



MODUL PRAKTIKUM

PEMROGRAMAN BERBASIS FRAMEWORK

Modul 4

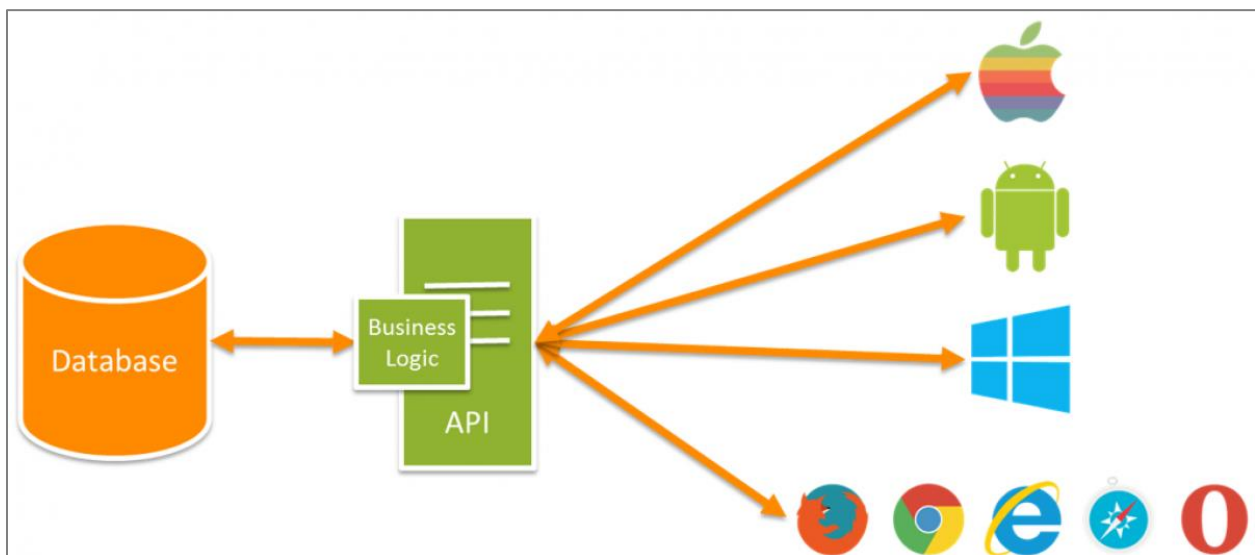
- Interaksi dengan API (GET)
- Interaksi dengan API (fake API)
- Interaksi dengan API (DELETE)
- Interaksi dengan API (POST)

JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
2019

API

API adalah singkatan dari *Application Programming Interface*, dan memungkinkan *developer* untuk mengintegrasikan dua bagian dari aplikasi atau dengan aplikasi yang berbeda secara bersamaan. *API* terdiri dari berbagai elemen seperti *function*, *protocols*, dan *tools* lainnya yang memungkinkan *developers* untuk membuat aplikasi.

Tujuan penggunaan *API* adalah untuk mempercepat proses *development* dengan menyediakan *function* secara terpisah sehingga *developer* tidak perlu membuat fitur yang serupa. Penerapan *API* akan sangat terasa jika fitur yang diinginkan sudah sangat kompleks.



Gambar 1. API

API dapat anda temui dalam kehidupan sehari-hari seperti saat anda memesan hotel, mengirimkan pesan, memesan makanan secara *online* maupun ketika mengunduh sebuah *software*.

Kenapa menggunakan API?

API membuat pemrograman menjadi lebih mudah. Kebutuhan kita sebagai pelanggan dan khususnya bagi *developer* sangat dimudahkan dengan adanya API. Dengan melihat hal tersebut, peran dari API sendiri sangat berat terlebih untuk membuat tampilan sebuah aplikasi menjadi interaktif, mudah untuk digunakan, dan bersahabat untuk pengguna. Tidak hanya itu, API juga digunakan untuk berkomunikasi antara layanan-layanan. API memiliki peran yang sangat penting dalam teknologi.

Keuntungan menggunakan API Bagi Para Developer Antara Lain:

1. Aplikasi.

API membantu kinerja dari aplikasi lebih cepat dan fleksibel seperti layanan dan informasi yang diberikan karena API dapat memasuki komponen-komponen aplikasi.

