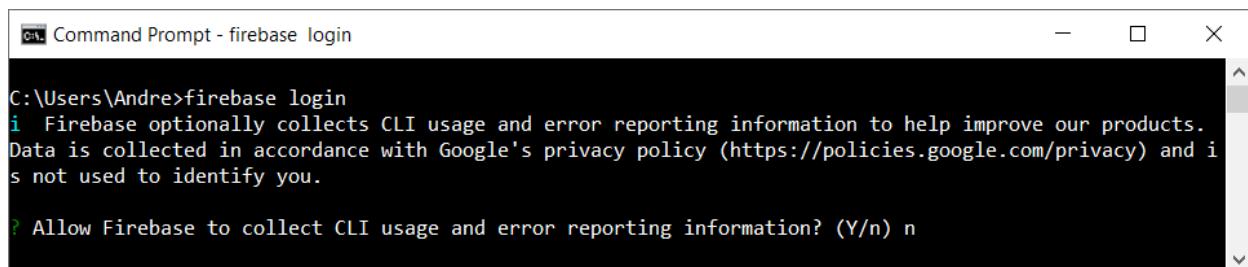
C:\Users\Andre>
```

Gambar: Proses instalasi firebase cli

Setelah instalasi selesai, selanjutnya kita harus login agar Firebase CLI bisa mengakses project yang baru saja dibuat. Caranya, ketik perintah berikut (akhiri dengan tombol enter):

```
firebase login
```

Akan tampil permintaan izin dari firebase terkait pengiriman informasi serta pesan error ke server Google. Jika anda mengizinkan hal ini ketik "Y", atau dalam contoh ini saya akan ketik "n" dan tekan enter.



```
Command Prompt - firebase login
C:\Users\Andre>firebase login
i Firebase optionally collects CLI usage and error reporting information to help improve our products.
Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to identify you.

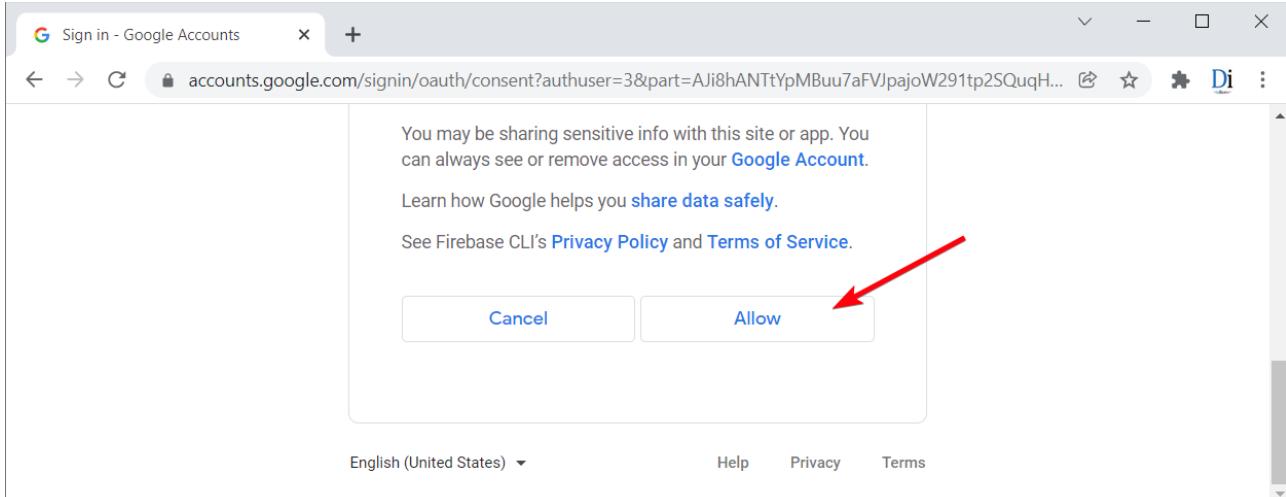
? Allow Firebase to collect CLI usage and error reporting information? (Y/n) n
```

Gambar: Konfirmasi izin pengaksesan data oleh Firebase

Mengonlinekan React App (Firebase Hosting)

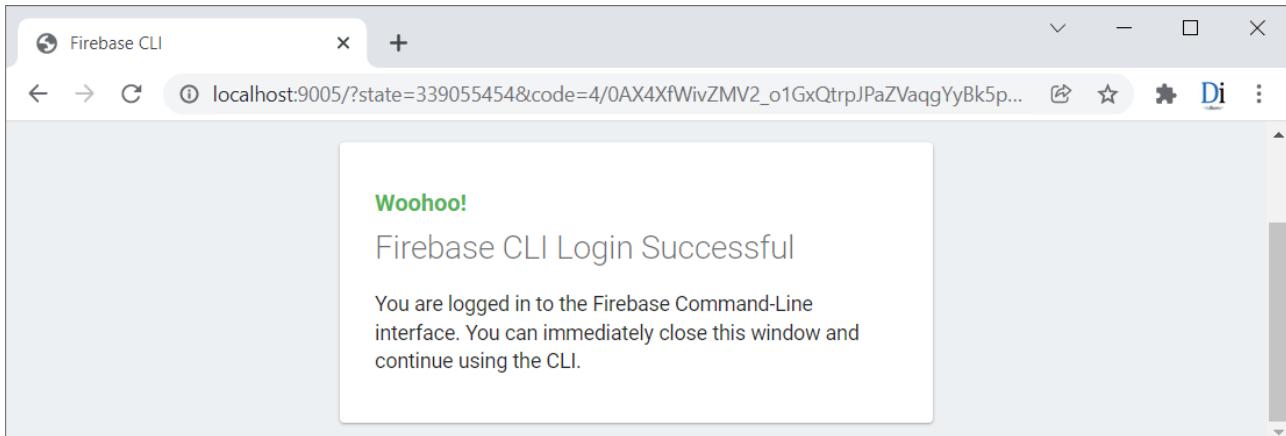
Sesaat kemudian web browser otomatis terbuka dan kita bisa pilih akun Google yang akan gunakan (jika anda memiliki beberapa akun). Pastikan itu adalah akun yang dipakai saat membuat project Firebase sebelumnya.

Di halaman "Firebase CLI wants to access your Google Account", klik tombol "**Allow**":



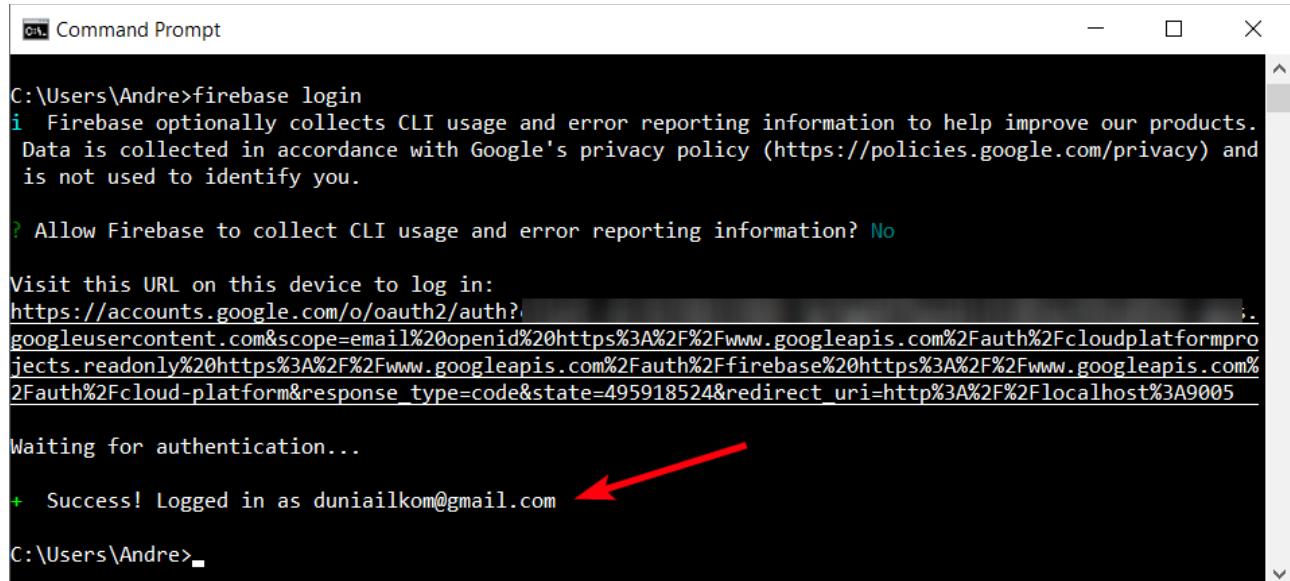
Gambar: Izinkan Firebase CLI untuk mengakses akun Google kita

Jika tidak ada masalah, akan tampil halaman konfirmasi berikut:



Gambar: Proses menghubungkan Firebase CLI sudah berhasil

Buka kembali jendela cmd, dan sekarang akan tampil teks "Success! Logged in as <alamat_email_anda@gmail.com>" sebagai tanda kalau Firebase CLI sudah terhubung ke akun Google yang kita pakai.



```
C:\Users\Andre>firebase login
i  Firebase optionally collects CLI usage and error reporting information to help improve our products.
Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and
is not used to identify you.

? Allow Firebase to collect CLI usage and error reporting information? No

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?...
googleusercontent.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=495918524&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+ Success! Logged in as duniailkom@gmail.com
```

Gambar: Firebase CLI sukses terhubung ke akun duniailkom@gmail.com

Firebase CLI akan terus terhubung selama beberapa waktu, meskipun jendela cmd sudah ditutup dan komputer di restart. Jika anda ingin keluar atau ingin masuk dengan akun lain, harus logout terlebih dahulu dengan perintah `firebase logout`. Kemudian login kembali dengan langkah yang sama seperti di atas.

20.4. Upload File Ilkoom Movie Database

Firebase CLI sudah berhasil di install dan kita sudah login, sekarang saatnya mengupload file.

Hal pertama yang harus di siapkan adalah (tentu saja) aplikasi React yang akan di upload. Pastikan aplikasi itu sudah final, atau jika masih setengah jadi juga tidak masalah selama tidak error pada saat dijalankan dengan perintah `npm start`.

Dalam praktek ini saya ingin mengupload file Ilkoom Movie Database yang baru saja kita buat pada bab sebelumnya. Apabila folder ini terlanjur dihapus, bisa install kembali `create react app` dan copy kode program dari `belajar_react.zip` yang tersedia di Google Drive.

Setelah itu, buka terminal atau cmd dan masuk ke folder tempat aplikasi React berada. Folder yang saya gunakan ada di `D:\belajar_react\my-app\`.



```
C:\Users\Andre>D:
D:\>cd belajar_react\my-app
D:\belajar_react\my-app>
```

Gambar: Pindahkan directory cmd ke folder tempat aplikasi React berada

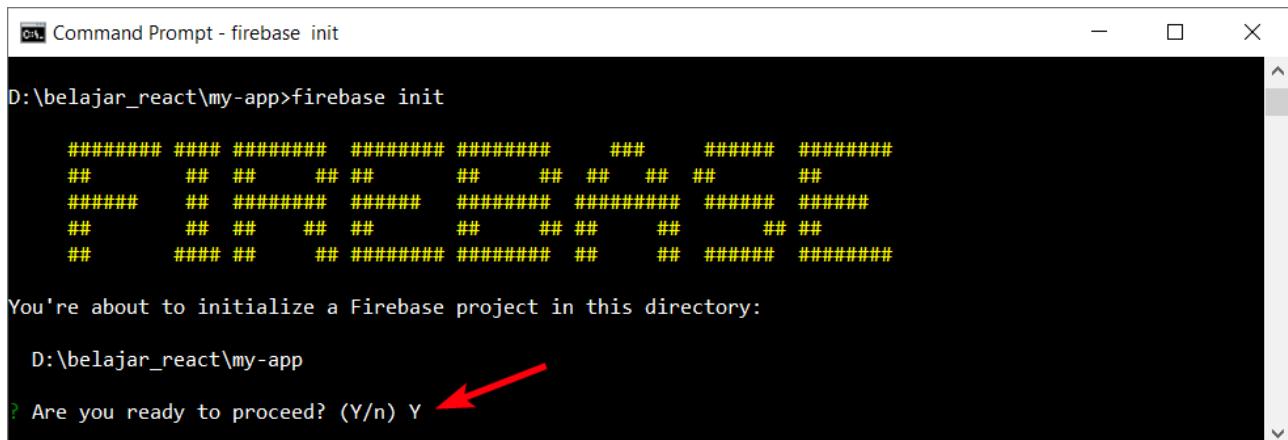
Mengonlinekan React App (Firebase Hosting)

Selanjutnya kita bisa mulai membuat file konfigurasi Firebase CLI dengan mengetik perintah berikut (akhiri dengan tombol enter):

```
firebase init
```

Terdapat 7 pertanyaan yang akan diajukan oleh Firebase CLI.

Pertanyaan pertama berisi konfirmasi apakah kita ingin membuat Firebase project di folder saat ini, yaitu D:\belajar_react\my-app. Jawab Y dan tekan enter.



```
Command Prompt - firebase init
D:\belajar_react\my-app>firebase init

#####
# # # # #
#####
# # # # #
#####
# # # # #
#####
# # # # #

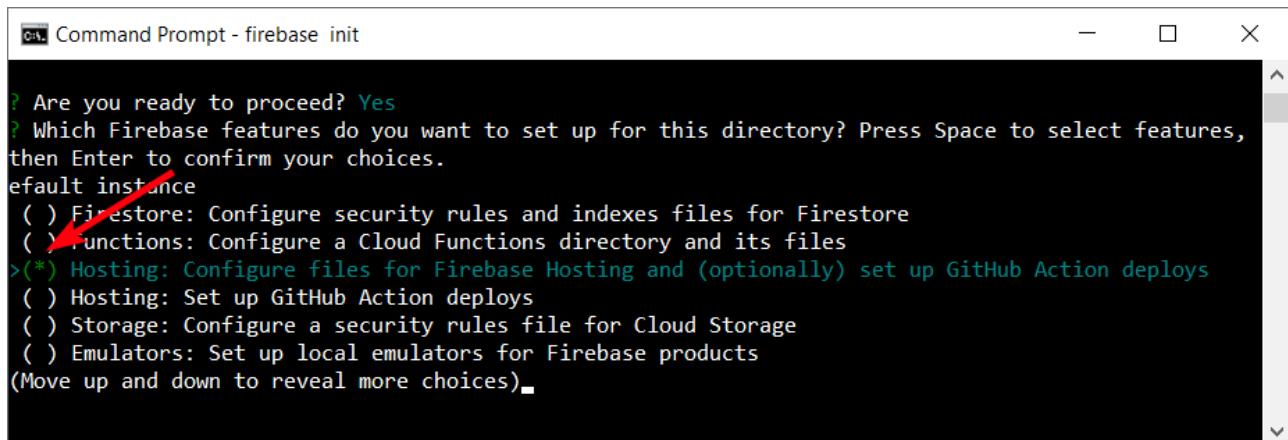
You're about to initialize a Firebase project in this directory:

D:\belajar_react\my-app

? Are you ready to proceed? (Y/n) Y
```

Gambar: Konfirmasi folder untuk Firebase project

Pertanyaan kedua berisi pilihan fitur Firebase yang ingin kita gunakan. Tekan tanda panah atas atau bawah dan cari pilihan "**Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys**". Tekan tombol space di keyboard untuk memilih yang akan ditandai dengan tanda bintang "(*")", lalu akhiri dengan tombol enter:

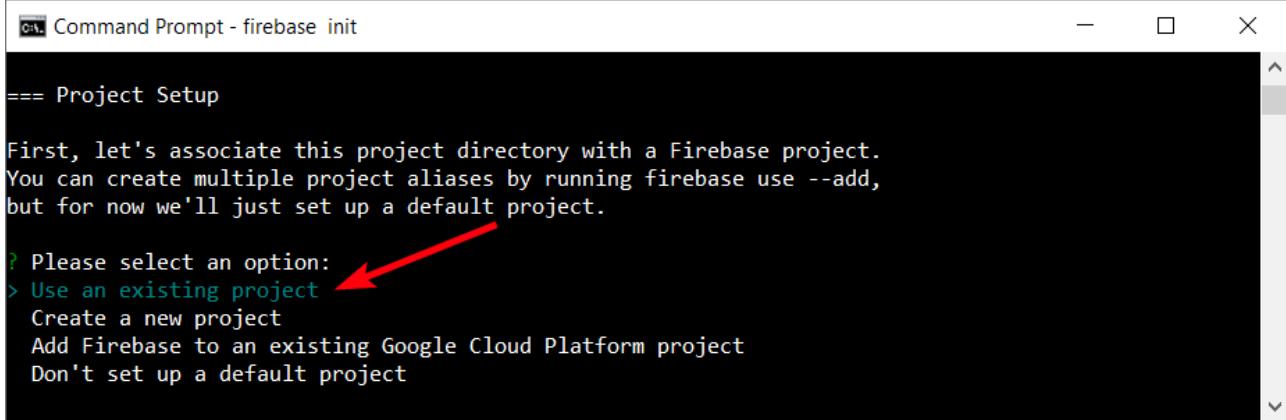


```
Command Prompt - firebase init
? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to select features,
then Enter to confirm your choices.
default instance
( ) Firestore: Configure security rules and indexes files for Firestore
( ) Functions: Configure a Cloud Functions directory and its files
>(*) Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
( ) Hosting: Set up GitHub Action deploys
( ) Storage: Configure a security rules file for Cloud Storage
( ) Emulators: Set up local emulators for Firebase products
(Move up and down to reveal more choices).
```

Gambar: Pilih fitur Hosting dari Firebase

Pertanyaan ketiga berupa konfirmasi apakah akan menggunakan project yang sudah ada atau membuat project baru. Karena kita sudah membuat project tersebut di Firebase console, maka pilih "**Use an existing project**". Sama seperti sebelumnya, gunakan tanda panah untuk menukar pilihan, lalu tekan enter di pilihan "Use an existing project":

Mengonlinekan React App (Firebase Hosting)



```
Command Prompt - firebase init
== Project Setup

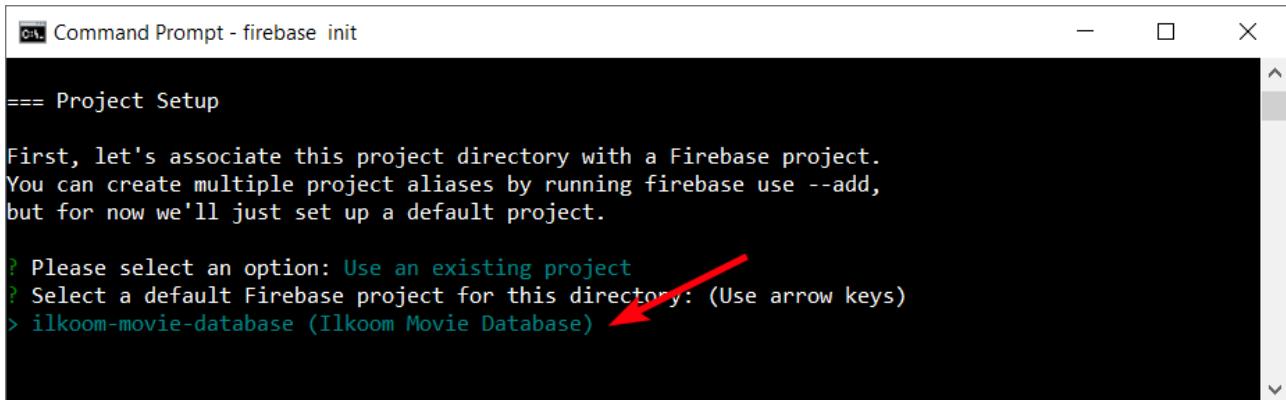
First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option:
> Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project
```

Gambar: Pilih "Use an existing project" karena kita sudah membuat project via Firebase Console

Pertanyaan keempat untuk menentukan nama project yang ingin dipakai. Karena kita baru membuat 1 project, pilihan project ini hanya ada 1, yaitu "**ilkoom-movie-database (Ilkoom Movie Database)**". Jika anda memiliki project lain di akun Firebase, daftar project tersebut juga akan tampil di pilihan ini.

Pastikan memilih project yang akan dipakai, lalu tekan enter:



```
Command Prompt - firebase init
== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

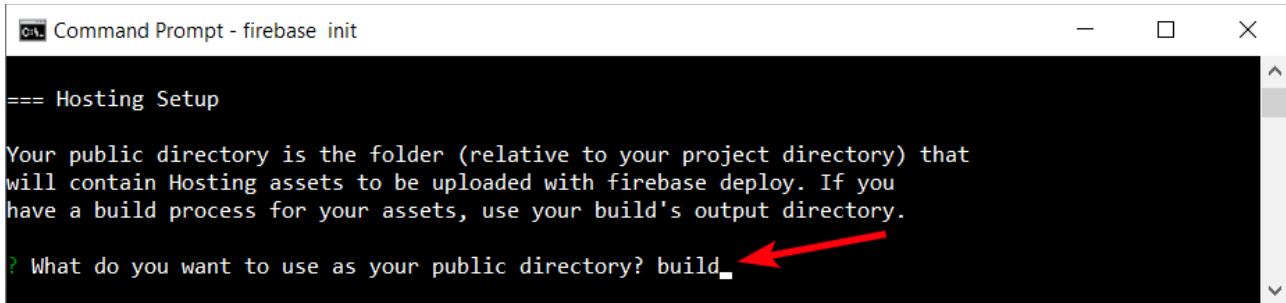
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: (Use arrow keys)
> ilkoom-movie-database (Ilkoom Movie Database)
```

Gambar: Pilih nama project yang akan dipakai

Pertanyaan kelima untuk menentukan nama folder public yang akan diupload (*public directory*). Secara default pertanyaan ini sudah berisi "(*public*)", akan tetapi ini **bukan** folder yang kita ingin upload. Alasannya karena folder *public* di *create react app* hanya berisi file template HTML, sedangkan file React ada di folder *src*.

Maka nama folder yang ingin kita upload adalah **build**, yakni folder hasil proses compile *create react app*. Saat ini folder *build* memang belum ada, tapi itu tidak masalah dan akan kita buat sesaat lagi. Hapus teks "(*public*)", lalu ganti dengan **build**. Tekan enter untuk melanjutkan:

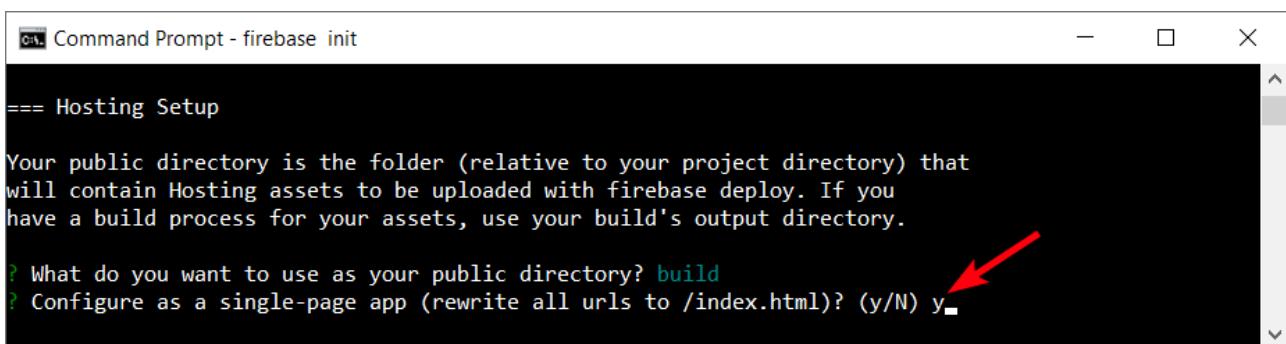
Mengonlinekan React App (Firebase Hosting)