2. Kustomisasi

Dengan API, kustomisasi untuk konten dan layanan dapat dilakukan sesuai kebutuhan dan keinginan.

3. Fleksibel

API membuat layanan menjadi lebih fleksibel. Hal tersebut karena API mendukung data migrasi lebih baik dan informasi yang didapat ditinjau lebih dekat.

4. Integrasi / integration

API dapat menjamin pengiriman informasi lebih lancar dikarenakan API memungkinkan konten tertanam dari aplikasi maupun situs dengan mudah. Hal tersebut memberikan pengalaman yang terintegrasi bagi pengguna.

5. Lebih banyak data

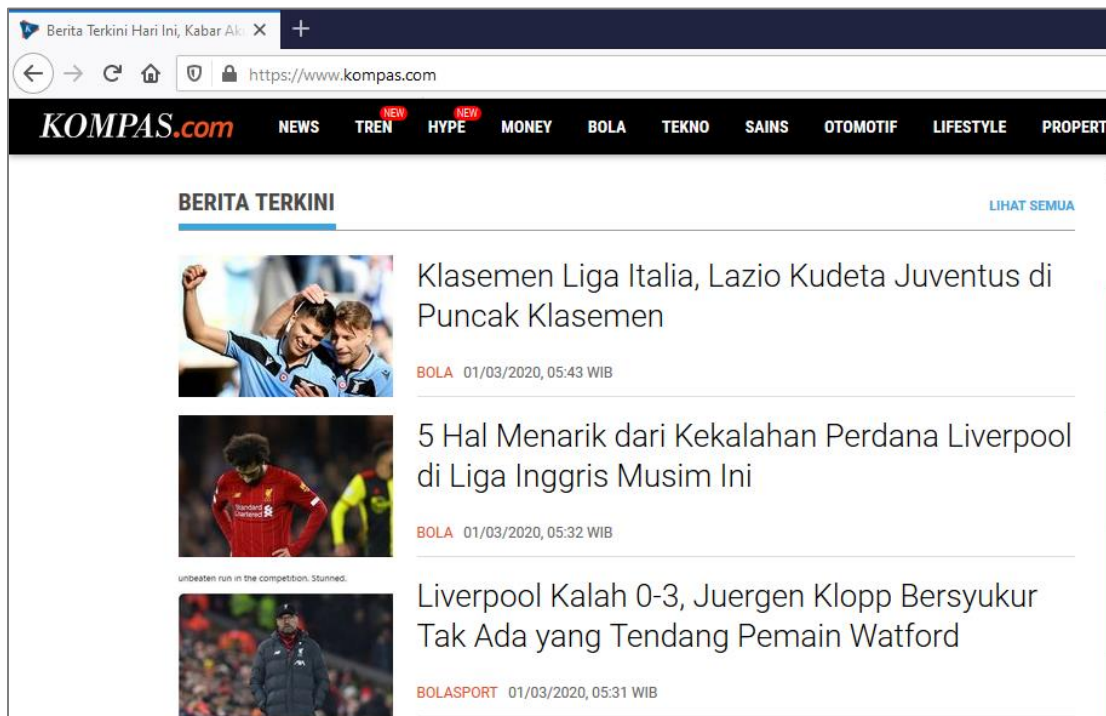
API memberikan banyak pilihan karena semua informasi yang dihasilkan di tingkat pemerintah tersedia untuk setiap warga negara.

Praktikum 1

Interaksi dengan API menggunakan method GET

1.1 Contoh Program

Contoh program yang akan kita buat adalah list artikel (*blog post*) pada suatu halaman website, seperti pada contoh Gambar 1.1.