```
Command Prompt - firebase init
--- Hosting Setup
Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.
? What do you want to use as your public directory? build
```

Gambar: Isi nama public directory dengan "build" (tanpa tanda kutip)

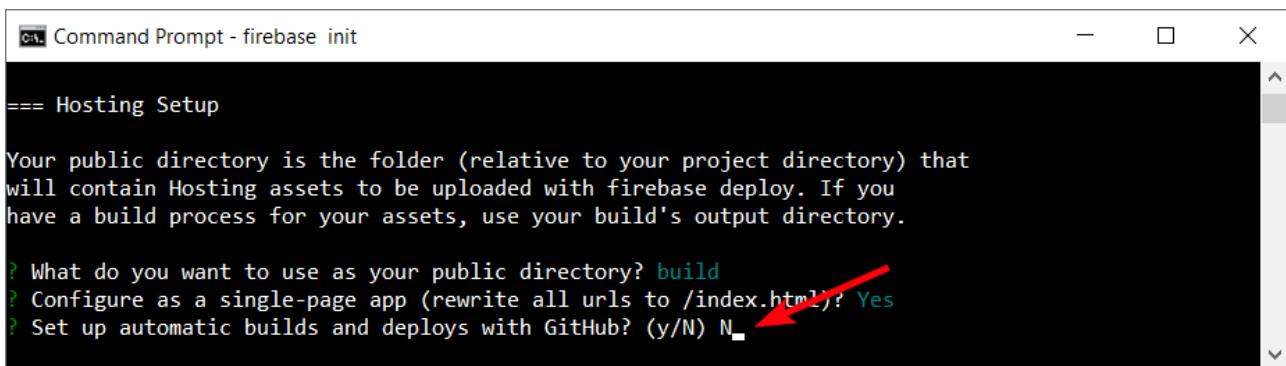
Pertanyaan keenam, apakah kita ingin membuat single-page app (SPA)? Karena Ilkoom Movie Database memang hanya berisi 1 halaman, maka jawab dengan mengetik huruf **y**, lalu tekan tombol enter:



```
Command Prompt - firebase init
--- Hosting Setup
Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.
? What do you want to use as your public directory? build
? Configure as a single-page app (rewrite all urls to /index.html)? (y/N) y
```

Gambar: Jawab **y** karena kita memang akan mengupload SPA

Pertanyaan ketujuh dan sekaligus pertanyaan terakhir, apakah kita ingin mengupload file ke github? Ini sepenuhnya opsional (tidak wajib), oleh karena itu saya akan jawab dengan **N** dan tekan enter:

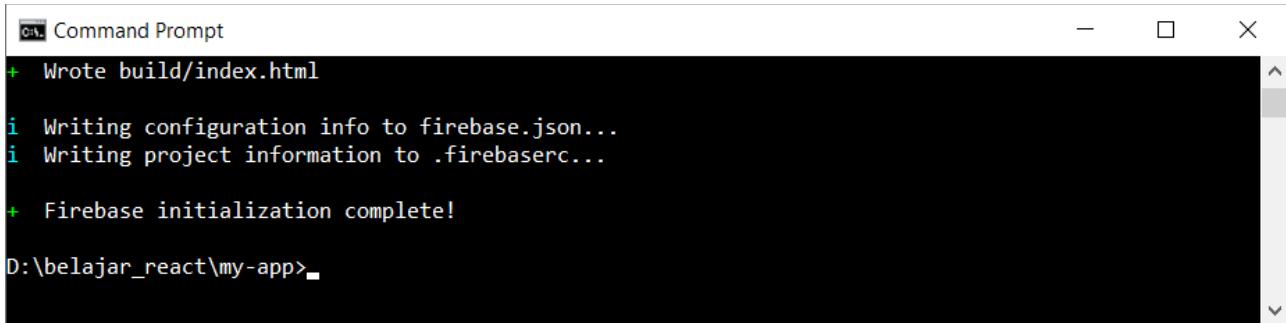


```
Command Prompt - firebase init
--- Hosting Setup
Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.
? What do you want to use as your public directory? build
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? (y/N) N
```

Gambar: Jawab **N** agar file tidak diupload ke github

Proses konfigurasi Firebase project sudah selesai. Di akhir layar cmd akan tampil pesan "Firebase initialization complete!".

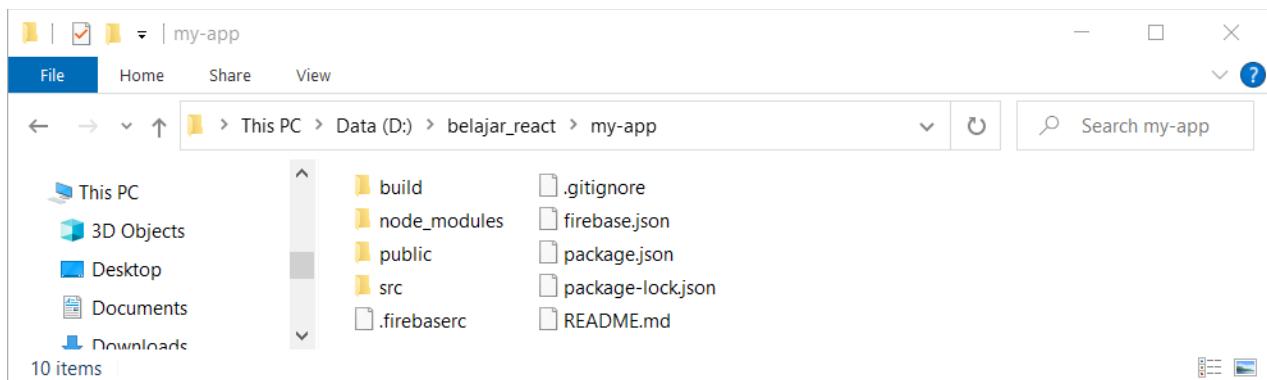
Mengonlinekan React App (Firebase Hosting)



```
Command Prompt
+ Wrote build/index.html
i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...
+ Firebase initialization complete!
D:\belajar_react\my-app>
```

Gambar: Proses Firebase initialization sudah selesai

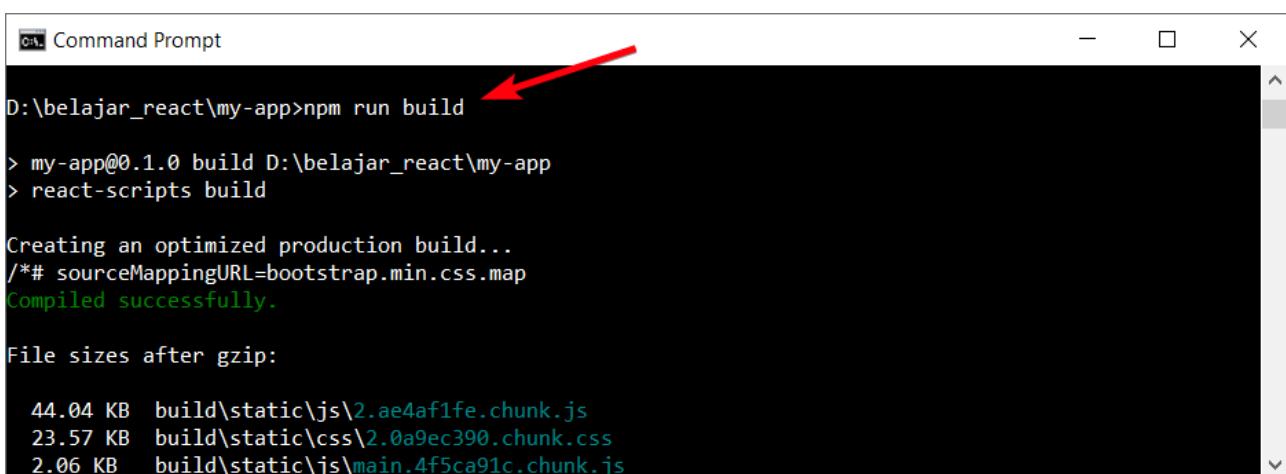
Jika anda membuka folder D:\belajar_react\my-app, akan terlihat 2 file baru, yakni `firebase.json` dan `.firebaserc`. Keduanya berisi data konfigurasi dari Firebase dan tidak perlu kita utak-atik. Selain itu juga ada folder `build` yang berisi `index.html` sebagai file dummy Firebase.



Gambar: Isi folder D:\belajar_react\my-app saat ini

Langkah berikutnya, meng-compile aplikasi React dengan perintah:

```
npm run build
```



```
Command Prompt
D:\belajar_react\my-app>npm run build
> my-app@0.1.0 build D:\belajar_react\my-app
> react-scripts build

Creating an optimized production build...
/*# sourceMappingURL=bootstrap.min.css.map
Compiled successfully.

File sizes after gzip:
 44.04 KB  build\static\js\2.ae4af1fe.chunk.js
 23.57 KB  build\static\css\2.0a9ec390.chunk.css
  2.06 KB  build\static\js\main.4f5ca91c.chunk.js
```

Gambar: Proses compile aplikasi React dengan perintah `npm run build`

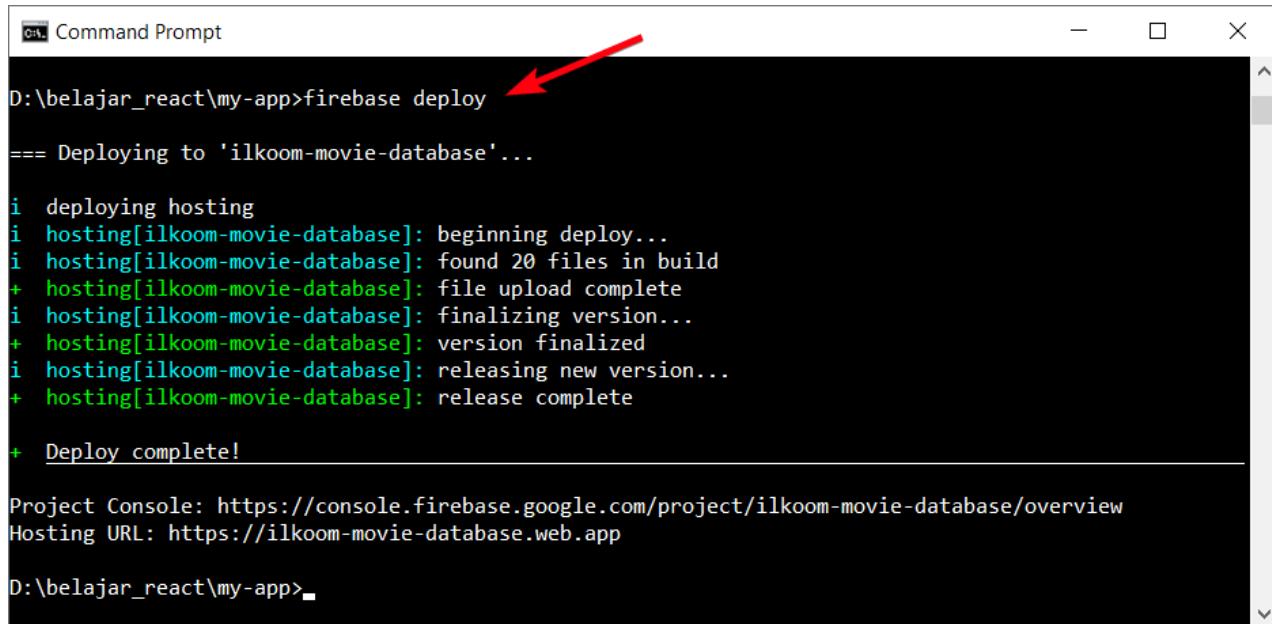
Perintah ini sudah pernah kita bahas dan merupakan bawaan `create react app`. Hasilnya, folder `build` akan dibuat ulang dan berisi aplikasi React final yang siap di upload.

Mengonlinekan React App (Firebase Hosting)

Daan... kita sampai ke langkah terakhir, yakni:

```
firebase deploy
```

Ketik perintah di atas dan akhiri dengan tombol enter:



```
Command Prompt
D:\belajar_react\my-app>firebase deploy
=== Deploying to 'ilkoom-movie-database'...
i  deploying hosting
i  hosting[ilkoom-movie-database]: beginning deploy...
i  hosting[ilkoom-movie-database]: found 20 files in build
+  hosting[ilkoom-movie-database]: file upload complete
i  hosting[ilkoom-movie-database]: finalizing version...
+  hosting[ilkoom-movie-database]: version finalized
i  hosting[ilkoom-movie-database]: releasing new version...
+  hosting[ilkoom-movie-database]: release complete

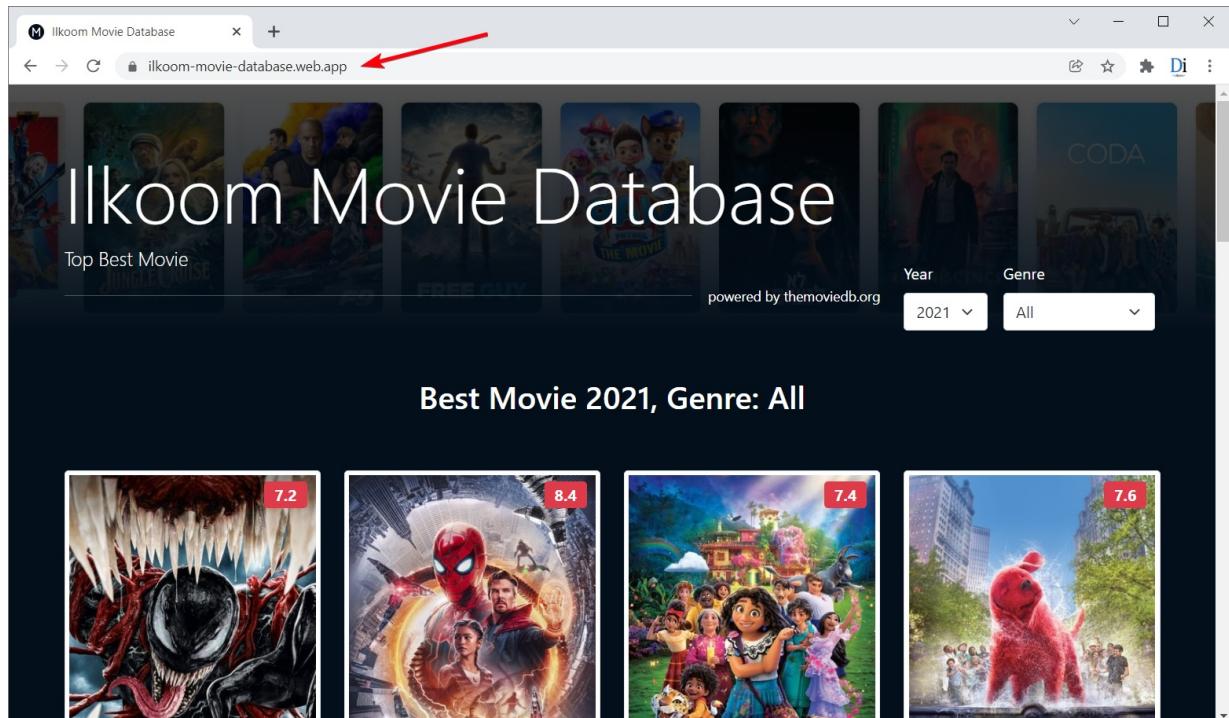
+ Deploy complete!

Project Console: https://console.firebaseio.google.com/project/ilkoom-movie-database/overview
Hosting URL: https://ilkoom-movie-database.web.app

D:\belajar_react\my-app>
```

Gambar: Proses upload file ke Firebase Hosting

Perintah ini akan mengupload seluruh isi folder build ke dalam Firebase hosting. Di bagian akhir juga tampil alamat Hosting URL dari aplikasi kita, yang dalam contoh ini ada di <https://ilkoom-movie-database.web.app>. Untuk memastikan, silahkan buka alamat di web browser:



Gambar: Aplikasi React sudah "online" di internet

Mengonlinekan React App (Firebase Hosting)

Done! aplikasi kita sudah bisa diakses dari internet.

Itulah panduan cara mengupload file React ke internet menggunakan fitur hosting gratis dari Firebase. Jika nantinya aplikasi React perlu perbaikan, kita bisa ubah dengan cara biasa secara offline. Setelah selesai, jalankan ulang `npm run build` dan upload kembali dengan perintah `firebase deploy`. File lama yang ada di firebase hosting akan ditimpa dengan file baru.

Sebagai bahan praktek, saya ingin tukar judul aplikasi dari "Ilkoom Movie Database" menjadi "My Movie Database". Judul ini ada di 2 tempat, pertama di file `my-app/public/index.html`

`my-app/public/index.html`

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8" />
6      <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
7      <meta name="viewport" content="width=device-width, initial-scale=1" />
8      <meta name="theme-color" content="#000000" />
9      <meta name="description" content="My Movie Database - 
10     React Uncover DuniaIlkom" />
11     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
12     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
13     <title>My Movie Database</title>
14 </head>
15
16 <body>
17     <noscript>You need to enable JavaScript to run this app.</noscript>
18     <div id="root"></div>
19 </body>
20
21 </html>
```

Perubahannya ada di baris 9 untuk tag `<meta>` dan di baris 13 untuk tag `<title>`. File kedua yang harus di modif ada di `my-app/src/components/Header.js`:

`my-app/src/components/Header.js`

```
1  const Header = () => {
2
3      return (
4          <header className="text-white py-5">
5              <div className="container">
6                  <div className="row">
7                      <div className="col-12">
8                          <h1 className="display-1">My Movie Database</h1>
9                          <h3 className="lead mb-0">Top Best Movie</h3>
10                     </div>
11                 </div>
12             </div>
13         </header>
14     )
```

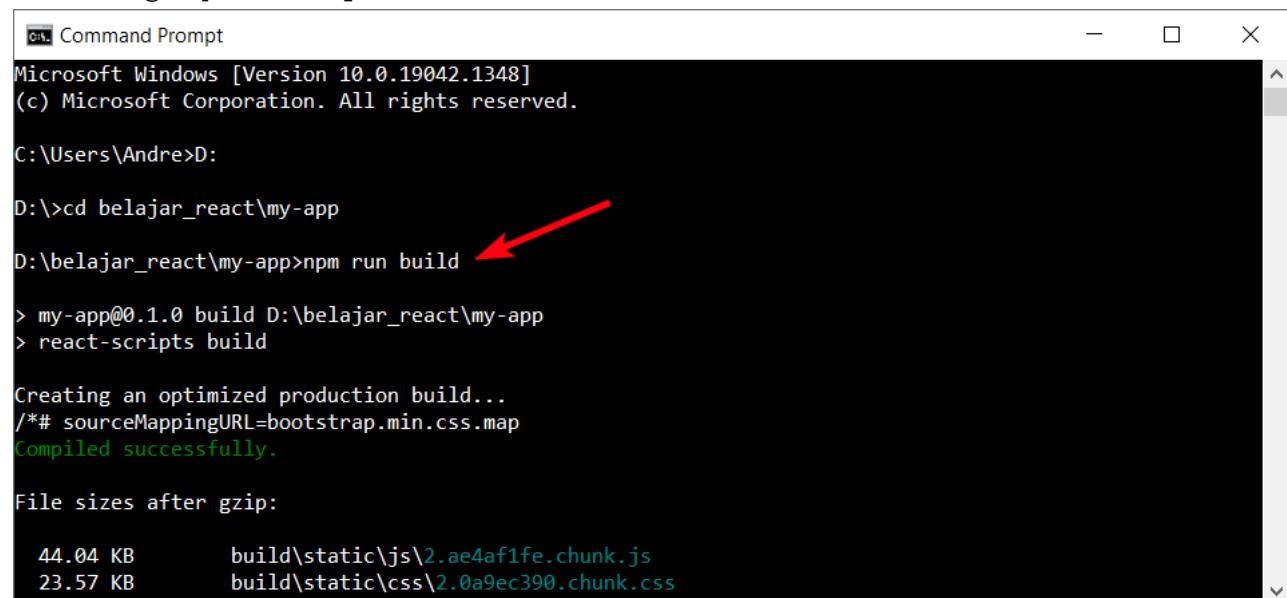
Mengonlinekan React App (Firebase Hosting)

```
15 }
16
17 export default Header;
```

Perubahan ada di baris 8, yakni tag <h1> yang sekarang berisi "My Movie Database". Simpan perubahan ini dan pastikan tidak ada error dengan menjalankan **npm start**.

Agar lebih lebih real, silahkan tutup jendela cmd yang kita pakai sebelumnya (untuk upload file firebase). Ini sebagai simulasi bahwa perubahan kita lakukan di kemudian hari.