Gambar 1.1. List berita

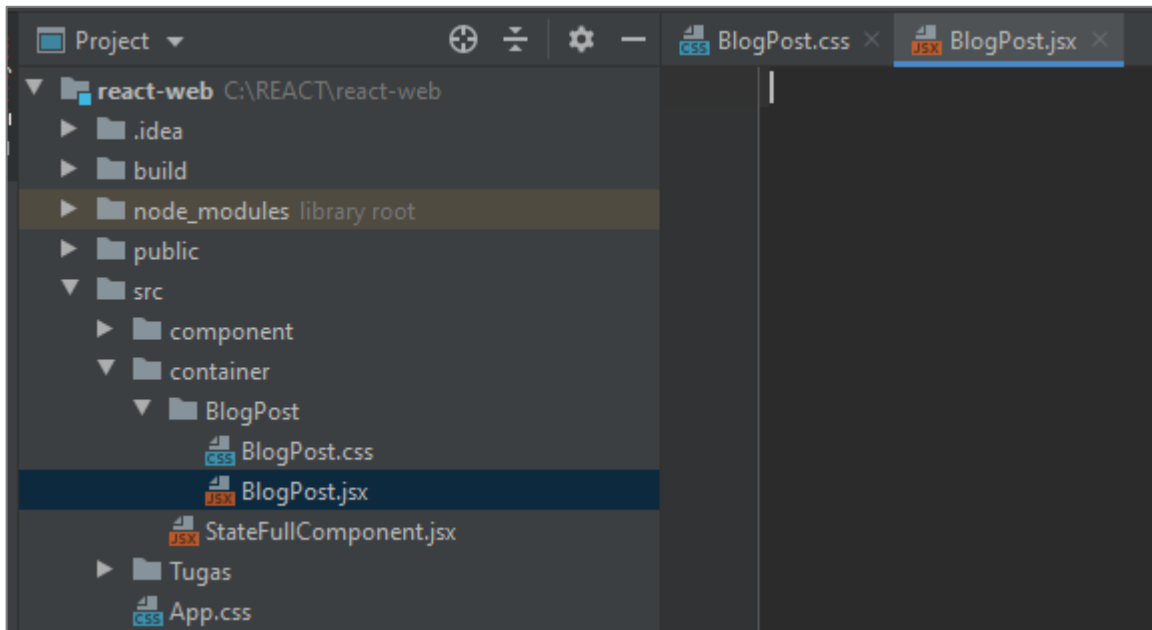
1.2 Data yang dipakai

Data yang akan kita pakai adalah data *dummy* atau *Fake Online REST API for Testing and Prototyping* dari halaman API <https://jsonplaceholder.typicode.com>.

1.3 Langkah Praktikum

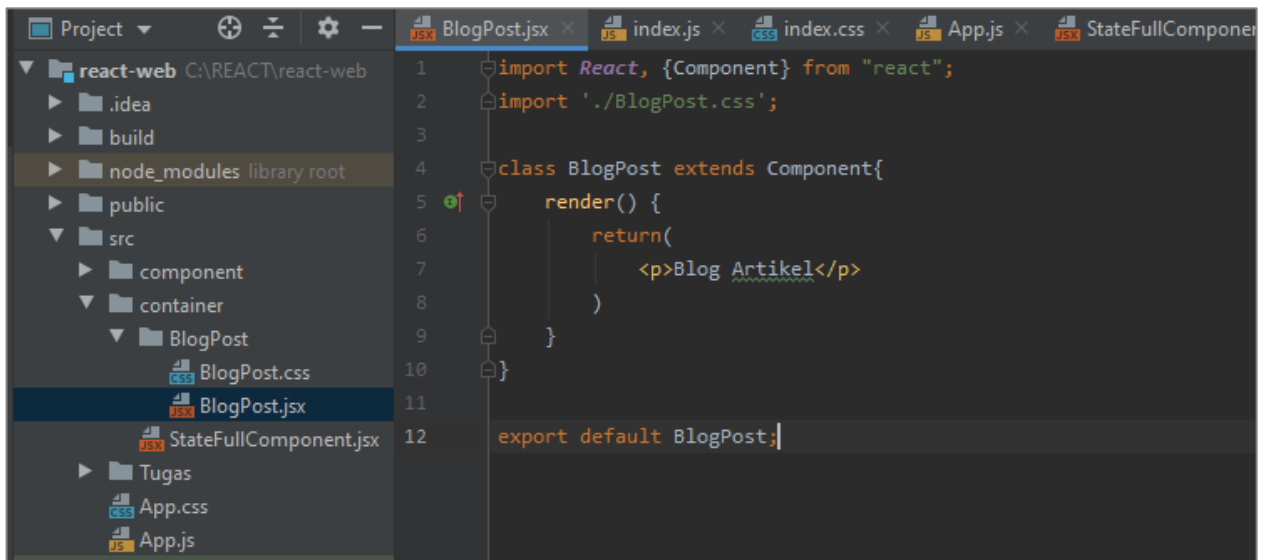
Dalam penggunaan API pada suatu website, maka data yang akan kita pakai adalah data dinamis dan memerlukan operasi logic pada ReactJS. Sehingga kita akan menggunakan *statefull component* ReactJS untuk membuatnya.

1. Buka Project React pada pertemuan sebelumnya dan jalankan **"npm start"** menggunakan *cmd* dalam direktori tersebut.
2. Buat folder baru bernama **"BlogPost"** pada folder **container** (*statefull component*).
3. Buat file **BlogPost.jsx** dan **BlogPost.css** di dalam folder **"BlogPost"**, seperti pada Gambar 1.2.



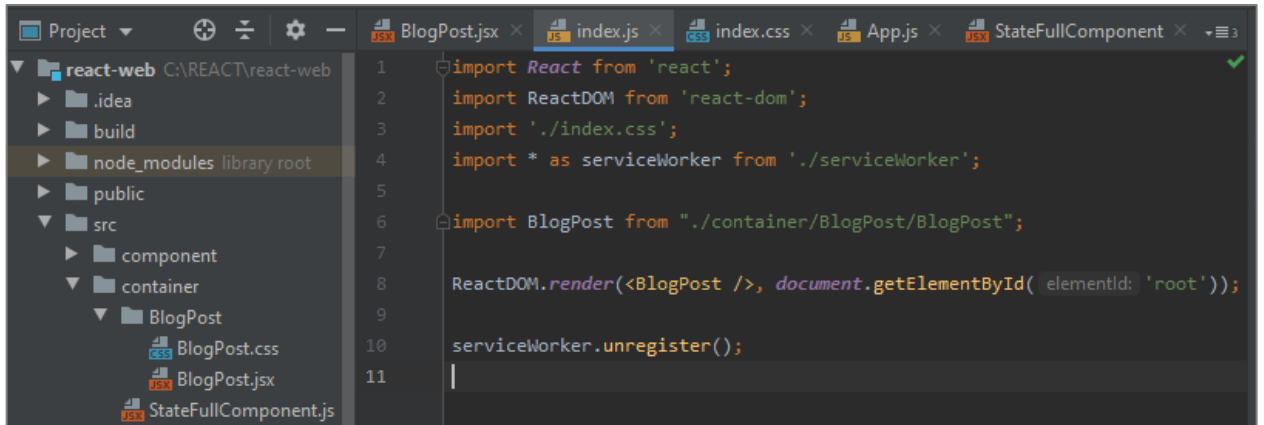
Gambar 1.2. Pembuatan folder untuk statefull component

4. Buka file **BlogPost.jsx** dan ketikkan kode seperti Gambar 1.3.



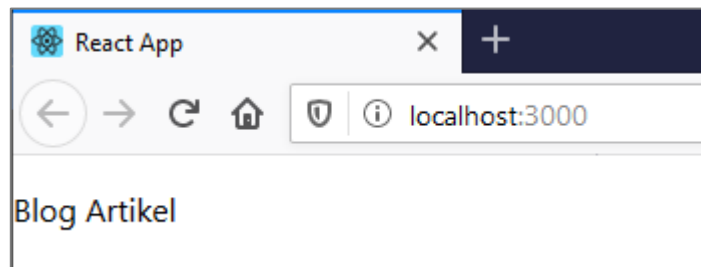
Gambar 1.3. Statefull component BlogPost