Lalu buka jendela cmd baru, masuk ke folder D:\belajar_react\my-app, dan compile ulang file React dengan perintah **npm run build**:



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

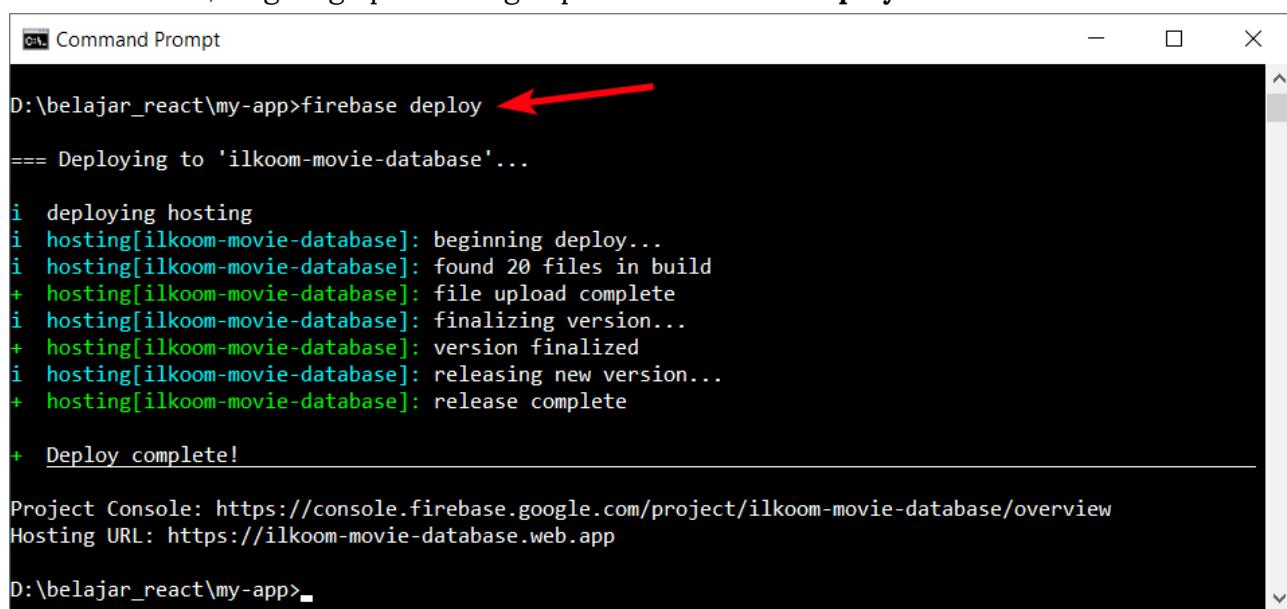
C:\Users\Andre>D:
D:>cd belajar_react\my-app
D:\belajar_react\my-app>npm run build →
> my-app@0.1.0 build D:\belajar_react\my-app
> react-scripts build

Creating an optimized production build...
/*# sourceMappingURL=bootstrap.min.css.map
Compiled successfully.

File sizes after gzip:
  44.04 KB      build\static\js\2.ae4af1fe.chunk.js
  23.57 KB      build\static\css\2.0a9ec390.chunk.css
```

Gambar: Compile ulang dengan perintah npm run build

Setelah selesai, langsung upload dengan perintah **firebase deploy**:



```
Command Prompt
D:\belajar_react\my-app>firebase deploy →
=== Deploying to 'ilkoom-movie-database'...

i  deploying hosting
i  hosting[ilkoom-movie-database]: beginning deploy...
i  hosting[ilkoom-movie-database]: found 20 files in build
+  hosting[ilkoom-movie-database]: file upload complete
i  hosting[ilkoom-movie-database]: finalizing version...
+  hosting[ilkoom-movie-database]: version finalized
i  hosting[ilkoom-movie-database]: releasing new version...
+  hosting[ilkoom-movie-database]: release complete

+  Deploy complete!

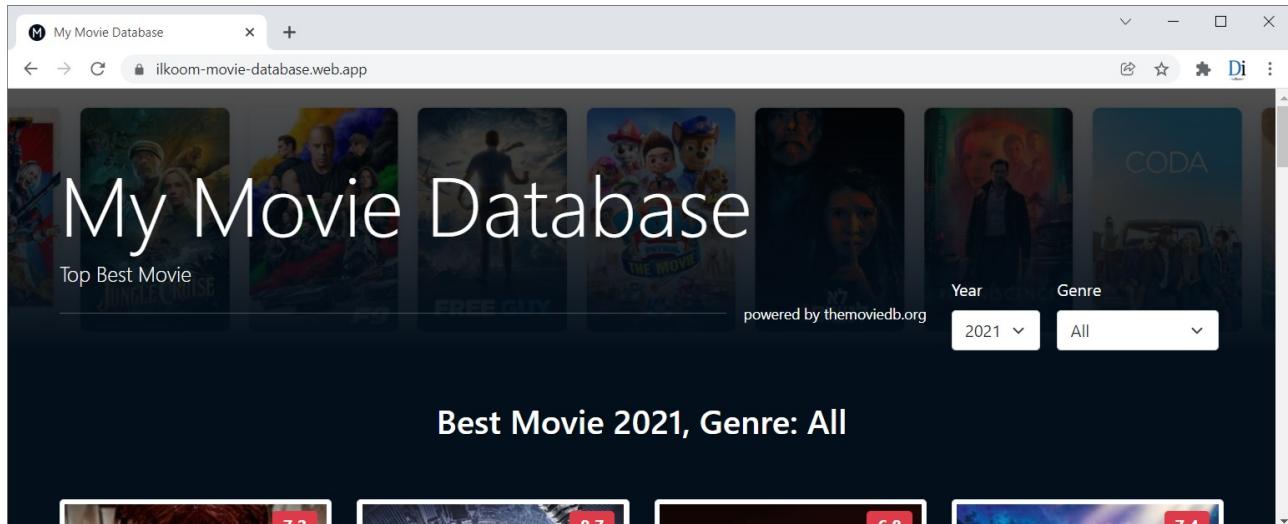
Project Console: https://console.firebaseio.google.com/project/ilkoom-movie-database/overview
Hosting URL: https://ilkoom-movie-database.web.app

D:\belajar_react\my-app>
```

Gambar: Upload file React dengan perintah firebase deploy

Mengonlinekan React App (Firebase Hosting)

Hasilnya, tampilan halaman web juga sudah berubah:



Gambar: Judul aplikasi sudah berubah menjadi My Movie Database

Inilah yang saya maksud bahwa Firebase CLI sangat praktis dalam proses pengembangan aplikasi. Kita tidak perlu upload file secara manual di web browser, cukup ketik 2 perintah singkat dari cmd. Firebase CLI mengingat kenapa harus mengupload berdasarkan file `firebase.json` dan `.firebaserc` yang sudah ada di folder `my-app`.

Jika kita buka kembali Firebase Console di web browser dan masuk ke halaman Hosting, disitu bisa terlihat informasi tambahan, diantaranya alamat URL dari aplikasi kita:

Gambar: Halaman hosting di Firebase console

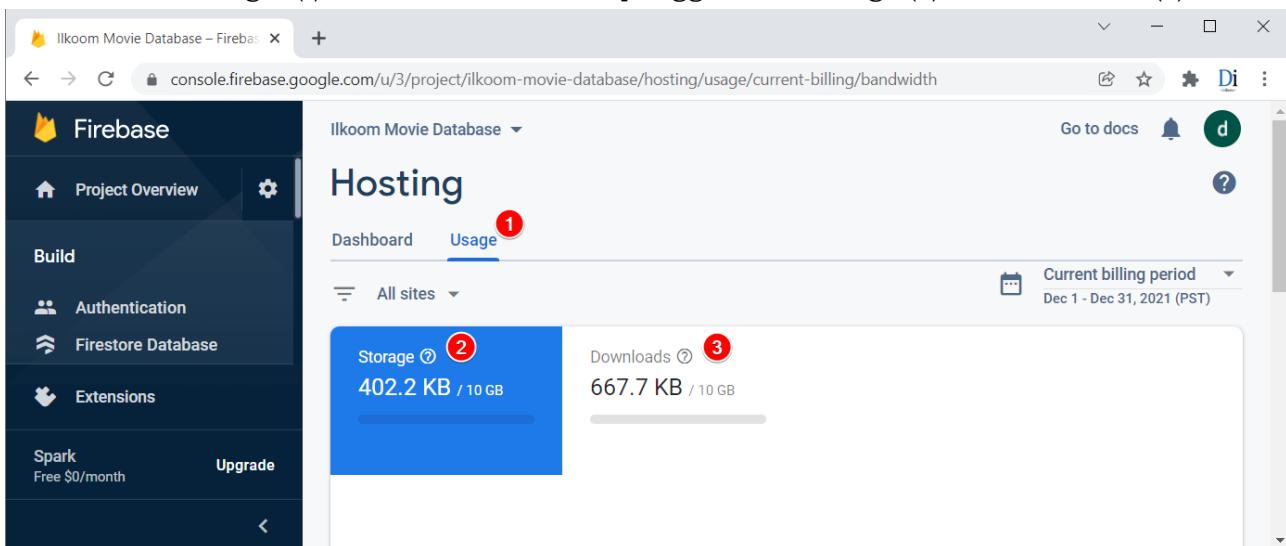
Di bagian atas terlihat alamat URL yang disediakan Firebase ternyata 2 buah, yakni dengan domain `*.web.app`, serta `*.firebaseapp.com`. Sehingga project Ilkoom Movie Database bisa

Mengonlinekan React App (Firebase Hosting)

diakses dari <https://ilkoom-movie-database.web.app> dan juga dari <https://ilkoom-movie-database.firebaseio.com>.

Untuk keperluan yang lebih serius, terdapat tombol "Add custom domain", sehingga kita bisa menggunakan domain sendiri seperti domainku.com. Syaratnya, domain tersebut harus sudah disewa sebelumnya.

Di dalam tab "Usage" (1), bisa terlihat batasan penggunaan Storage (2) dan Download (3):



Gambar: Penggunaan storage dan downloads dari Firebase Hosting

Untuk Firebase hosting versi gratis, storage dibatasi maksimum 10GB. Dalam contoh di atas, aplikasi Ilkoom Movie Database hanya memakan ruang 402kb saja.

Sedangkan untuk Download, dibatasi sebanyak 10BG per bulan. Ini sebenarnya merujuk ke kapasitas bandwidth, yakni lalu lintas data saat web kita di buka. Firebase juga hanya mengizinkan bandwidth sebanyak 360MB/hari, sehingga dapatlah hitungan $360 \times 30 = 10\text{GB}$ per bulan.

Jadi jika di satu hari web kita sudah menggunakan bandwidth lebih dari 360MB, web tersebut baru bisa diakses lagi keesokan harinya.

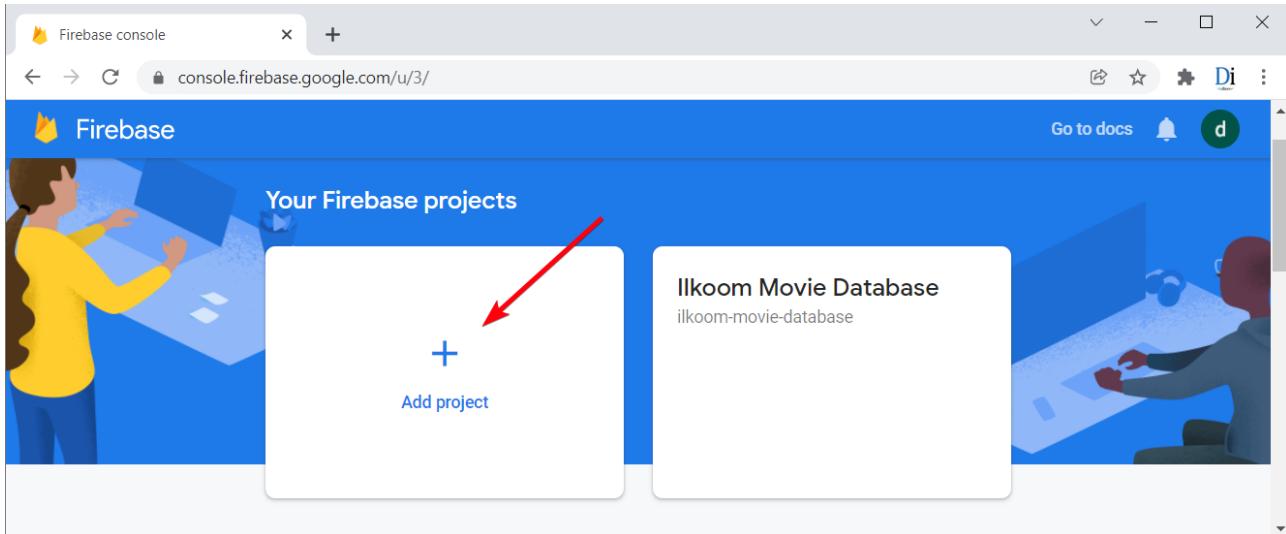
20.5. Upload File Ilkoom Expense Tracker

Tidak cukup dengan Ilkoom Movie Database, saya juga ingin mengonlinekan aplikasi Ilkoom Expense Tracker. Cara yang dipakai kurang lebih sama, yakni buat project baru di Firebase Console, jalankan Firebase CLI, build aplikasi, dan deploy.

Silahkan buka kembali Firebase Console di alamat console.firebaseio.google.com, atau bisa juga klik logo Firebase di kiri atas jika anda berada di halaman lain.

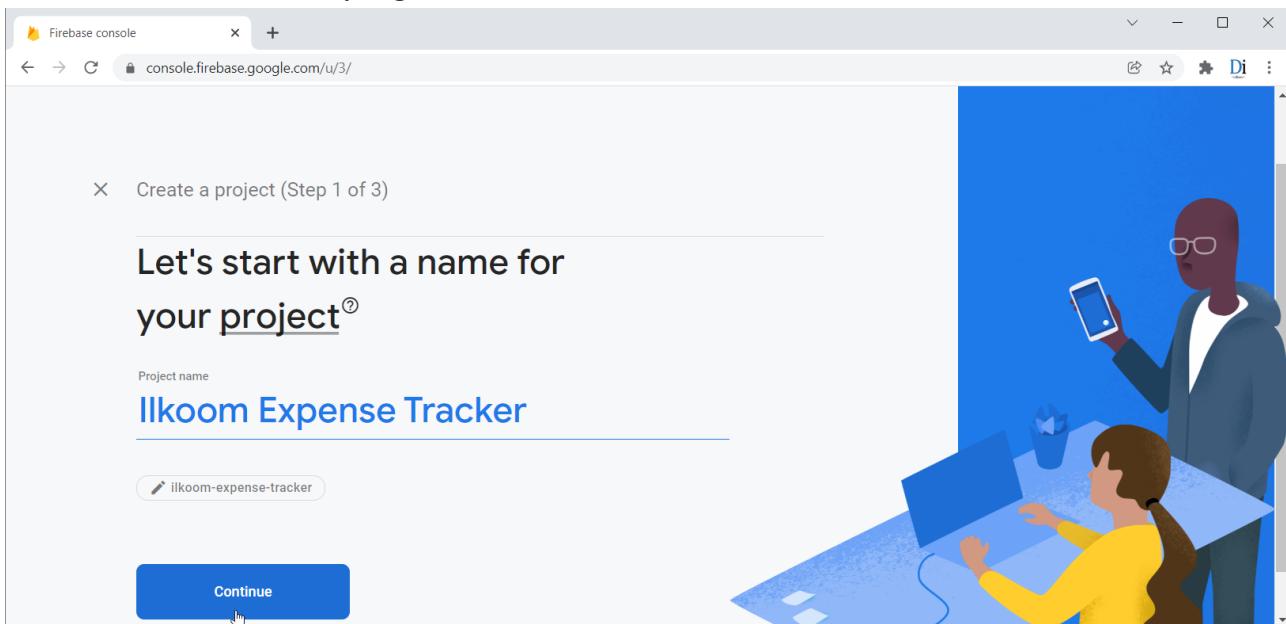
Di halaman home, sekarang terdapat pilihan Firebase project yang sudah ada. Untuk membuat project baru, klik "Add project":

Mengonlinekan React App (Firebase Hosting)



Gambar: Klik "Add project" untuk membuat Firebase project

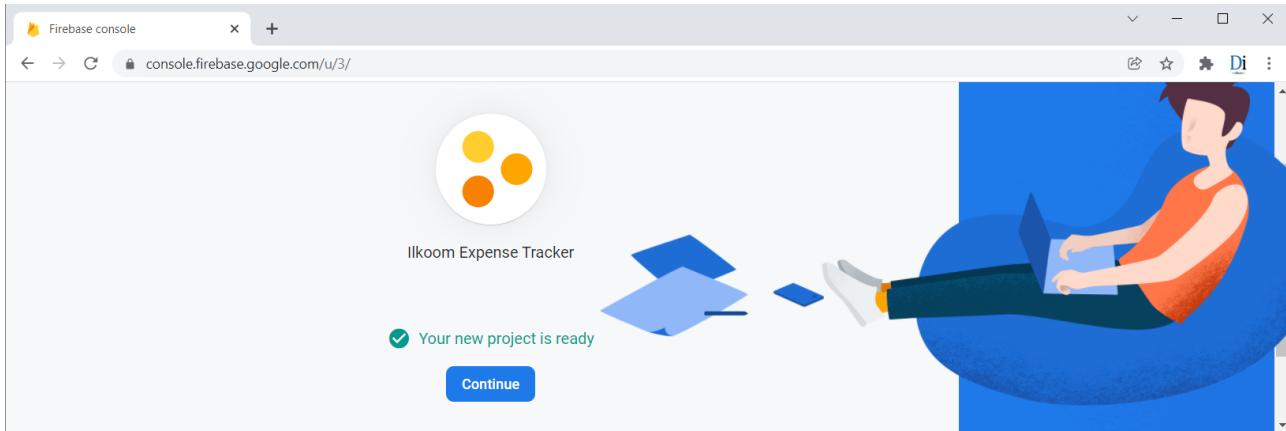
Kita kembali mengisi form nama project. Kali ini saya akan input "**Ilkoom Expense Tracker**". Anda bisa memilih nama yang sama atau bisa memakai nama lain. Setelah itu klik continue:



Gambar: Buat project "Ilkoom Expense Tracker"

Di halaman berikutnya kembali off-kan "Enable Google Analytics", lalu klik tombol "**Create Project**". Tunggu beberapa saat dan project siap digunakan.

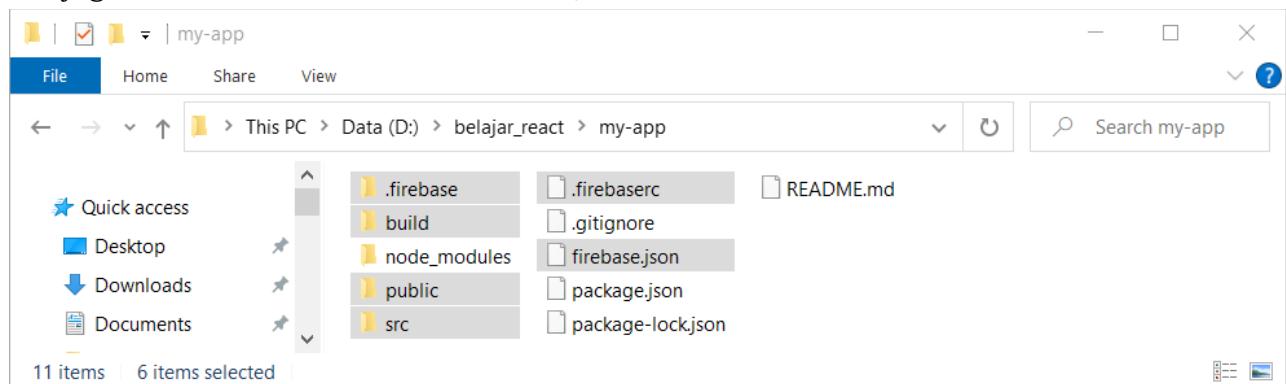
Mengonlinekan React App (Firebase Hosting)



Gambar: Project "Ilkoom Expense Tracker" siap digunakan

Sebelum lanjut ke Firebase CLI, kita harus siapkan folder create react app yang berisi aplikasi React. Cara ideal, silahkan buat folder baru agar project sebelumnya tidak tertimpas. Folder `public` dan `src` untuk Ilkoom Expense Tracker bisa dicopy dari file `belajar_react.zip`.

Cara yang lebih cepat (meskipun kurang disarankan), adalah memakai ulang folder create react app yang sudah ada. Buka folder `my-app`, lalu hapus folder `public`, `src`, `build`, `.firebase`, dan juga file `.firebaserc` serta `firebase.json`.



Gambar: Hapus 4 folder dan 2 file yang ada di dalam my-app

Dengan menghapus semua file ini, folder `my-app` hanya tinggal `node_modules` serta beberapa file tambahan saja. Setelah itu silahkan copy file `public` dan `src` dari `belajar_react.zip` dan aplikasi kita siap digunakan.

Cara ini memang lebih cepat daripada menginstall create react app baru, akan tetapi karena kita sudah menghapus file konfigurasi Firebase, maka nanti perlu membuat ulang lagi jika ingin mengupdate project sebelumnya.

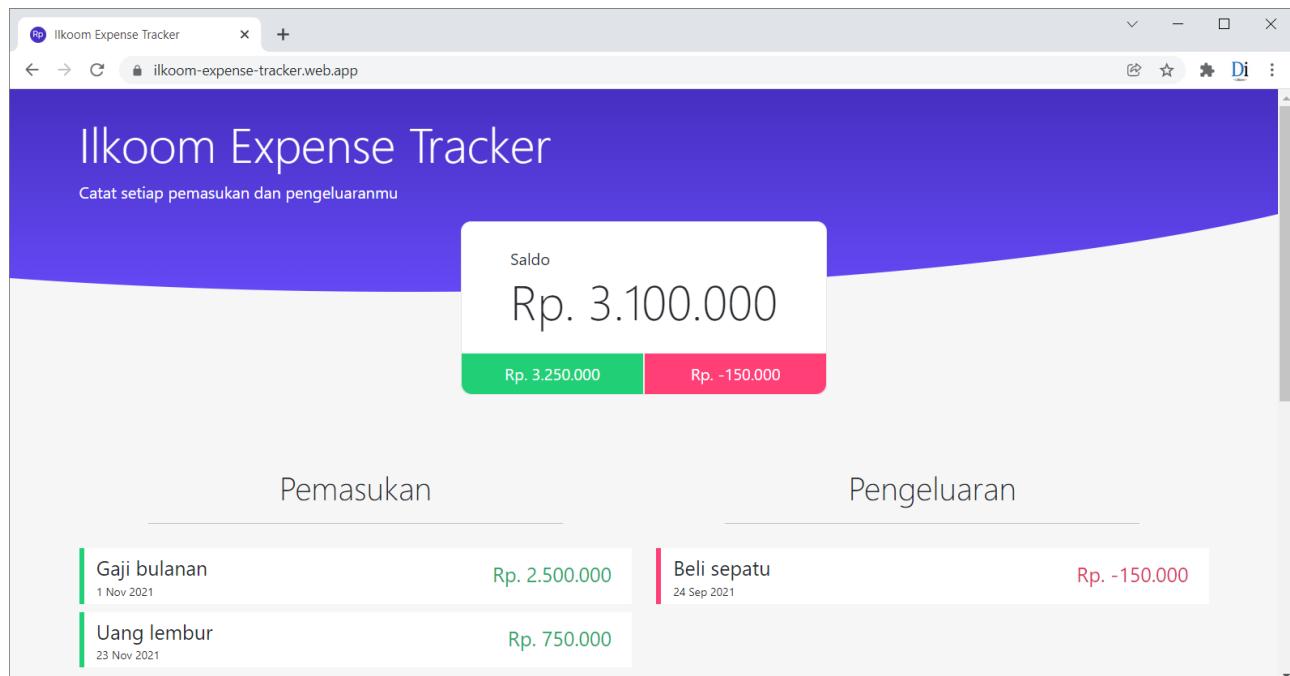
Setelah memastikan file React sudah tidak error (tes dahulu dengan `npm start`), sekarang saatnya masuk ke Firebase CLI. Langkah yang dipakai sama persis seperti di praktik Ilkoom Movie Database. Berikut rangkuman langkah yang diperlukan:

1. Buka cmd, masuk ke folder create react app berada, misal D:\belajar_react\my-app.

Mengonlinekan React App (Firebase Hosting)

2. Jalankan perintah **firebase init**
3. Are you ready to proceed? **Y**
4. Pilih menu "Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys"
5. Pilih menu "Use an existing project"
6. Pilih project yang akan dipakai, misalnya "ilkoom-expense-tracker (Ilkoom Expense Tracker)"
7. What do you want to use as your public directory? **build**
8. Configure as a single-page app (rewrite all urls to /index.html)? **Y**
9. Set up automatic builds and deploys with GitHub? **N**
10. Compile file create react app dengan perintah **npm run build**
11. Upload file ke Firebase hosting dengan perintah **firebase deploy**

Done! Aplikasi Ilkoom Expense Tracker sudah online di alamat <https://ilkoom-expense-tracker.web.app>.



Gambar: Aplikasi Ilkoom Expense Tracker sudah online

20.6. Mencoba Firebase Realtime Database

Karena sudah terlanjur menggunakan Firebase, kurang pas rasanya jika kita tidak mencoba fitur paling populer dari Firebase, yakni database.

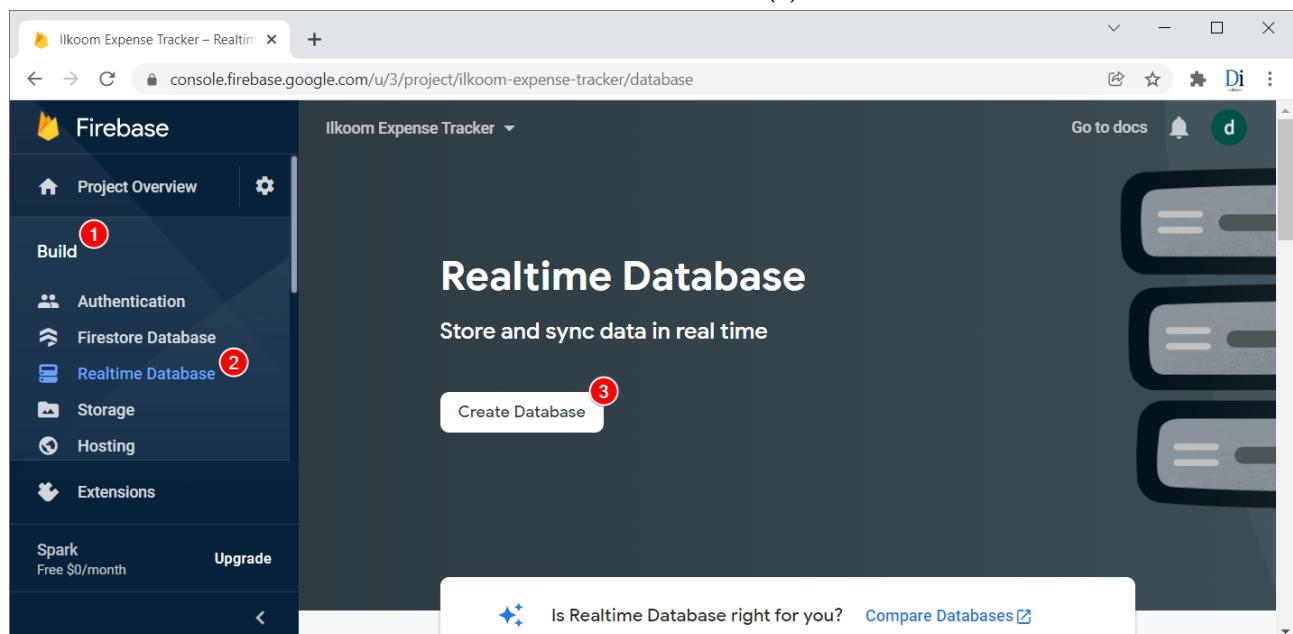
Saat ini Firebase menyediakan 2 jenis database: **Firebase Database** dan **Realtime Database**.

Dalam praktek ini kita akan pakai realtime database karena lebih sederhana, sedangkan firestore database lebih cocok untuk aplikasi kompleks yang butuh fitur offline. Penjelasan lebih lanjut bisa dibaca ke [Choose a Database: Cloud Firestore or Realtime Database²⁹](#).

Salah satu keunggulan *realtime database* adalah sudah menyediakan REST API siap pakai, sehingga data yang didapat nanti sudah langsung berbentuk JSON. Kita akan coba terapkan realtime database ini ke Ilkoom Expense Tracker agar data transaksi bisa disimpan permanen.

Materi ini hanya sekedar "perkenalan" dengan Firebase Realtime Database. Saya tidak akan bahas semua fitur yang ada karena bisa sangat panjang dan rasanya lebih cocok di buku tersendiri.

Untuk memulai praktek, pastikan sudah masuk ke project "Ilkoom Expense Tracker" di firebase console. Setelah itu silahkan klik menu **Build** (1), pilih **Realtime Database** (2), dan setelah halaman terbuka klik tombol "**Create Database**" (3):

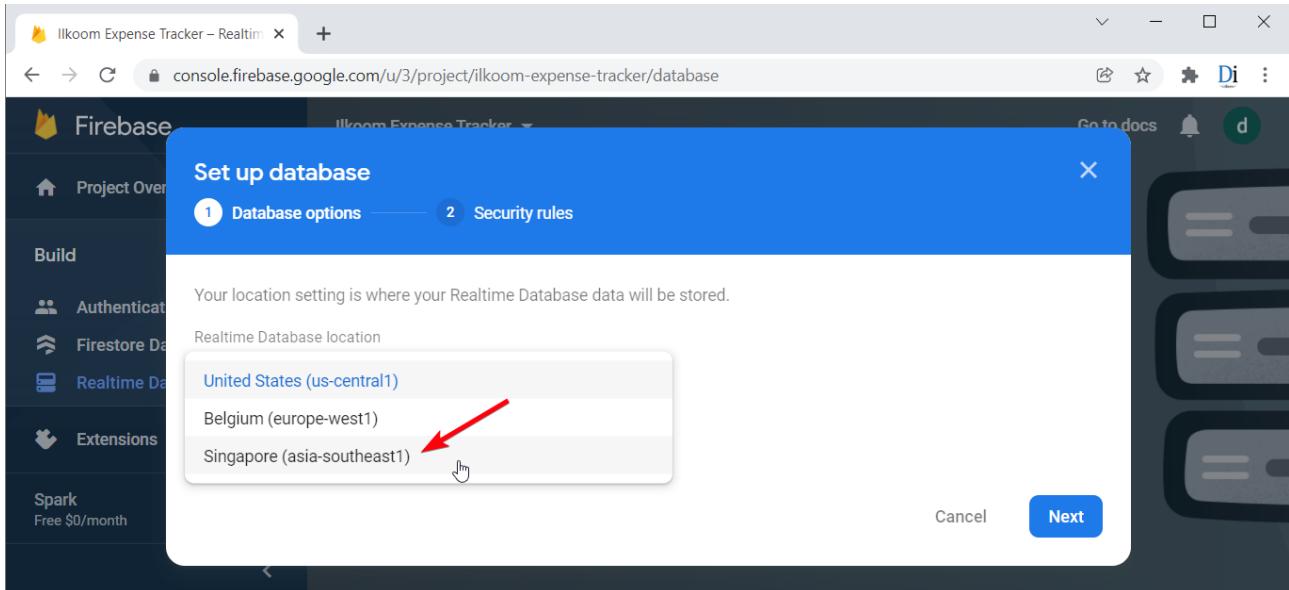


Gambar: Membuat Firebase Realtime Database

Selanjutnya akan tampil jendela popup untuk memilih lokasi server. Kita bisa pilih lokasi mana saja, namun karena lebih dekat dengan Indonesia, silahkan pilih "**Singapore (asia-southeast1)**". Lalu klik tombol **Next:**

29. <https://firebase.google.com/docs/database/rtdb-vs-firebase>

Mengonlinekan React App (Firebase Hosting)

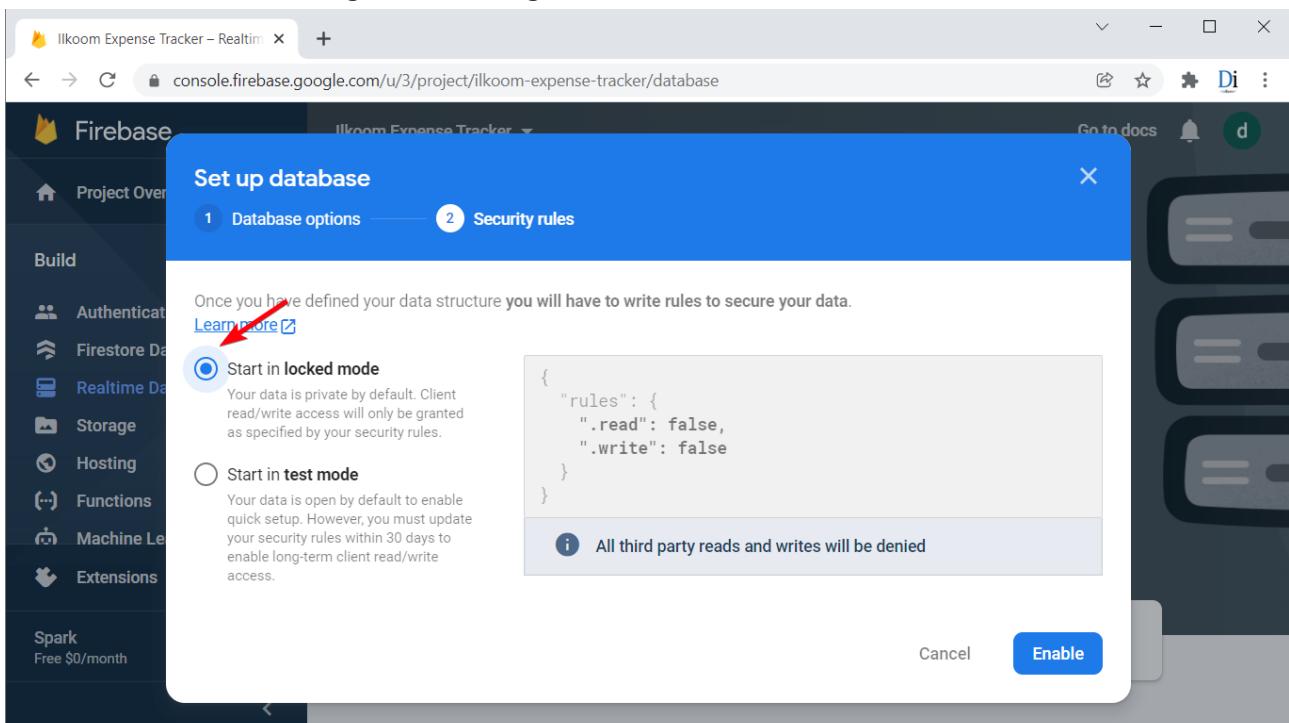


Gambar: Pilih lokasi server di "Singapore (asia-southeast1)"

Jendela berikutnya berisi "Security rules" dengan 2 pilihan: "Start in locked mode" atau "Start in test mode". Keduanya dipakai untuk mengatur hak akses database.

Pilihan "Start in test mode" sebenarnya lebih cocok untuk kebutuhan kita, sebab pengaturan itu akan membuka semua hak akses. Akan tetapi dalam 30 hari, hak akses ini harus di update supaya database tetap bisa diakses.

Agar tidak kerja dua kali, pilih saja "**Start in locked mode**". Pengaturan hak akses akan langsung kita atur di awal sesaat lagi. Akhiri dengan klik tombol **Enable**:



Gambar: Pilih pengaturan "Start in locked mode"

Mengonlinekan React App (Firebase Hosting)

Firebase akan menyiapkan database beberapa saat, dan jika sudah selesai akan tampil halaman dashboard berikut:

The screenshot shows the Firebase Realtime Database dashboard. On the left, there's a sidebar with 'Project Overview' and other services like Authentication, Firestore Database, Realtime Database, and Storage. The main area is titled 'Realtime Database' with tabs for Data, Rules (which has a red circle with '3'), Backups, and Usage. A banner at the top says 'Protect your Realtime Database resources from abuse, such as billing fraud or phishing' with a 'Configure App Check' button. Below the banner, there's a URL field (1) containing 'https://ilkoom-expense-tracker-default-rtbd.firebaseio.com/' and a security rules editor (2) where the current rules are shown as 'ilkoom-expense-tracker-default-rtbd: null'. A red circle with '3' is on the Rules tab itself.

Gambar: Halaman dashboard Realtime Database

Di tengah halaman, terdapat alamat URL (1). Ini merupakan alamat server dari realtime database untuk project kita. Nantinya alamat inilah yang dipakai untuk mengakses API.

Sedikit ke bawah (2), terdapat semacam diagram yang bisa di klik. Ini adalah tempat untuk membuat struktur database secara visual. Jika sebelumnya anda pernah memakai XAMPP, maka ini mirip seperti phpMyAdmin, yakni tempat membuat struktur database berbasis visual.

Pada bagian atas, terdapat tab **Rules** (3) yang berisi pengaturan hak akses. Silahkan klik tab ini untuk mengubah **security rules**.

Di bagian tengah halaman terdapat editor yang berisi kode JSON berikut:

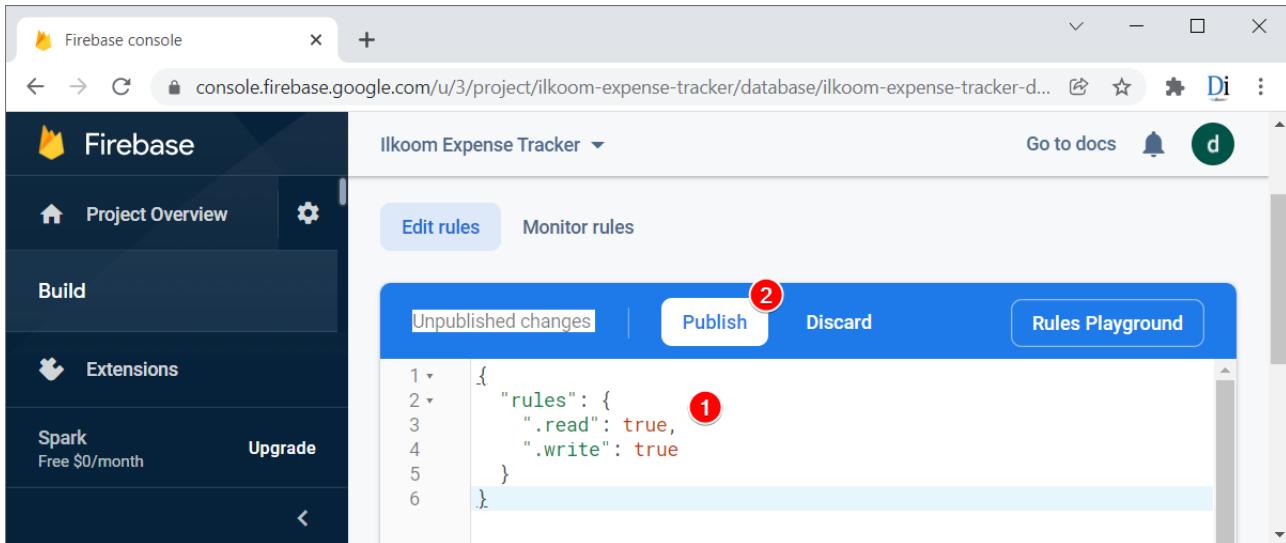
```
{  
  "rules": {  
    ".read": false,  
    ".write": false  
  }  
}
```

Kode ini berarti kita membatasi hak akses *read* dan *write* database untuk semua user. Jika ini tidak diubah, tidak ada aplikasi bisa mengaksesnya. Oleh karena itu modifikasi kedua nilai *false* menjadi **true** (1), lalu klik tombol **Publish** (2):

```
{  
  "rules": {  
    ".read": true,  
    ".write": true  
  }  
}
```

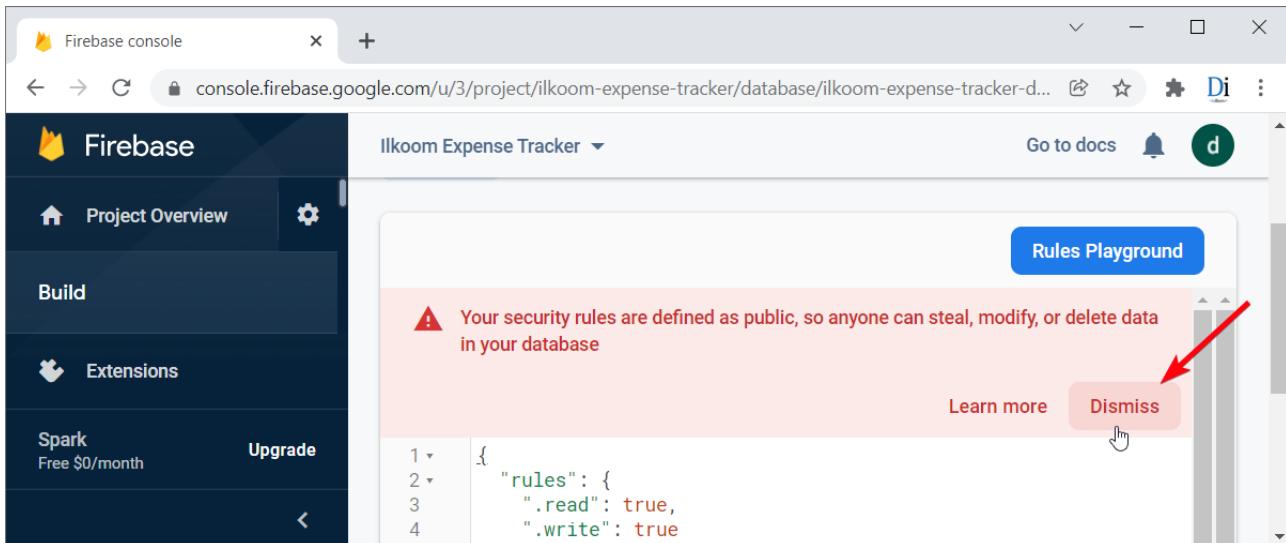
Mengonlinekan React App (Firebase Hosting)

}



Gambar: Ubah nilai false menjadi true untuk read dan write

Dengan mengubah pengaturan *read* dan *write* menjadi *true*, artinya kita mengizinkan aplikasi apa saja untuk mengakses database. Ketika pengaturan di publish, firebase akan memberikan pesan warning kalau ini tidaklah aman. Akan tetapi karena kita hanya ingin mencoba saja, pesan ini boleh diabaikan dan klik tombol **Dismiss**:



Gambar: Pesan warning dari firebase terkait hak akses

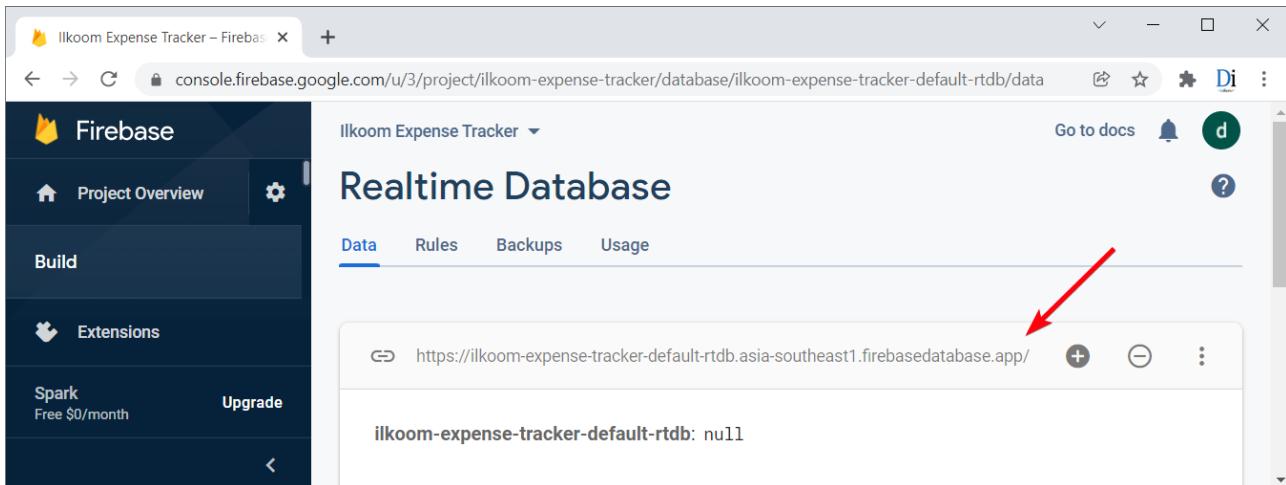
Jika anda ingin membuat aplikasi yang lebih serius, bisa pelajari lebih dalam terkait pengaturan hak akses di Firebase.

REST API Realtime Database

Realtime database yang baru saja kita buat bisa diakses dengan berbagai cara, salah satunya menggunakan API.

Dalam bab tentang [JavaScript Fetch API](#), saya sempat bahas terkait **REST API**, yakni cara mengakses CRUD menggunakan API. Namun praktek di bab tersebut baru untuk Read saja, karena sekarang kita sudah memiliki database sendiri, saatnya praktek untuk proses Create, Update dan Delete.

Alamat URL pengaksesan API firebase tersedia di halaman dashboard, yang ditunjukkan oleh gambar berikut:



Gambar: URL API dari firebase realtime database

Alamat inilah yang perlu di input ke dalam fungsi `fetch()` dan akan berbeda untuk setiap project, misalnya yang saya dapat adalah:

<https://ilkoom-expense-tracker-default-rtdb.firebaseio.app/>

URL yang anda dapat pastinya berbeda dengan alamat ini, jadi bisa disesuaikan saja.

Sebagai bahan praktek, silahkan buat file HTML baru dan isi dengan kode berikut:

01.input_data.html