5. Pada file **index.js**, lakukan import component **BlogPost** seperti Gambar 1.4.



Gambar 1.4. Import component BlogPost

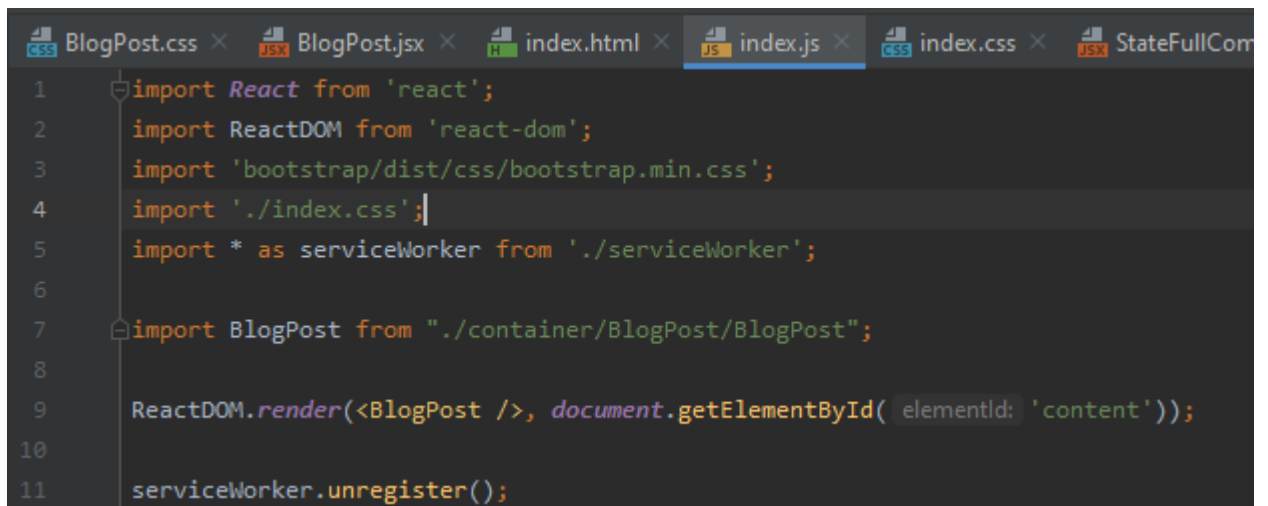
6. Pada web browser akan tampil seperti pada Gambar 1.5.



Gambar 1.5. Tampilan Browser

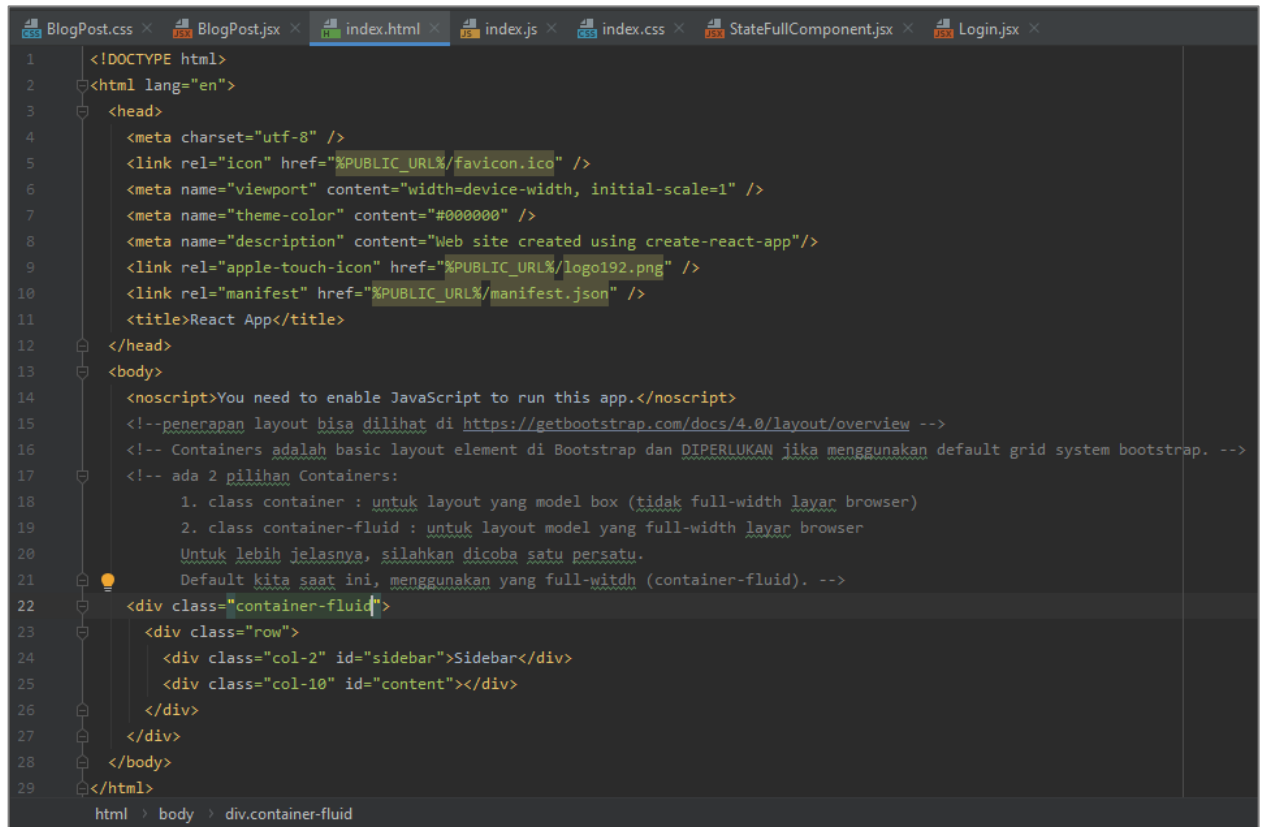
Tahapan selanjutnya adalah perbaikan tampilan sebuah website untuk mempercantik halaman website tersebut dengan menggunakan **Bootstrap** yang umum digunakan.