```
1 <!DOCTYPE html>
2 <html lang="id">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>React Uncover</title>
8 </head>
9
10 <body>
11   <h1>Belajar JavaScript</h1>
12   <script>
13     let data = {
14       "tanggal": new Date("01 Nov 2021 9:30").getTime(),
15       "keterangan": "Gaji bulanan",
16       "nominal": 2500000,
17     }
```

```
18
19     let url = "https://ilkoom-expense-tracker-default-rtdb.";
20     url += "asia-southeast1.firebaseio.database.app";
21     url += "/transaction.json";
22
23     const myFetch = async () => {
24         try {
25             let response = await fetch(url, {
26                 method: "POST",
27                 body: JSON.stringify(data)
28             })
29             alert("Data berhasil di kirim");
30             if (!response.ok) {
31                 throw new Error(response.status);
32             }
33         }
34         catch (error) {
35             console.log(`Terjadi gangguan dengan pesan: "${error}"`);
36         }
37     }
38
39     myFetch();
40 </script>
41 </body>
42
43 </html>
```

Ini merupakan contoh proses **Create**, yakni membuat atau mengisi data baru ke dalam firebase.

Data yang akan diinput saya simpan ke dalam variabel **data** di baris 13-17. Variabel **data** berbentuk object dengan 3 property: **tanggal**, **keterangan** dan **nominal**. Object ini mirip seperti data transaksi yang kita pakai di Ilkoom Expense Tracker, tanpa property **id**.

Selanjutnya di baris 19-21 terdapat variabel **url** yang berisi alamat API firebase. Alamat ini terpaksa saya pecah karena keterbatasan lebar buku. Jika ditulis dalam 1 baris panjang, akan menjadi:

<https://ilkoom-expense-tracker-default-rtdb.firebaseio.database.app/transaction.json>

Selain alamat yang berasal dari firebase, terdapat akhiran **/transaction.json**. Ini merupakan cara membuat "tabel" dengan nama "**transaction**". Anda bebas jika ingin mengganti "transaction" dengan nama lain seperti "data_transaksi", "tabel_transaksi", dsb.

Berdasarkan arsitektur yang dipakai, firebase bukanlah **RDBMS** (Relational Database Management System) sebagaimana MySQL atau MariaDB, melainkan sebuah database **NoSQL** seperti MongoDB.

Di firebase, data disimpan dalam bentuk JSON object, bukan dalam bentuk tabel dan

kolom seperti di MySQL. Namun untuk memudahkan pemahaman, "transaction" dalam contoh di atas masih bisa kita sebut sebagai "tabel", meskipun sebenarnya disimpan sebagai sebuah property dari JSON object.

Antara baris 23-37 terdapat fungsi `async-await myFetch()` yang strukturnya sudah kita bahas di bab Fetch API. Proses pemanggilan `fetch()` sendiri ada di baris 25-28.

Untuk pembuatan data baru, fungsi `fetch()` harus dijalankan dengan 2 argument. Argument pertama berupa alamat URL API (yang sudah tersedia di dalam variabel `url`), serta argument kedua berisi object dengan 2 property berikut:

```
{  
  method: "POST",  
  body: JSON.stringify(data)  
}
```

Argument kedua dari fungsi `fetch()` dipakai untuk mengubah berbagai pengaturan terkait proses request API. Dalam contoh ini, pengaturan tersebut adalah `method` dan `body`.

Sebagai pengingat, REST API memiliki 4 method untuk setiap proses CRUD dengan rincian sebagai berikut:

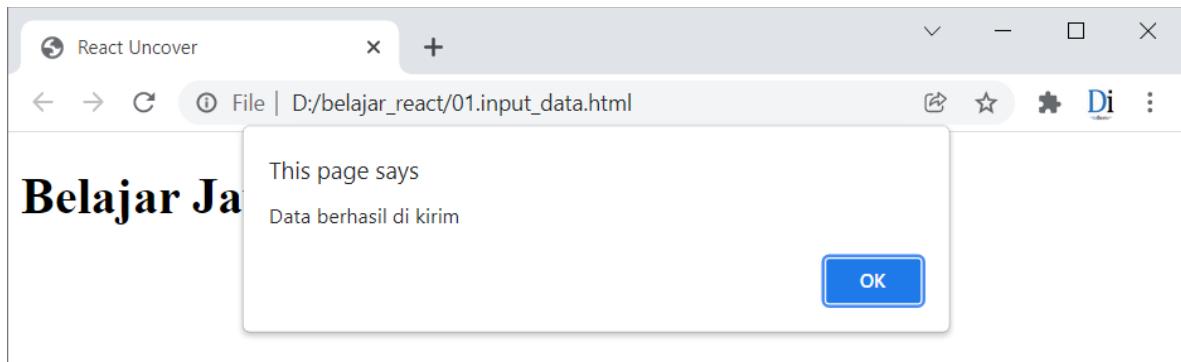
- Method **GET**: dipakai untuk mengakses data (**Read**)
- Method **POST**: dipakai untuk menyimpan data baru (**Create**)
- Method **PUT/PATCH**: dipakai untuk mengubah data yang sudah ada (**Update**)
- Method **DELETE**: dipakai untuk menghapus data (**Delete**)

Property `method` diisi "POST" karena kita akan membuat data baru. Untuk property `body`, diisi string JSON data yang akan disimpan. Di sini saya memanfaatkan fungsi `JSON.stringify()` untuk proses konversi JavaScript object menjadi string JSON.

Sebagai pemberitahuan bahwa API sudah berjalan, terdapat tambahan perintah `alert("Data berhasil di kirim")` di baris 29. Tidak lupa, fungsi `myFetch()` harus kita panggil di baris 39 agar bisa diproses.

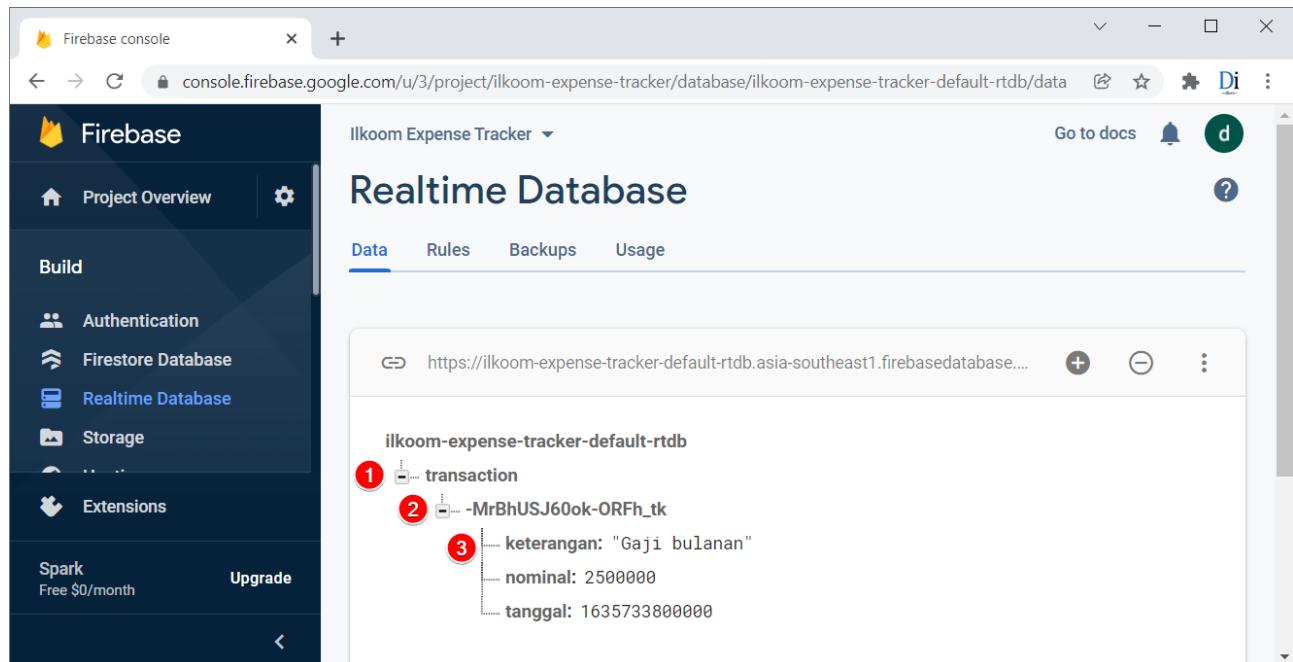
Silahkan tes kode program di atas di web browser. Jika tidak ada masalah, akan tampil pesan `alert` "Data berhasil di kirim":

Mengonlinekan React App (Firebase Hosting)



Gambar: Data sudah dikirim ke firebase database via API

Setelah itu buka dashboard firebase realtime database di web browser:



Gambar: Tampilan dashboard

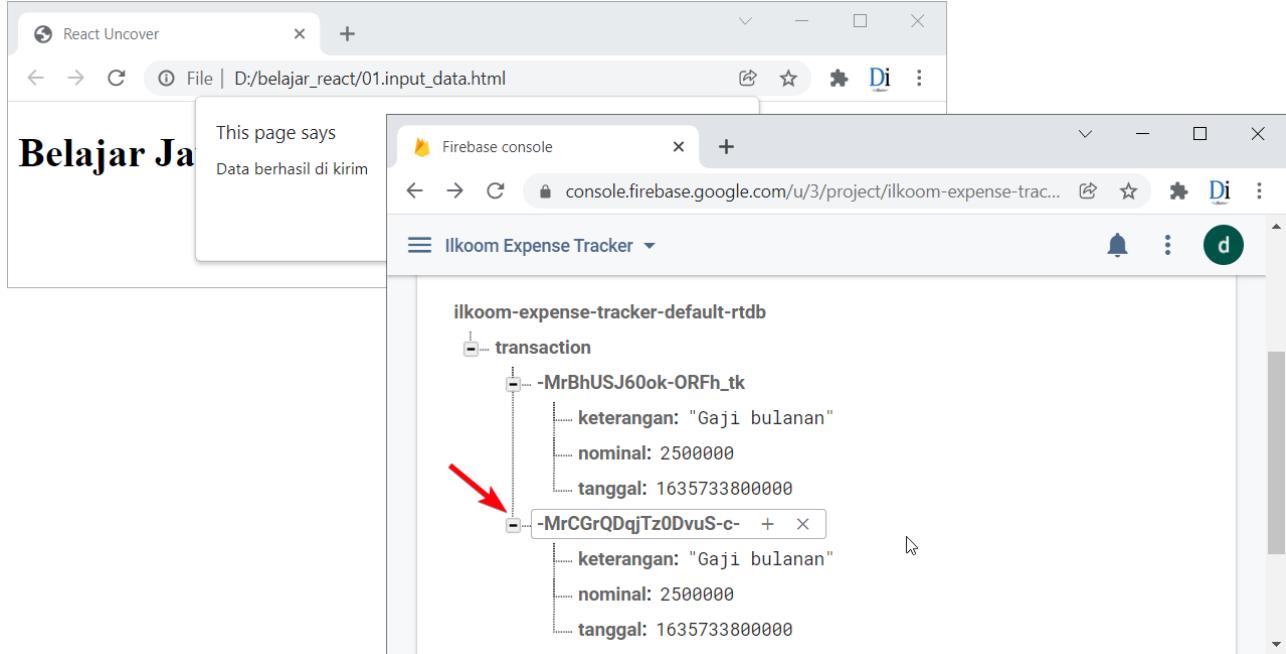
Bagian diagram di tengah halaman sekarang sudah berisi data yang berasal dari pemanggilan API kita. Tampilan ini sebenarnya mewakili sebuah struktur *nested JSON*.

Di bagian (1), terdapat property "transaction" yang sebelumnya saya sebut sebagai "tabel" di database. Ini berasal dari tambahan URL "./transaction.json" pada saat pemanggilan API.

Di bagian (2), terdapat property -MrBhUSJ60ok-ORFh_tk yang di generate otomatis oleh firebase (acak). Ini bisa disebut sebagai *id* dari data yang kita input. Setiap data nantinya mendapat id yang berbeda-beda.

Di bagian (3) terdapat data yang kita kirim. Semuanya sesuai dengan isi variabel *data* di dalam kode program.

Agar semakin jelas, silahkan refresh file *01.input_data.html* sekali lagi, kemudian cek kembali halaman dashboard firebase:



Gambar: Tambahan data baru saat file 01.input_data.html di jalankan ulang

Hasilnya bertambah 1 lagi property id acak di bawah `transaction`. Kesimpulannya, setiap kali terdapat data baru, itu akan disimpan sebagai object dari property id yang digenerate firebase. Property id inilah yang menjadi penanda identitas dari setiap data, termasuk saat proses update dan delete.

Sekarang kita coba praktik mengupdate data menggunakan method PATCH:

02.patch_data.html

```
1 <script>
2
3 let url = "https://ilkoom-expense-tracker-default-rtdb.";
4 url += "asia-southeast1.firebaseio.app";
5 url += "/transaction/";
6 url += "/-MrCGrQDqjTz0DvuS-c-.json";
7
8 const myFetch = async () => {
9     try {
10         let response = await fetch(url, {
11             method: "PATCH",
12             body: JSON.stringify({ "nominal": 4000000 })
13         })
14         alert("Edit data berhasil");
15         if (!response.ok) {
16             throw new Error(response.status);
17         }
18     }
19     catch (error) {
20         console.log(`Terjadi gangguan dengan pesan: "${error}"`);
21     }
22 }
```

Mengonlinekan React App (Firebase Hosting)

```
23
24     myFetch();
25 </script>
```

Perubahannya ada 2 tempat. Pertama, URL API mendapat tambahan data berupa id yang ingin kita update (baris 6). Id ini saya copy dari salah satu property dalam database. Jika anda ingin mencoba, samakan terlebih dahulu dengan id yang ada di dashboard firebase, karena nilai id ini pasti berbeda dengan contoh di atas.

Cara penulisan id juga harus diperhatikan, sebab id langsung disambung dengan extension .json menjadi "/-MrCGrQDqjTz0DvuS-c-.json". Tanda "-" di awal dan di akhir string id juga harus disertakan.

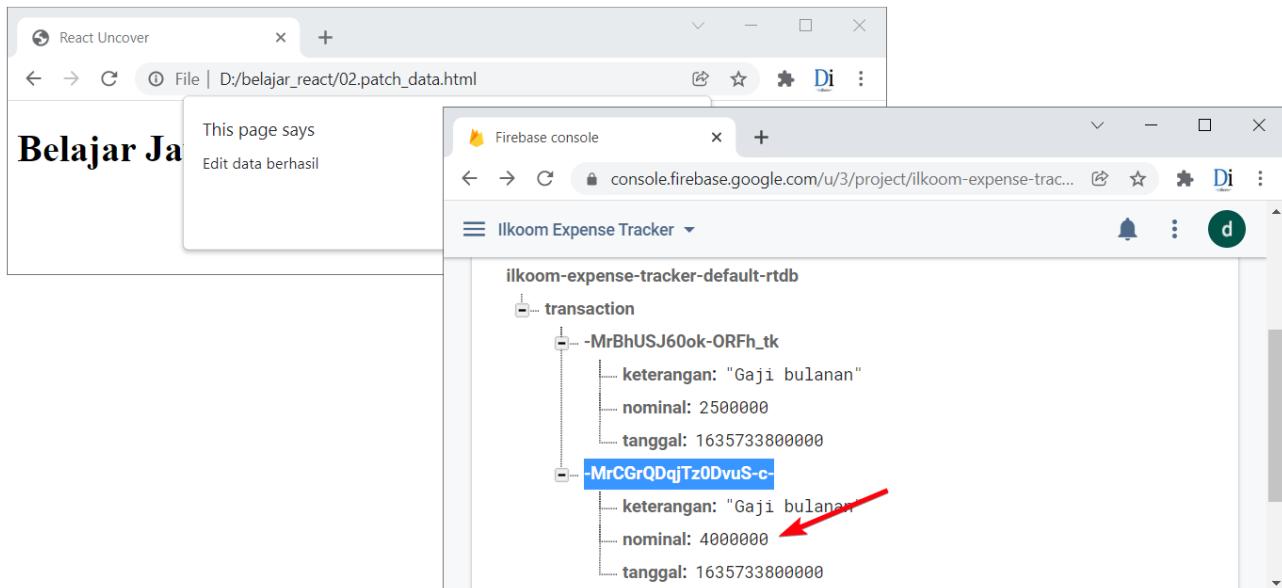
Perubahan kedua ada di dalam argument kedua fungsi patch():

```
{
  method: "PATCH",
  body: JSON.stringify({ "nominal": 4000000 })
}
```

Atribut `method` sekarang berisi string "PATCH" karena kita ingin melakukan proses update. Data yang di update menjadi nilai dari property `body`, yakni `{"nominal": 4000000}`. Data ini bisa saja disimpan ke dalam variabel terpisah seperti contoh sebelumnya.

Dengan melihat alamat URL serta isi dari property `method` dan `body`, maka kode program kita bisa dibaca: "Ubah nilai property 'nominal' menjadi 4000000 untuk data yang memiliki id: -MrCGrQDqjTz0DvuS-c-."

Jalankan file `02.patch_data.html` di atas, lalu cek hasilnya ke dashboard firebase:



Gambar: Property nominal sudah berubah menjadi 4000000

Hasilnya, property nominal untuk id `-MrCGrQDqjTz0DvuS-c-` sudah berubah menjadi

4000000. Itulah cara mengupdate data di *firebase realtime database*.

Akan tetapi di daftar REST API terdapat method **PUT** yang sama-sama berfungsi untuk proses update. Jadi apa perbedaan antara **PUT** dengan **PATCH**?

Untuk menjawabnya, langsung saja kita praktik dengan mengubah property **method** dari "**PATCH**" menjadi "**PUT**".

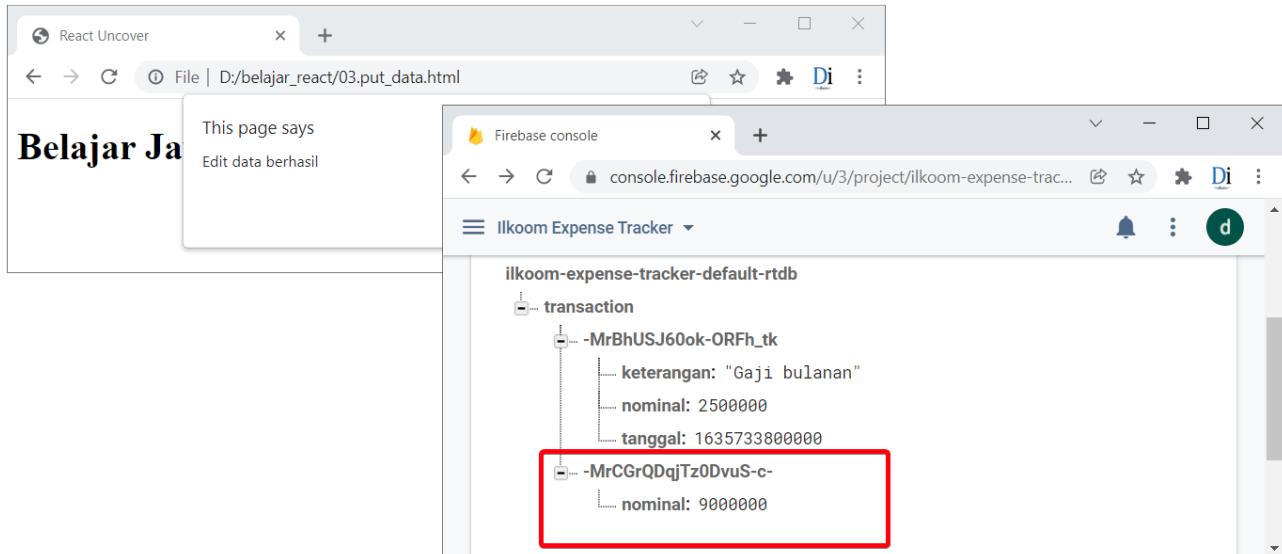
03.put_data.html

```
1 <script>
2
3     let url = "https://ilkoom-expense-tracker-default-rtdb.";
4     url += "asia-southeast1.firebaseio.database.app";
5     url += "/transaction/";
6     url += "-MrCGrQDqjTz0DvuS-c-.json";
7
8     const myFetch = async () => {
9         try {
10             let response = await fetch(url, {
11                 method: "PUT",
12                 body: JSON.stringify({ "nominal": 9000000 })
13             })
14             alert("Edit data berhasil");
15             if (!response.ok) {
16                 throw new Error(response.status);
17             }
18         }
19         catch (error) {
20             console.log(`Terjadi gangguan dengan pesan: "${error}"`);
21         }
22     }
23
24     myFetch();
25 </script>
```

Kode yang berubah hanya di baris 11 dan 12. Selain menukar nilai property **method** menjadi "**PUT**", saya juga mengisi property **nominal** menjadi 9000000 sekedar pembeda dari data sebelumnya (agar kita yakin data API sudah berjalan).

Save file di atas lalu jalankan di web browser:

Mengonlinekan React App (Firebase Hosting)



Gambar: Hasil dari method "PUT"

Ternyata, isi id `-MrCGrQDqjTz0DvuS-c-` hanya `nominal: 9000000` saja, property `keterangan` dan `tanggal` ikut hilang.

Inilah yang menjadi pembeda antara PATCH dengan PUT. Jika menggunakan method PATCH, property yang tidak ditulis tetap dipertahankan, sedangkan jika menggunakan method PUT, property yang tidak ditulis ikut terhapus.

Kita lanjut ke proses penghapusan data:

04.delete_data.html

```
1 <script>
2
3   let url = "https://ilkoom-expense-tracker-default-rtdb.";
4   url += "asia-southeast1.firebaseio.database.app";
5   url += "/transaction/";
6   url += "-MrCGrQDqjTz0DvuS-c-.json";
7
8   const myFetch = async () => {
9     try {
10       let response = await fetch(url, {
11         method: "DELETE",
12       })
13       alert("Hapus data berhasil");
14       if (!response.ok) {
15         throw new Error(response.status);
16       }
17     }
18     catch (error) {
19       console.log(`Terjadi gangguan dengan pesan: "${error}"`);
20     }
21   }
22
23   myFetch();
```

24 </script>

Untuk proses Delete atau menghapus data, caranya mirip seperti update. Nomor id yang ingin dihapus disambung sebagai akhiran URL seperti di baris 6. Kemudian argument kedua dari fungsi `fetch()` cukup diisi property method dengan nilai "DELETE".

Dengan melihat alamat URL serta isi dari property `method`, maka kode program kita bisa dibaca: "Hapus data untuk id: -MrCGrQDqjTz0DvuS-c-."

Setelah file di atas dijalankan, data property `-MrCGrQDqjTz0DvuS-c-` sudah tidak ada lagi di database:

The screenshot shows two windows side-by-side. The left window is a browser displaying a simple React application with the title 'Belajar Ja' and a message 'This page says Hapus data berhasil'. The right window is the Firebase Realtime Database console for a project named 'Ilkoom Expense Tracker'. It shows a tree structure under the path 'ilkoom-expense-tracker-default-rtdb/transaction'. A specific node, '-MrBhUSJ60ok-ORFh_tk', is expanded, revealing three child nodes: 'keterangan: "Gaji bulanan"', 'nominal: 250000', and 'tanggal: 163573380000'. The entire node is highlighted with a red rectangular box.

Gambar: Data sudah berhasil dihapus

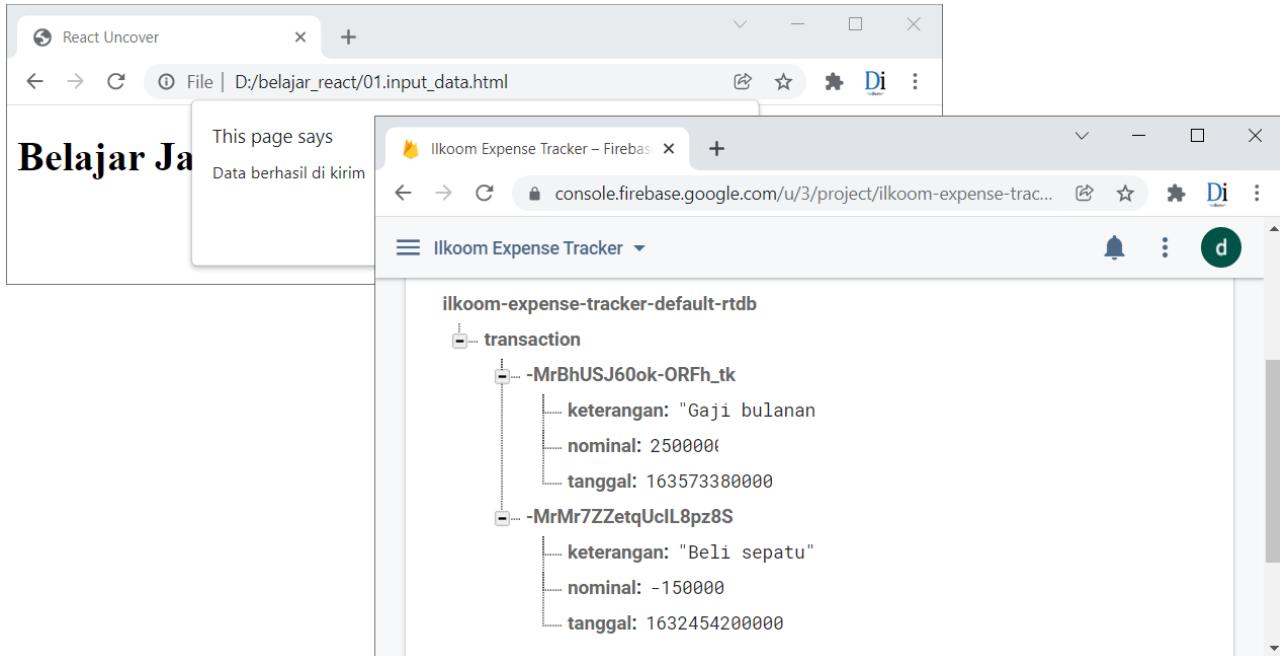
Jika kita ingin menghapus data lain, cukup ganti alamat URL dengan id dari data tersebut.

Sekarang kita masuk ke proses pembacaan data (Read). Sebelum itu, silahkan tambah 1 data lagi ke dalam database. Caranya, ganti isi variabel data di file `01.input_data.html` dengan nilai berikut, lalu jalankan ulang:

`01.input_data.html`

```
1 ...
2   let data = {
3     "tanggal": new Date("24 Sept 2021 10:30").getTime(),
4     "keterangan": "Beli sepatu",
5     "nominal": -150000,
6   }
7 ...
```

Mengonlinekan React App (Firebase Hosting)



Gambar: Terdapat 2 data di database

Hasilnya, kita memiliki 2 data untuk percobaan. Berikut kode program untuk mengakses data dari firebase database:

05.read_data.html

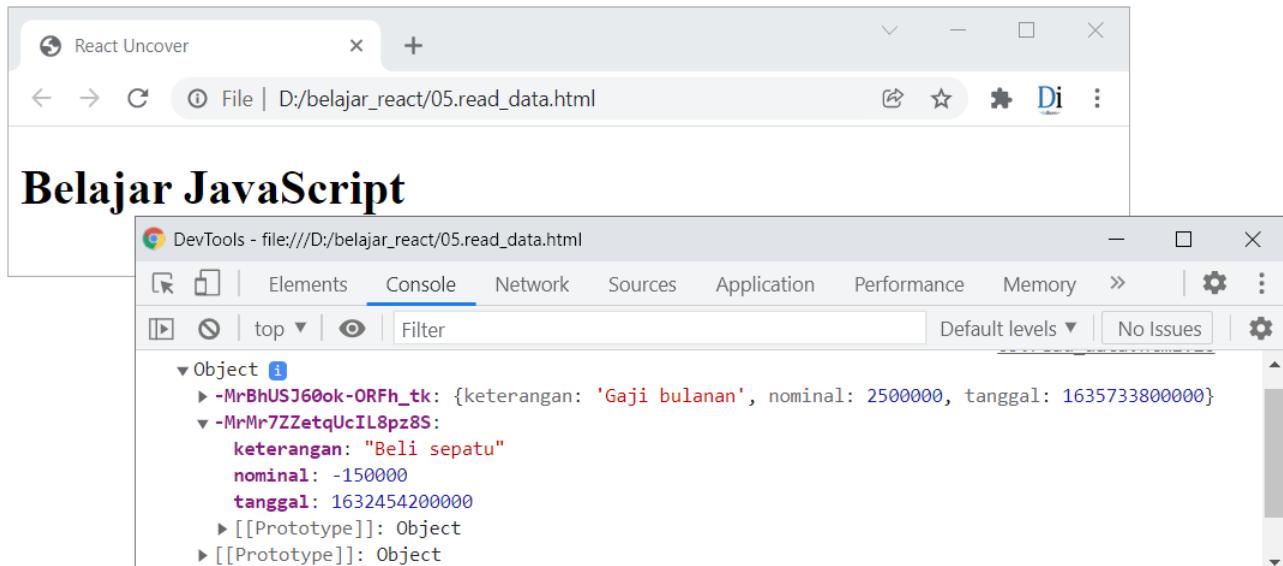
```
1 <script>
2   let url = "https://ilkoom-expense-tracker-default-rtdb.";
3   url += "asia-southeast1.firebaseio.database.app";
4   url += "/transaction.json";
5
6   const myFetch = async () => {
7     try {
8       let response = await fetch(url);
9       if (!response.ok) {
10         throw new Error(response.status);
11     }
12     let responseData = await response.json();
13     console.log(responseData);
14   }
15   catch (error) {
16     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
17   }
18 }
19
20 myFetch();
21 </script>
```

Untuk proses pembacaan data, alamat URL API tidak perlu kode tambahan, cukup copy alamat yang ada di dashboard firebase (baris 2-4). Alamat ini kemudian diinput ke dalam fungsi `fetch(url)` saja seperti di baris 8.

Untuk proses pembacaan data, kita tidak perlu mengisi argument kedua dari fungsi `fetch()` karena secara default sudah berisi `method: "GET"`, atau tidak masalah juga jika ingin ditambah seperti kode berikut:

```
let response = await fetch(url, {  
  method: "GET",  
});
```

Hasil dari pemanggilan fungsi `fetch(url)` ditampung oleh variabel `response` yang kemudian diakses lagi dengan perintah `responseData = await response.json()` di baris 12. Isi variabel `responseData` ini kemudian saya tampilkan dengan perintah `console.log(responseData)` di baris 13. Berikut hasilnya:



Gambar: Menampilkan hasil firebase database

Variabel `responseData` berisi object dengan 2 property, yakni `-MrBhUSJ60ok-ORFh_tk` dan `-MrMr7ZZetqUcIL8pz8S`. Keduanya berasal dari id yang digenerate oleh firebase dan berisi data transaksi yang sudah kita input. Jika nantinya terdapat data baru, maka akan menjadi sebuah property baru.

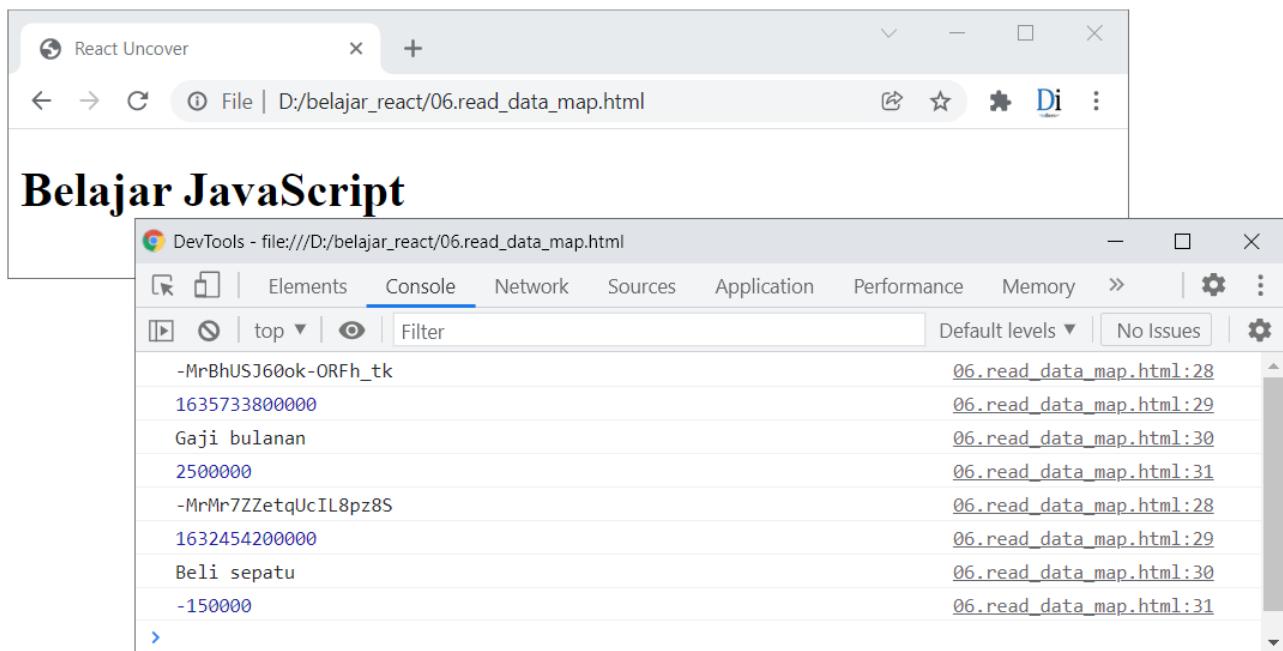
Mengakses data id secara manual tentu tidak praktis, karena itu kita bisa buat perulangan untuk mengakses nilai yang tersimpan di dalamnya. Berikut salah satu cara yang bisa dipakai:

06.read_data_for_in.html

```
1 <script>  
2   let url = "https://ilkoom-expense-tracker-default-rtdb.";  
3   url += "asia-southeast1.firebaseio.database.app";  
4   url += "/transaction.json";  
5  
6   const myFetch = async () => {  
7     try {  
      let response = await fetch(url, {  
        method: "GET",  
      });  
      let responseData = await response.json();  
      console.log(responseData);  
    } catch (error) {  
      console.error(error);  
    }  
  };  
11 </script>
```

Mengonlinekan React App (Firebase Hosting)

```
9         method: "GET",
10    });
11    if (!response.ok) {
12      throw new Error(response.status);
13    }
14    let responseData = await response.json();
15
16    for (let key in responseData) {
17      console.log(key);
18      console.log(responseData[key].tanggal);
19      console.log(responseData[key].keterangan);
20      console.log(responseData[key].nominal);
21    }
22
23  }
24  catch (error) {
25    console.log(`Terjadi gangguan dengan pesan: "${error}"`);
26  }
27}
28
29 myFetch();
30 </script>
```



Gambar: Mengakses data firebase dengan perulangan

Di baris 16-21 terdapat perulangan for in yang akan "membuka" isi object responseData. Di sini kita tidak bisa memakai perulangan .map() karena responseData berisi kumpulan object, bukan array.

Dengan menulis `for (let key in responseData)`, maka di dalam blok perulangan, variabel key akan berisi id dari setiap object. Variabel key inilah yang dipakai untuk mengakses setiap data, seperti `responseData[key].tanggal` atau `responseData[key].keterangan`. Hasilnya, semua data yang tersimpan di dalam database sudah bisa diakses.

Ketika mengakses data yang berasal dari API, struktur JSON yang didapat bisa sangat berbeda dari satu web ke web lain. Bisa "membuka" data JSON yang didapat, menjadi salah satu skill yang wajib dimiliki.

Itulah bahasan singkat tentang cara mengakses data dari firebase realtime database. Selanjutnya kita akan coba terapkan ke dalam aplikasi Ilkoom Expense Tracker.

20.7. REST API Ilkoom Expense Tracker

Aplikasi **Ilkoom Expense Tracker** yang kita upload sebelumnya masih menyimpan data di memory, akibatnya data transaksi otomatis terhapus saat web browser ditutup. Karena kita sudah memiliki database, saatnya konversi proses pembacaan dan penyimpanan data agar bisa permanen.

Di Ilkoom Expense Tracker, proses pembacaan dan penyimpanan data ada di dalam komponen App. Berikut saya tampilkan kembali kode program yang ada saat ini:

my-app\src\App.js

```
1 import React, { useState } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
5 import SaldoBox from './components/SaldoBox';
6 import AddTransaction from './components/AddTransaction';
7
8 const initTransactions = [
9   {
10     "id": "619941539079",
11     "tanggal": new Date("01 Nov 2021 9:30").getTime(),
12     "keterangan": "Gaji bulanan",
13     "nominal": 2500000,
14   },
15   {
16     "id": "749179155708",
17     "tanggal": new Date("23 Nov 2021 10:00").getTime(),
18     "keterangan": "Uang lembur",
19     "nominal": 750000,
20   },
21   {
22     "id": "568004092688",
23     "tanggal": new Date("24 Sept 2021 10:30").getTime(),
24     "keterangan": "Beli sepatu",
25     "nominal": -150000,
26   }
27 ];
28
29 const App = () => {
30   const [transactions, setTransaction] = useState(initTransactions);
```

Mengonlinekan React App (Firebase Hosting)

```
31 // handler untuk menambah data transaction,
32 // akan di-trigger dari komponen AddTransaction
33 const handleTambahTransaction = (data) => {
34   let newTransactions = [
35     ...transactions, data
36   ];
37
38   // atur ulang urutan transaction agar tanggal terkecil di bagian atas
39   newTransactions.sort((a, b) => a.tanggal - b.tanggal);
40
41   setTransaction(newTransactions);
42 }
43
44 // handler untuk menghapus data transaction di komponen AddTransaction
45 const handleHapusTransaction = (e) => {
46
47   // cari index transaction yang akan dihapus berdasarkan id
48   const result = transactions.findIndex(
49     transaction => (transaction.id === e.target.id)
50   );
51
52   // copy transactions karena fungsi splice akan mengubah array asal (mutate)
53   const newTransactions = transactions;
54   newTransactions.splice(result, 1);
55   setTransaction([...newTransactions]);
56 }
57
58 return (
59   <React.Fragment>
60     <Header />
61     <SaldoBox transactions={transactions} />
62     <Transaction transactions={transactions}
63       onHapusTransaction={handleHapusTransaction} />
64     <AddTransaction onTambahTransaction={handleTambahTransaction} />
65     <Footer />
66   </React.Fragment>
67 )
68 }
69
70 export default App;
```

Di awal kode program (baris 8-27) terdapat pendefinisian konstanta `initTransactions` yang dipakai untuk mengisi nilai awal dari state `transactions` di baris 30.

Proses menambah data transaksi dilakukan oleh fungsi `handleTambahTransaction()` di baris 34-43, dan untuk menghapus data transaksi dilakukan dari fungsi `handleHapusTransaction()` di baris 46-56. Agar bisa menggunakan API, kita perlu memodifikasi kedua fungsi ini.

Berikut perubahan kode yang diperlukan:

my-app\src\App.js

```
1 import React, { useState, useEffect } from 'react';
2 import Header from './components/Header';
```

Mengonlinekan React App (Firebase Hosting)

```
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
5 import SaldoBox from './components/SaldoBox';
6 import AddTransaction from './components/AddTransaction';
7
8 let url = "https://ilkoom-expense-tracker-default-rtdb.";
9 url += "asia-southeast1.firebaseio.app";
10 url += "/transaction.json";
11
12 const App = () => {
13   const [transactions, setTransaction] = useState([]);
14
15   // Use effect untuk mengakses data dari API firebase
16   useEffect(() => {
17     const myFetch = async () => {
18       try {
19         let response = await fetch(url);
20         if (!response.ok) {
21           throw new Error(response.status);
22         }
23         let responseData = await response.json();
24
25         const initTransactions = [];
26         for (const key in responseData) {
27           initTransactions.push({
28             id: key,
29             tanggal: responseData[key].tanggal,
30             keterangan: responseData[key].keterangan,
31             nominal: responseData[key].nominal,
32           })
33         }
34
35         // atur ulang urutan transaction agar tanggal terkecil di bagian atas
36         initTransactions.sort((a, b) => a.tanggal - b.tanggal);
37
38         setTransaction(initTransactions);
39     }
40
41     catch (error) {
42       console.log(`Terjadi gangguan dengan pesan: "${error}"`);
43     }
44   }
45   myFetch();
46 },[]);
47
48 // handler untuk menambah data transaction,
49 // akan di-trigger dari komponen AddTransaction
50 const handleTambahTransaction = async (data) => {
51   // Kirim data ke server (firebase)
52   try {
53     let response = await fetch(url, {
54       method: "POST",
55       body: JSON.stringify(data)
56     })
57     if (!response.ok) {
```

Mengonlinekan React App (Firebase Hosting)

```
58         throw new Error(response.status);
59     }
60   }
61   catch (error) {
62     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
63   }
64 }
65
66 // handler untuk menghapus data transaction di komponen AddTransaction
67 const handleHapusTransaction = async (e) => {
68   // Rangkai alamat URL agar berisi id data yang dihapus
69
70   let url = "https://ilkoom-expense-tracker-default-firebase.firebaseio.com/";
71   url += "asia-southeast1.firebaseio.database.json";
72   url += "/transaction/";
73   url += `${e.target.id}.json`;
74
75   // Kirim delete request ke server (firebase)
76   try {
77     let response = await fetch(url, {
78       method: "DELETE",
79     })
80     if (!response.ok) {
81       throw new Error(response.status);
82     }
83   }
84   catch (error) {
85     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
86   }
87 }
88
89 return (
90   <React.Fragment>
91     <Header />
92     <SaldoBox transactions={transactions} />
93     <Transaction transactions={transactions}
94       onHapusTransaction={handleHapusTransaction} />
95     <AddTransaction onTambahTransaction={handleTambahTransaction} />
96     <Footer />
97   </React.Fragment>
98 )
99 }
100
101 export default App;
```

Perubahan pertama ada di baris 1, sekarang terdapat tambahan import useEffect yang akan dipakai dalam kode program nanti. Kemudian di baris 8 saya mendefinisikan variabel url yang berisi alamat API dari firebase realtime database.

Masuk ke dalam komponen App, state transactions diisi dengan nilai awal array kosong "[]". Alasannya karena nilai awal akan kita ambil dari database firebase menggunakan useEffect sesaat lagi.

useEffect tersebut ada di baris baris 16-46 yang berisi fungsi `myFetch()` seperti bahasan kita sebelumnya. Fungsi `fetch()` di baris 19 akan mengambil data JSON dari server firebase untuk kemudian ditampung ke dalam variabel `responseData`.

Perulangan `for in` antara baris 25-33 akan "membuka" data JSON tersebut untuk dikonversi menjadi array. Ini diperlukan supaya kode program di komponen lain tidak perlu kita edit, sebab struktur state `transactions` tetap tidak berubah.

Sebagai contoh, jika variabel `responseData` berisi data JSON berikut:

```
{  
  "transaction" : {  
    "-MrBhUSJ60ok-ORFh_tk" : {  
      "keterangan" : "Gaji bulanan",  
      "nominal" : 2500000,  
      "tanggal" : 1635733800000  
    },  
    "-MrMr7ZZetqUcIL8pz8S" : {  
      "keterangan" : "Beli sepatu",  
      "nominal" : -150000,  
      "tanggal" : 1632454200000  
    }  
  }  
}
```

Maka hasil dari perulangan `for in` akan mengisi variabel `initTransactions` dengan data berikut:

```
[  
  {  
    "id": "-MrBhUSJ60ok-ORFh_tk",  
    "tanggal": 1635733800000,  
    "keterangan": "Gaji bulanan",  
    "nominal": 2500000  
  },  
  {  
    "id": "-MrMr7ZZetqUcIL8pz8S",  
    "tanggal": 1632454200000,  
    "keterangan": "Beli sepatu",  
    "nominal": -150000  
  }  
]
```

Isi dari `initTransactions` kemudian diinput ke dalam state `transactions` dengan perintah `setTransaction(initTransactions)` di baris 38. Hasilnya, saat halaman di load, state `transactions` akan berisi data dari database.

Fungsi *event handling* `handleTambahTransaction()` dan `handleHapusTransaction()` juga perlu mengirim request ke URL API firebase agar data di database ikut berubah. Karena pengiriman API butuh pemrosesan `promise`, maka kedua fungsi perlu tambahan keyword `async` sebelum penulisan parameter (baris 51 dan 68).

Mengonlinekan React App (Firebase Hosting)

Untuk penambahan data transaksi, itu akan diproses oleh fungsi `fetch()` di baris 54-57:

```
let response = await fetch(url, {
  method: "POST",
  body: JSON.stringify(data)
})
```

Variabel `data` berasal dari komponen `AddTransaction` yang diterima sebagai parameter.

Sedangkan untuk menghapus transaksi, akan diproses oleh fungsi `fetch()` di baris 78-80:

```
let response = await fetch(url, {
  method: "DELETE",
})
```

Tambahan untuk proses delete, struktur URL harus ditulis ulang dengan menyertakan id transaksi yang akan dihapus. Ini dilakukan antara baris 71-74:

```
let url = "https://ilkoom-expense-tracker-default-rtbd.";
url += "asia-southeast1.firebaseio.app";
url += "/transaction/";
url += `${e.target.id}.json`;
```

Variabel `e.target.id` berasal dari argument yang dikirim saat form disubmit (`e` berisi event object). Penjelasan lebih lanjut sudah kita bahas di bab pembuatan Ilkoom Expense Tracker.

Simpan perubahan yang ada, jalankan `npm start` dan cek ke web browser:

The screenshot displays two windows side-by-side. On the left is a web browser window titled 'Ilkoom Expense Tracker' showing the application's home screen. It features a balance of 'Rp. 2.350.000' and two transaction buttons: 'Rp. 2.500.000' (green) and 'Rp. -150.000' (red). Below these are sections for 'Pemasukan' (Income) and 'Pengeluaran' (Expense), each with a single transaction listed. On the right is a Firebase Realtime Database console window titled 'Realtime Database'. It shows a tree structure under the 'transaction' node. The first transaction is 'MrBhUSJ60ok-ORFh_tk' with details: 'keterangan: "Gaji bulanan"', 'nominal: 2500000', and 'tanggal: 1635733800000'. The second transaction is 'MrMr7ZZetqUclL8pz8S' with details: 'keterangan: "Beli sepatu"', 'nominal: -150000', and 'tanggal: 1632454200000'. Red arrows point from the transaction buttons in the application to their corresponding entries in the database.

Gambar: Data transaksi sudah sesuai dengan database

Sip, data transaksi yang berasal dari firebase sudah berhasil ditampilkan.

Sekarang mari kita coba tambah 1 transaksi baru menggunakan form:

Mengonlinekan React App (Firebase Hosting)

Tambah Transaksi

Tanggal Keterangan Nominal* (+/-)

21/12/2021 Isi kuota HP -100000 Tambah

* Jika diisi angka negatif, akan tercatat di pengeluaran

Gambar: Tambah satu transaksi baru dari form

Pada saat form di submit, data transaksi sudah berhasil masuk ke database firebase (cek isi dashboard firebase). Akan tetapi penambahan itu tidak tampil di aplikasi kita:

Pemasukan

Gaji bulanan Rp. 2.500.000

Pengeluaran

Beli sepatu Rp. -150.000

Ilkoom Expense Tracker - Firebase

-MRD1hu0SNvV2RNwEF8

- keterangan: "Gaji bulanan"
- nominal: 2500000
- tanggal: 163573380000

-MrMr7ZZetqUclL8pz8S

- keterangan: "Beli sepatu"
- nominal: -150000
- tanggal: 163245420000

-MrRD1hu0SNvV2RNwEF8

- id: "402261913888"
- keterangan: "Isi kuota HP"
- nominal: -100000
- tanggal: 1640077277276

Gambar: Data sudah berhasil masuk ke database, tapi tidak tampil di halaman

Setelah halaman di-reload, barulah terlihat tambahan data transaksi tersebut:

Pemasukan Pengeluaran

Gaji bulanan Rp. 2.500.000

Beli sepatu Rp. -150.000

Isi kuota HP Rp. -100.000

Gambar: Transaksi baru tampil setelah halaman refresh

Kenapa bisa seperti ini? Mengapa hasil inputan baru terlihat setelah di reload terlebih dahulu?

Ini berhubungan dengan konsep *React lifecycle*. Jika anda masih ingat, aplikasi React butuh proses re-render dari waktu ke waktu, karena itu merupakan cara untuk mengupdate tampilan halaman. Salah satu yang men-trigger proses re-render adalah perubahan nilai state.

Dalam aplikasi kita saat ini, proses penambahan dan penghapusan transaksi langsung terhubung ke database, tidak lagi melalui state *transactions*. Inilah yang menjadi alasan kenapa halaman perlu di reload terlebih dahulu, sebab tidak ada mekanisme yang akan me-re-render halaman di dalam fungsi `handleTambahTransaction()` dan `handleHapusTransaction()`.

Terdapat beberapa solusi untuk masalah ini. Salah satunya dengan mengupdate state *transactions* di dalam fungsi `handleTambahTransaction()` dan `handleHapusTransaction()`.

Berikut contoh yang dimaksud:

```
1 ...  
2     const handleTambahTransaction = async (data) => {  
3         // Kirim data ke server (firebase)  
4         try {  
5             let response = await fetch(url, {  
6                 method: "POST",  
7                 body: JSON.stringify(data)  
8             })  
9  
10        const newTransactions = [  
11            ...transactions, data  
12        ];  
13  
14        // atur ulang urutan transaction agar tanggal terkecil di bagian atas  
15        newTransactions.sort((a, b) => a.tanggal - b.tanggal);  
16  
17        setTransaction(newTransactions);  
18  
19        if (!response.ok) {  
20            throw new Error(response.status);  
21        }  
22    }  
23    catch (error) {  
24        console.log(`Terjadi gangguan dengan pesan: "${error}"`);  
25    }  
26 }  
27 ...
```

Setelah fungsi `fetch()` di baris 5-8, saya mengupdate state *transactions* dengan data yang ditambah tadi. Kode yang dipakai sama seperti di versi tanpa database. Perintah `setTransaction(newTransactions)` akan menjadi trigger untuk proses re-render, karena sudah terjadi perubahan nilai state.

Ketika coba dijalankan, transaksi baru memang akan langsung muncul (tidak perlu di reload dulu). Akan tetapi muncul masalah baru terkait nomor id transaksi. Dengan cara di atas,

transaksi yang tampil di halaman saat ini berasal dari penambahan state `transactions`. Masalahnya, transaksi itu tidak memiliki nomor id, sebab nomor id perlu di generate oleh server firebase.

Maka setelah menambah data ke firebase, kita harus ambil lagi data tadi lewat API agar nomor id-nya bisa didapat. Di dalam fungsi `handleTambahTransaction()`, harus terdapat 2 kali pemanggilan API, satu untuk menambah data, dan satu lagi untuk mengakses data.

Solusi lain dari masalah ini, saya akan menambah sebuah state dummy untuk men-trigger proses re-render. Idenya, saat terjadi penambahan dan penghapusan data, update nilai state dummy agar terjadi proses re-render. Selain itu, state dummy turut diinput sebagai dependency dari `useEffect` hook. Tujuannya agar setiap kali terjadi perubahan, ambil data baru dari server database.

Berikut kode program lengkap dari solusi ini:

```
my-app\src\App.js

1 import React, { useState, useEffect } from 'react';
2 import Header from './components/Header';
3 import Footer from './components/Footer';
4 import Transaction from './components/Transaction';
5 import SaldoBox from './components/SaldoBox';
6 import AddTransaction from './components/AddTransaction';
7
8 let url = "https://ilkoom-expense-tracker-default-rtdb.";
9 url += "asia-southeast1.firebaseio.database.app";
10 url += "/transaction.json";
11
12 const App = () => {
13   const [transactions, setTransaction] = useState([]);
14   const [submitCount, setSubmitCount] = useState(0);
15
16   // Use effect untuk mengakses data dari API firebase
17   useEffect(() => {
18     const myFetch = async () => {
19       try {
20         let response = await fetch(url);
21         if (!response.ok) {
22           throw new Error(response.status);
23         }
24         let responseData = await response.json();
25
26         const initTransactions = [];
27         for (const key in responseData) {
28           initTransactions.push({
29             id: key,
30             tanggal: responseData[key].tanggal,
31             keterangan: responseData[key].keterangan,
32             nominal: responseData[key].nominal,
33           })
34         }
35       }
36     }
37   })
38 }
```

Mengonlinekan React App (Firebase Hosting)

```
35     console.log(JSON.stringify(initTransactions));
36     // atur ulang urutan transaction agar tanggal terkecil di bagian atas
37     initTransactions.sort((a, b) => a.tanggal - b.tanggal);
38
39     setTransaction(initTransactions);
40   }
41
42   catch (error) {
43     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
44   }
45 }
46 myFetch();
47 }, [submitCount]);
48
49 // handler untuk menambah data transaction,
50 // akan di-trigger dari komponen AddTransaction
51 const handleTambahTransaction = async (data) => {
52   // Kirim data ke server (firebase)
53   try {
54     let response = await fetch(url, {
55       method: "POST",
56       body: JSON.stringify(data)
57     })
58     if (!response.ok) {
59       throw new Error(response.status);
60     }
61     setSubmitCount(submitCount + 1);
62   }
63   catch (error) {
64     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
65   }
66 }
67
68 // handler untuk menghapus data transaction di komponen AddTransaction
69 const handleHapusTransaction = async (e) => {
70   // Rangkai alamat URL agar berisi id data yang dihapus
71
72   let url = "https://ilkoom-expense-tracker-default-firebase.firebaseio";
73   url += ".com.firebaseio/app";
74   url += "/transaction/";
75   url += `${e.target.id}.json`;
76
77   // Kirim delete request ke server (firebase)
78   try {
79     let response = await fetch(url, {
80       method: "DELETE",
81     })
82     if (!response.ok) {
83       throw new Error(response.status);
84     }
85     setSubmitCount(submitCount + 1);
86   }
87   catch (error) {
88     console.log(`Terjadi gangguan dengan pesan: "${error}"`);
89   }
```

```
90     }
91
92     return (
93       <React.Fragment>
94         <Header />
95         <SaldoBox transactions={transactions} />
96         <Transaction transactions={transactions}
97           onHapusTransaction={handleHapusTransaction} />
98         <AddTransaction onTambahTransaction={handleTambahTransaction} />
99         <Footer />
100      </React.Fragment>
101    )
102  }
103
104 export default App;
```

Tidak banyak yang harus ditambah, cukup 4 baris saja.

Di awal komponen `App`, terdapat pendefinisian state `submitCount` di baris 14. Inilah state `dummy` yang saya siapkan sebagai *trigger* untuk proses re-render. Nilai dari `submitCount` akan naik sebanyak 1 angka di baris 61 dan 85, tepat setelah penambahan atau penghapusan data.

Di baris 47, state `submitCount` juga di set sebagai *dependency* dari `useEffect` hook. Sehingga setiap kali nilai `submitCount` berubah, `useEffect` akan dijalankan ulang, termasuk di dalamnya fungsi `fetch()` yang akan mengambil data baru ke database server.

Dengan tambahan ini, state `transactions` akan selalu diisi ulang setiap kali terjadi penambahan dan penghapusan transaction. Silahkan tes langsung di web browser untuk mengujinya.

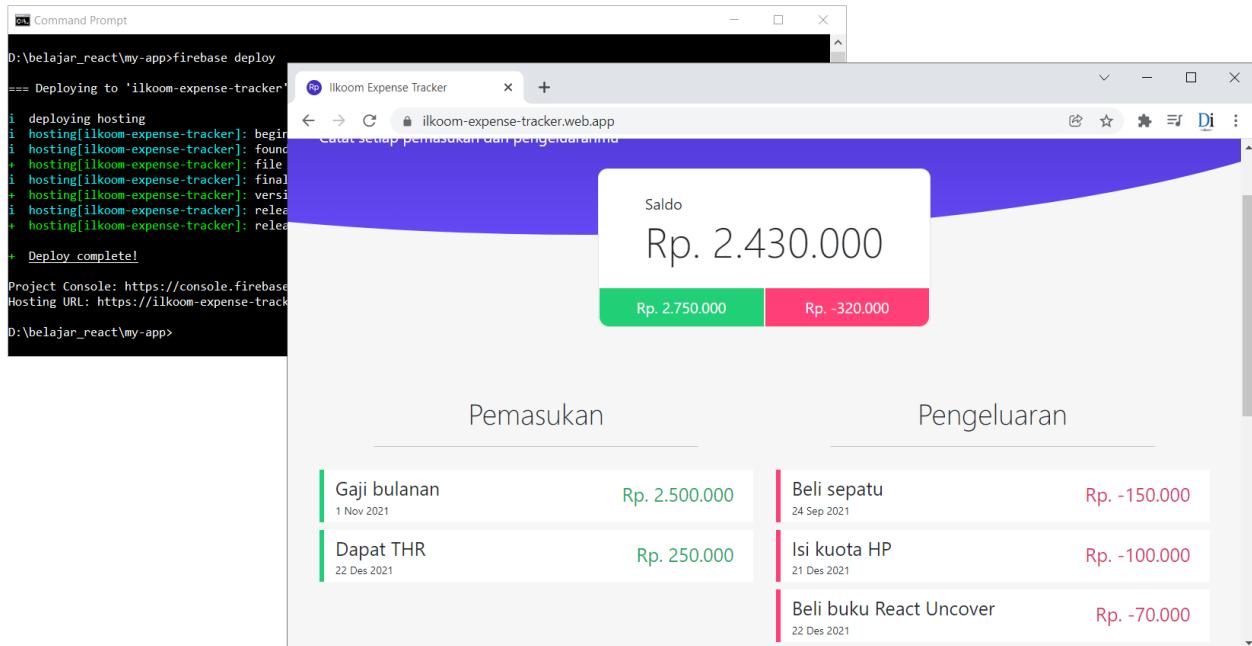
Jika tidak ada masalah, saatnya update file Ilkoom Expense Tracker yang ada di firebase hosting. Pertama, kita harus compile ulang kode program yang ada dengan perintah:

```
npm run build
```

Setelah selesai, tinggal upload dengan perintah:

```
firebase deploy
```

Mengonlinekan React App (Firebase Hosting)



Gambar: Proses update aplikasi Ilkoom Expense Tracker sudah berhasil

Dan..., file kita sudah update! Sekarang data transaksi sudah permanen dan tersimpan di dalam firebase realtime database.

20.8. Upload File CRUD Mahasiswa

Sebagai materi tambahan dan ajang latihan, saya juga ingin mengupload aplikasi CRUD mahasiswa. Tidak lupa, nanti kita juga akan modif kode program yang ada agar data mahasiswa bisa disimpan secara permanen di firebase realtime database.

Kali ini saya yakin anda sudah hafal apa saja yang harus kita lakukan. Karena itu saya akan tulis dalam instruksi singkat saja:

1. Buat folder `create react app` baru, atau bisa juga pakai folder `my-app` sebelumnya. Jika memakai folder `my-app` yang sudah ada, hapus folder `public`, `src`, `build`, `.firebase`, dan juga file `.firebaserc` serta `firebase.json` (silahkan backup jika diperlukan)
2. Copy file source code CRUD mahasiswa dari `belajar_react.zip` ke dalam `my-app`. Folder yang diperlukan adalah `public` dan `src`.
3. Jalankan `npm start` dan pastikan aplikasi sudah berjalan (tidak error).
4. Buat project baru di Firebase console, misalnya "**Ilkoom CRUD Mahasiswa**". Skip pilihan google analytics.
5. Buka cmd, masuk ke folder `create react app` berada, misal `D:\belajar_react\my-app`.
6. Jalankan perintah **firebase init**
7. *Are you ready to proceed? Y*

Mengonlinekan React App (Firebase Hosting)

8. Pilih menu "Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys"
9. Pilih menu "Use an existing project"
10. Pilih project yang akan dipakai, misalnya "ilkoom-crud-mahasiswa (Ilkoom CRUD Mahasiswa)"
11. What do you want to use as your public directory? **build**
12. Configure as a single-page app (rewrite all urls to /index.html)? **Y**
13. Set up automatic builds and deploys with GitHub? **N**
14. Compile file create react app dengan perintah **npm run build**
15. Upload file ke Firebase hosting dengan perintah **firebase deploy**

Done! Aplikasi Ilkoom CRUD Mahasiswa sudah online di alamat <https://ilkoom-crud-mahasiswa.web.app/>.

NIM	Nama	Jurusan	Asal Provinsi	
19010214	Lisa Permata	Sistem Informasi	Sumatera Barat	<button>Edit</button> <button>Hapus</button>
20010710	Rudi Setiawan	Ilmu Komputer	Jawa Tengah	<button>Edit</button> <button>Hapus</button>
20010790	Friska Ramadhani	Ilmu Komputer	Kalimantan Barat	<button>Edit</button> <button>Hapus</button>

Tambah

Nim tidak boleh kosong
Nama tidak boleh kosong
Jurusan tidak boleh kosong
Asal Provinsi tidak boleh kosong

Gambar: Aplikasi Ilkoom CRUD Mahasiswa sudah online

Sekarang saatnya buat Firebase Realtime Database:

1. Buka kembali Firebase Console dan pastikan masuk ke project **Ilkoom CRUD Mahasiswa**
2. Buat database dengan cara klik menu **Build**, pilih **Realtime Database**, dan setelah halaman terbuka klik tombol "**Create Database**"
3. Pilih server di **Singapore (asia-southeast1)**
4. Pilih security rules "**Start in locked mode**"

5. Setelah database tersedia, update security rules dengan aturan berikut:

```
{  
  "rules": {  
    ".read": true,  
    ".write": true  
  }  
}
```

6. Tutup jendela warning dengan klik tombol "**Dismiss**"

Sekarang kita harus update file kode program. Saat ini, data mahasiswa disimpan ke dalam memory, atau tepatnya state `mahasiswas`. Sama seperti di project Ilkoom Expense Tracker, data mahasiswa akan dikirim via API ke dalam database firebase.

Berikut modifikasi akhir dari komponen App:

my-app\src\App.js

```
1 import React, { useState, useEffect } from 'react';  
2 import RowMahasiswa from './components/RowMahasiswa';  
3 import RowTambahMahasiswa from './components/RowTambahMahasiswa';  
4  
5 let url = "https://ilkoom-crud-mahasiswa-default-rtdb.";  
6 url += "asia-southeast1.firebaseio.database.app";  
7 url += "/mahasiswas.json";  
8  
9 const App = () => {  
10   const [mahasiswas, setMahasiswas] = useState([]);  
11   const [submitCount, setSubmitCount] = useState(0);  
12  
13   useEffect(() => {  
14     const myFetch = async () => {  
15  
16       let response = await fetch(url);  
17       let responseData = await response.json();  
18  
19       const initMahasiswas = [];  
20       for (const key in responseData) {  
21         initMahasiswas.push({  
22           id: key,  
23           nim: responseData[key].nim,  
24           nama: responseData[key].nama,  
25           jurusan: responseData[key].jurusan,  
26           asalProvinsi: responseData[key].asalProvinsi,  
27         })  
28       }  
29       setMahasiswas(initMahasiswas);  
30     }  
31     myFetch();  
32   }, [submitCount]);  
33  
34 // handler untuk menambah data mahasiswa di komponen FormMahasiswa  
35 const handleTambahMahasiswa = async (data) => {
```

Mengonlinekan React App (Firebase Hosting)

```
36      await fetch(url, {
37        method: "POST",
38        body: JSON.stringify(data)
39      })
40
41      setSubmitCount(submitCount + 1);
42    }
43
44
45 // handler untuk mengedit data mahasiswa di komponen RowMahasiswa
46 const handleEditMahasiswa = async (id, data) => {
47
48  let url = "https://ilkoom-crud-mahasiswa-default-rtdb.";
49  url += "asia-southeast1.firebaseio.database.app";
50  url += `/mahasiswas/${id}.json`;
51
52  await fetch(url, {
53    method: "PUT",
54    body: JSON.stringify(data)
55  })
56
57  setSubmitCount(submitCount + 1);
58}
59
60 // handler untuk menghapus data mahasiswa di komponen RowMahasiswa
61 const handleHapusMahasiswa = async (e) => {
62
63  let url = "https://ilkoom-crud-mahasiswa-default-rtdb.";
64  url += "asia-southeast1.firebaseio.database.app";
65  url += `/mahasiswas/${e.target.id}.json`;
66
67  await fetch(url, {
68    method: "DELETE"
69  });
70
71  setSubmitCount(submitCount + 1);
72}
73
74 return (
75   <div className="container mt-5">
76
77     <div className="row mt-5">
78       <div className="col">
79         <h1 className="text-center">Tabel Mahasiswa</h1>
80
81         <table className="table mt-4">
82           <thead>
83             <tr>
84               <th>NIM</th>
85               <th>Nama</th>
86               <th>Jurusan</th>
87               <th>Asal Provinsi</th>
88               <th></th>
89             </tr>
90           </thead>
```

Mengonlinekan React App (Firebase Hosting)

```
91      <tbody>
92        {
93          mahasiswa.map((mahasiswa) =>
94            <RowMahasiswa
95              key={mahasiswa.nim}
96              mahasiswa={mahasiswa}
97              onEditMahasiswa={handleEditMahasiswa}
98              onHapusMahasiswa={handleHapusMahasiswa}
99            />
100        )
101      }
102      <RowTambahMahasiswa
103        mahasiswa={mahasiswa}
104        onTambahMahasiswa={handleTambahMahasiswa}
105      />
106    </tbody>
107  </table>
108</div>
109
110</div>
111
112</div>
113)
114}
115
116 export default App;
```

Agar lebih mudah memahami kode yang ada, silahkan pelajari sejenak kode program dari komponen App di CRUD mahasiswa sebelumnya (yang tidak pakai API), lalu bandingkan dengan kode di atas.

Di baris 1 terdapat perintah import useEffect yang nantinya akan kita pakai untuk pemanggilan API. Kemudian di baris 5-7 saya membuat variabel url yang berisi string alamat URL server firebase. Alamat ini bisa diambil dari halaman dashboard firebase.

Di baris 10, state `mahasiswa` didefinisikan dengan nilai awal array kosong "[]". Selain itu kita juga butuh state `submitCount` yang disiapkan sebagai trigger untuk proses re-render.

Masuk ke useEffect hook di baris 13-32, isinya mirip seperti di aplikasi Ilkoom Expense Tracker. Hanya saja untuk mempersingkat kode program saya tidak menulis lagi block try-catch.

Data JSON yang didapat dari server firebase akan "dibuka" oleh perulangan for in di baris 20-21. Tujuannya agar object JSON bisa disusun ulang menjadi array data mahasiswa yang mirip seperti struktur CRUD Mahasiswa tanpa API, meskipun tidak sama persis.

Sebagai contoh, jika data dari firebase berisi JSON berikut:

```
{
  "mahasiswa" : {
    "-MrXA6q40CZAr1Zch673" : {
      "asalProvinsi" : "Sumatera Utara",
```

Mengonlinekan React App (Firebase Hosting)

```
"jurusan" : "Sistem Informasi",
"nama" : "Eka Putra Siregar",
"nim" : "19003037"
},
"-MrXAB6_cuJZwfLcKwJP" : {
  "asalProvinsi" : "Kalimantan Barat",
  "jurusan" : "Teknik Informatika",
  "nama" : "Anissa Lestari",
  "nim" : "19003099"
}
}
}
```

Maka isi state `mahasiswa` setelah perulangan for in akan menjadi:

```
[
{
  "id": "-MrXA6q40CZAr1Zch673",
  "nim": "19003037",
  "nama": "Eka Putra Siregar",
  "jurusan": "Sistem Informasi",
  "asalProvinsi": "Sumatera Utara"
},
{
  "id": "-MrXAB6_cuJZwfLcKwJP",
  "nim": "19003099",
  "nama": "Anissa Lestari",
  "jurusan": "Teknik Informatika",
  "asalProvinsi": "Kalimantan Barat"
}
]
```

Sedikit perbedaan dengan object mahasiswa di versi tanpa API ada di penambahan property `id`. Sebelumnya kita tidak butuh property ini karena property `nim` sudah unik dan bisa dipakai sebagai identitas pembeda dari setiap mahasiswa. Sekarang peran itu lebih cocok diambil alih oleh property `id` yang digenerate firebase.

Di akhir `useEffect` pada baris 32, state `submitCount` di set sebagai *dependency*. Sehingga seluruh kode `useEffect` akan dijalankan ulang jika nilai state `submitCount` berubah.

Di baris 35-43 terdapat fungsi `handleTambahMahasiswa()` yang bertugas untuk menambah data mahasiswa ke server, yakni proses **Create**. Untuk mempersingkat kode program, saya juga tidak menulis block try-catch, langsung fungsi `fetch()` saja:

```
await fetch(url, {
  method: "POST",
  body: JSON.stringify(data)
})
```

Data mahasiswa yang dikirim ada di variabel `data`. Variabel `data` ini berasal dari parameter yang akan dikirim oleh komponen `FormMahasiswa`. Tidak lupa, di akhir fungsi terdapat perintah

`setSubmitCount(submitCount + 1)` untuk menjalankan ulang useEffect hook.

Di baris 46-58, giliran fungsi `handleEditMahasiswa()` yang dipakai untuk proses **Update** data ke server firebase. Sedikit perbedaan dengan versi tanpa API, fungsi ini sekarang menerima 2 buah parameter, yakni `id` dan `data` (akhir baris 46).

Parameter `id` berisi string `id` dari mahasiswa yang akan di edit. `Id` ini merujuk ke karakter acak yang digenerate oleh firebase, seperti `-MrXA6q40CZAr1Zch673`. Nantinya, parameter `id` dan juga parameter `data` akan dikirim oleh komponen `RowMahasiswa`.

Untuk proses edit data, string `id` harus disertakan ke dalam alamat URL. Karena itulah variabel `url` perlu didefinisikan ulang dengan kode berikut:

```
let url = "https://ilkoom-crud-mahasiswa-default-rtbd.";
url += "asia-southeast1.firebaseio.database.app";
url += `/mahasiswas/${id}.json`;
```

Potongan string ``/mahasiswas/${id}.json`` akan diproses misalnya menjadi `"/mahasiswas/-MrXA6q40CZAr1Zch673.json"`

Kemudian untuk pengiriman data ke server, dilakukan oleh fungsi `fetch()` berikut:

```
await fetch(url, {
  method: "PUT",
  body: JSON.stringify(data)
})
```

Di sini saya menggunakan method `PUT` dan bukan `PATCH` karena parameter `data` sudah berisi data mahasiswa lengkap dengan semua property, jadi sebenarnya juga tidak masalah apakah ingin menggunakan `PUT` maupun `PATCH`.

Lanjut, fungsi `handleHapusMahasiswa()` di baris 61-72 dipakai untuk menghapus data mahasiswa. Variabel `url` kembali di definisikan ulang karena `id` mahasiswa yang dihapus harus ditulis ke dalam URL. `Id` tersebut bisa diambil dari `e.target.id`, karena `e` berisi event object yang dikirim dari komponen `RowMahasiswa`.

Setelah itu penghapusan data mahasiswa akan di proses oleh fungsi `fetch()` berikut:

```
await fetch(url, {
  method: "DELETE"
});
```

Dengan fungsi ini, maka aplikasi Ilkoom CRUD Mahasiswa sudah menerapkan REST API lengkap, yakni proses **Create**, **Read**, **Update** dan **Delete**.

Selesai dengan komponen App, kita juga perlu mengubah sedikit kode program di komponen `RowMahasiswa`. Alasannya untuk menambah `id` mahasiswa pada saat submit form edit dan tombol hapus:

Mengonlinekan React App (Firebase Hosting)

my-app\src\components\RowMahasiswa.js

```
1  ...
2  // form di submit
3  const handleFormSubmit = (e) => {
4      ...
5      ...
6      // proses data jika form valid
7      if (formValid) {
8          props.onEditMahasiswa(props.mahasiswa.id, formInput);
9          setEditStatus(false);
10     }
11 }
12
13 return (
14     <React.Fragment>
15     {/* Tampilkan form jika tombol Edit di klik, atau tampilkan row normal */}
16     {editStatus ?
17         <tr>
18             <td colSpan="5">
19                 <form onSubmit={handleFormSubmit} onReset={handleFormReset}>
20                     ...
21                     ...
22                     </form>
23                 </td>
24             </tr>
25             :
26             <tr>
27                 <td>{formInput.nim}</td>
28                 <td>{formInput.nama}</td>
29                 <td>{formInput.jurusan}</td>
30                 <td>{formInput.asalProvinsi}</td>
31                 <td>
32                     <button className="btn btn-secondary me-2"
33                         onClick={handleEditClick}>Edit</button>
34                     <button className="btn btn-danger" id={props.mahasiswa.id}
35                         onClick={props.onHapusMahasiswa}>Hapus</button>
36                 </td>
37             </tr>
38     }
39
40     </React.Fragment >
41 )
42 }
43
44 export default RowMahasiswa;
```

Yang perlu dimodifikasi hanya 2 baris.

Di baris 13, terdapat penambahan `props.mahasiswa.id` sebagai argument pertama saat pemanggilan fungsi `props.onEditMahasiswa()`. Ini adalah fungsi *callback* yang dipakai untuk proses edit mahasiswa. Nantinya `props.mahasiswa.id` akan diterima oleh parameter `id` di fungsi `handleEditMahasiswa()` dalam komponen App. Parameter `id` inilah yang menjadi

patokan dari mahasiswa yang ingin di edit.

Modifikasi kedua berupa perubahan nilai atribut `id` di tombol Hapus (baris 39). Sebelumnya tag `<button>` ini memiliki atribut `id={props.mahasiswa.nim}`, sekarang diubah menjadi `id={props.mahasiswa.id}`. Perubahan ini diperlukan karena yang menjadi patokan mahasiswa sekarang adalah `id` yang didapat dari firebase, bukan lagi nomor `nim`.

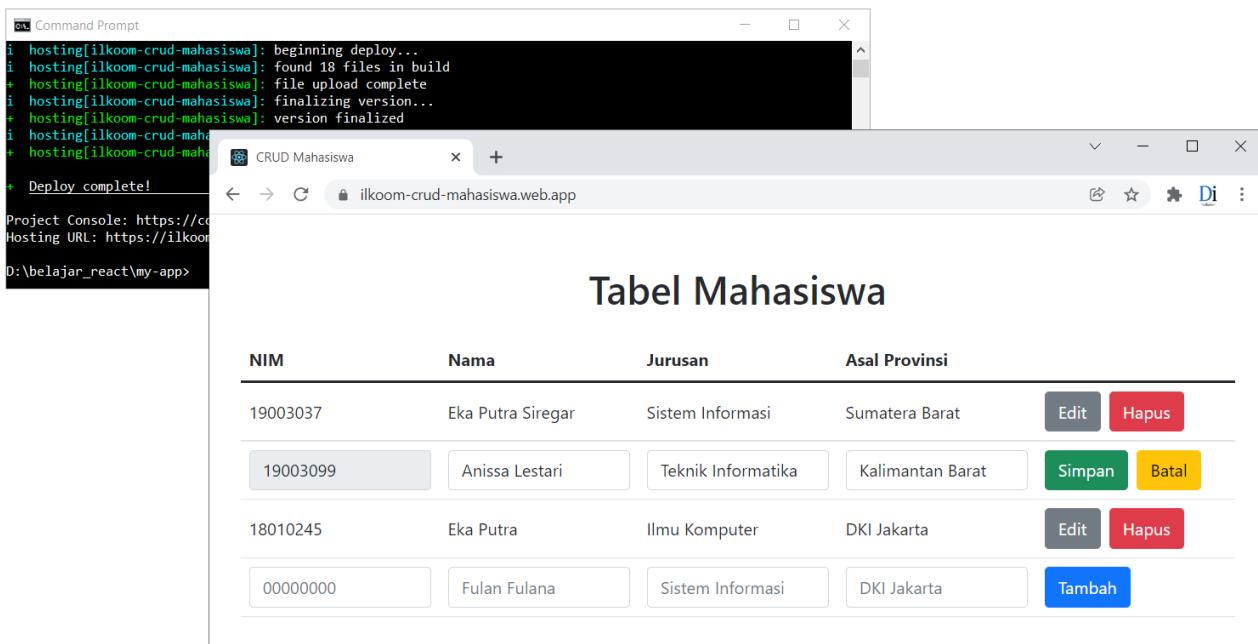
Save perubahan di atas dan test di web browser (jalankan `npm start`). Jika semua fitur sudah berjalan, saatnya update file di firebase hosting.

Seperti biasa, kita harus compile ulang dengan perintah:

```
npm run build
```

Setelah selesai, tinggal upload dengan perintah:

```
firebase deploy
```



Gambar: Aplikasi Ilkoom CRUD Mahasiswa sudah terhubung ke firebase realtime database

Akhirnya..., file kita sudah update dan bisa diakses dari internet.

Dalam bab ini kita telah membahas cara upload file React ke Firebase Hosting serta mengupdate dua mini project agar bisa menyimpan data ke Firebase Realtime Database.

Materi ini sebenarnya lebih ke "bonus" karena tidak berhubungan langsung dengan kode program React. Tapi bisa "showoff" karya yang sudah kita buat dengan React menjadi kebanggaan tersendiri sekaligus penyemangat belajar.

Penutup React Uncover

Akhirnya.., setelah lebih dari 600 halaman (mencakup 20 bab), kita sampai di penghujung buku **React Uncover**. Semoga semua materi ini bisa menjadi pondasi dasar yang kuat bagi anda untuk memahami apa itu framework/Library React dan bagaimana cara penggunaannya.

Pertanyaan utama, *apakah ini sudah selesai?*

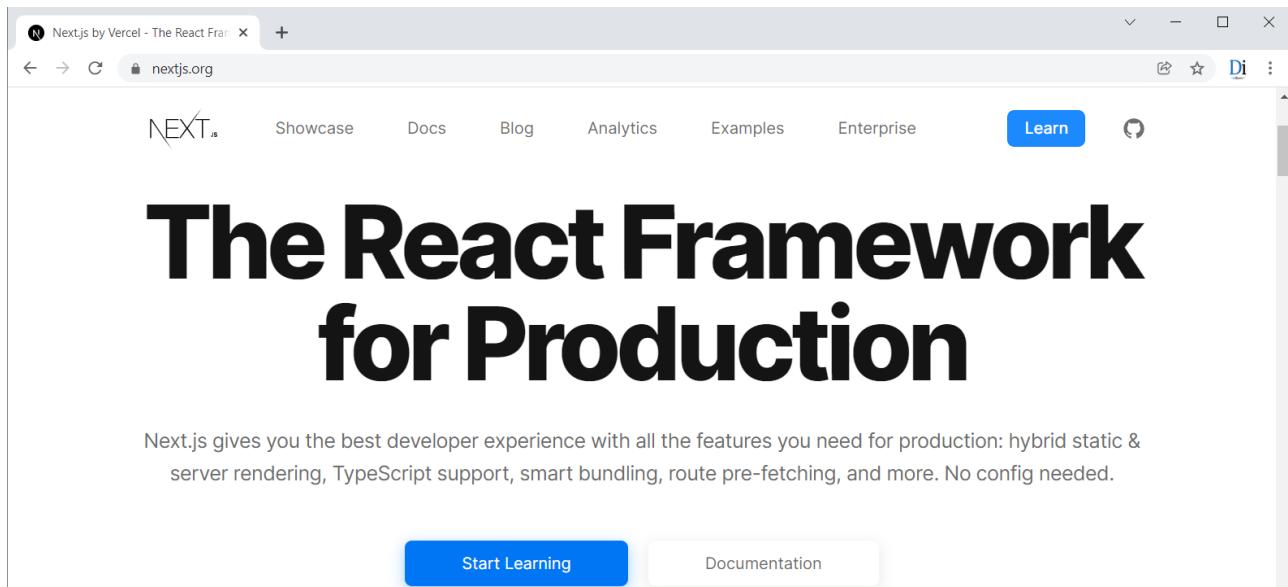
Well... seperti yang bisa di tebak, jawaban ideal adalah: **Belum**.

Meskipun kita sudah mempelajari cukup banyak materi dasar React, tapi ini baru sebagian kecil dari "dunia React" yang sesungguhnya. Jika tertarik, bisa di explore lebih lanjut hal-hal berikut:

- **Advanced State Management.** Dalam buku ini kita baru membahas 1 cara untuk pemrosesan state, yakni menggunakan `useState hook`. Untuk aplikasi yang lebih besar, bisa saja terdapat puluhan state yang tersebar di berbagai komponen. Jika menggunakan `useState hook`, maka setiap data harus dikirim sebagai `props`.
Alternatif lain, tersedia fitur state management seperti `useReducer` dan `Context API`. Keduanya masih fitur bawaan dari React. Atau jika itu belum cukup, tersedia library `Redux` untuk pengelolaan state.
- **React Router.** Aplikasi React yang kita buat sepanjang buku ini hanya menggunakan 1 alamat URL saja, karena memang berbentuk SPA (Single Page Application). Akan tetapi aplikasi SPA tidak harus memiliki 1 URL. Dengan library `react router`, kita bisa mengatur alamat URL untuk SPA.
- **React Native.** Salah satu keunggulan belajar React adalah bisa lanjut belajar `react native` untuk pembuatan aplikasi android. Konsep dasar React seperti `component`, `JSX`, `props` dan `state` akan tetap terpakai di `react native` nanti.
- **Next.js.** Alasan dibalik penamaan React sebagai "library" adalah karena fiturnya yang relatif terbatas. Jika butuh paket yang lebih lengkap, Next.js bisa menjadi solusi.

Next.js memposisikan diri sebagai "The React Framework for Production", yang berisi berbagai fitur tambahan dari React biasa. Kelemahan aplikasi SPA yang susah untuk SEO diatasi dengan konsep `pre-rendering`. Selain itu terdapat juga fitur routing bawaan dan kemudahan integrasi dengan server back-end.

Framework Next.js akan membawa skill React anda ke level yang lebih tinggi.



Gambar: Tampilan Web Next.js di <https://nextjs.org>

Pelajari Materi Web Programming Lain

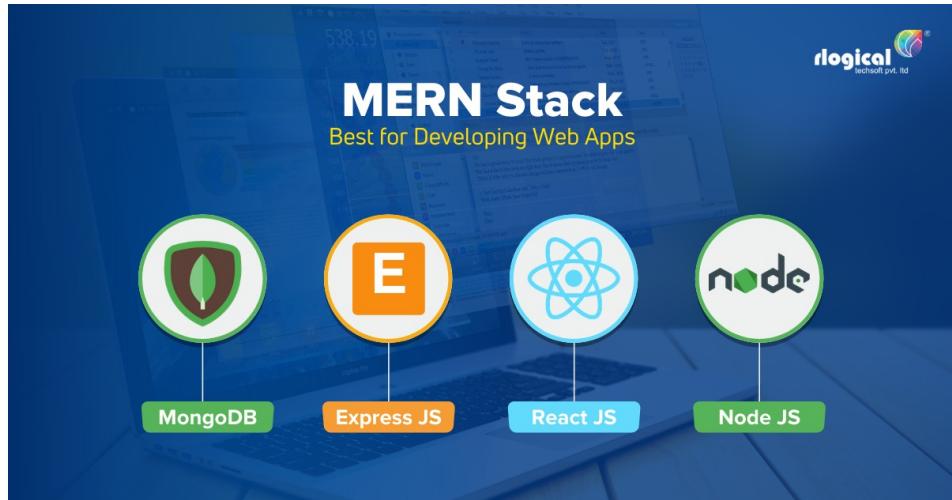
Alternatif lanjutan adalah pelajari materi web programming yang masih berhubungan dengan React. Jika anda "lompat" ke React tanpa JavaScript dasar, maka saatnya untuk memperdalam materi JavaScript.

Sebagaimana yang kita lihat di pembuatan mini project, aplikasi React tetap butuh JavaScript native. Pengetahuan materi *Date object*, *Event object*, serta berbagai method bawaan JavaScript sangat membantu dalam pembuatan aplikasi.

Materi CSS juga cukup penting, sebab posisi React ada di front-end. Framework CSS seperti **Bootstrap** dan **Tailwind CSS** akan membantu mempercepat proses desain tampilan.

Yang tidak kalah menarik, masuk ke fullstack JavaScript. Untuk di front-end kita sudah memiliki React dan JavaScript (tentunya juga HTML dan CSS). Sedangkan untuk di sisi back-end tersedia NodeJS+ExpressJS sebagai server dan MongoDB sebagai database. Stack ini dikenal juga dengan istilah **MERN** (**MongoDB**, **ExpressJS**, **React**, **NodeJS**).

Namun stack teknologi ini tidak dalam bentuk baku. React dan JavaScript tidak harus bergantung dengan JavaScript back-end. Selama server back-end bisa mengirim data dalam bentuk API, itu juga bisa dipakai. Termasuk menggunakan framework Laravel di back-end, atau layanan BaaS (Backend as a Service) seperti Firebase. Tinggal kita yang ingin memutuskan akan belajar materi apa.



Gambar: Ilustrasi dari MERN Stack (sumber gambar: rlogical.com)

Perbanyak Studi Kasus / Mini Project

Berbekal skill React dan server penyedia API, anda juga bisa coba membuat berbagai studi kasus lain. Misalnya bagaimana dengan aplikasi prakiraan cuaca? Cukup banyak web yang menyediakan data API tersebut, seperti openweathermap.org dan data.bmkg.go.id. Atau mempelajari API dari facebook, twitter atau instagram.

Dengan memanfaatkan database firebase, juga bisa buat berbagai aplikasi CRUD lain seperti pendaftaran mahasiswa baru, inventori kantor, dsb.

Akhir kata, semoga materi yang ada di buku **React Uncover** ini bisa bermanfaat. Mohon maaf jika ada kata-kata atau penjelasan yang salah. Jika ada typo, salah ketik atau kode program yang error, saya sangat berterimakasih jika di infokan via WA atau email.

Terima kasih juga atas dukungannya dengan membeli versi asli buku React Uncover (bukan dari sumber bajakan / copy dari teman). Donasi pembelian buku ini menjadi penyemangat saya untuk bisa terus berkarya.

Sampai jumpa di buku DuniaIlkom selanjutnya, semoga ilmu yang di dapat bisa berkah dan bermanfaat :)

Daftar Pustaka

Sepanjang penulisan buku React Uncover, saya mengumpulkan bahan dari berbagai sumber. Anda bisa mengunjungi daftar pustaka ini untuk menambah pengetahuan seputar React:

- React Documentation : <https://reactjs.org/docs/getting-started.html>
- Codecademy: <https://www.codecademy.com/learn/react-101>
- Traversy Media: <https://www.youtube.com/user/TechGuyWeb>
- Dev Ed: <https://www.youtube.com/channel/UClb90NQQcskPUGDIXsQEz5Q>
- Blog Logrocket: <https://blog.logrocket.com/tag/react/>
- Stackoverflow React: <https://stackoverflow.com/questions/tagged/reactjs>