7. Import css **bootstrap.min.css** (css bootstrap yang sudah dikompresi) ke dalam **index.js** (seperti Gambar 1.6). Jika css tidak ditemukan, install lewat cmd dengan perintah **"npm install bootstrap"**



Gambar 1.6. Import bootstrap css

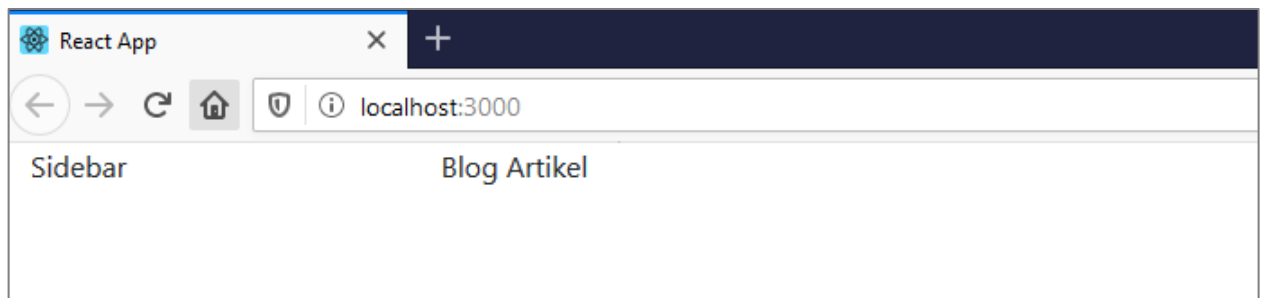
8. Modifikasi file `index.html` pada folder `"public"` seperti Gambar 1.7. Cermati *code program* yang ada dalam gambar!.



```
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 name="description" content="Web site created using create-react-app"/>
9   <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
10  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
11  <title>React App</title>
12 </head>
13 <body>
14  <noscript>You need to enable JavaScript to run this app.</noscript>
15  <!-- penerapan layout bisa dilihat di https://getbootstrap.com/docs/4.0/layout/overview -->
16  <!-- Containers adalah basic layout element di Bootstrap dan DIPERLUKAN jika menggunakan default grid system bootstrap. -->
17  <!-- ada 2 pilihan Containers:
18    1. class container : untuk layout yang model box (tidak full-width layar browser)
19    2. class container-fluid : untuk layout model yang full-width layar browser
20    Untuk lebih jelasnya, silahkan dicoba satu persatu.
21    Default kita saat ini, menggunakan yang full-width (container-fluid). -->
22  <div class="container-fluid">
23    <div class="row">
24      <div class="col-2" id="sidebar">Sidebar</div>
25      <div class="col-10" id="content"></div>
26    </div>
27  </div>
28 </body>
29 </html>
```

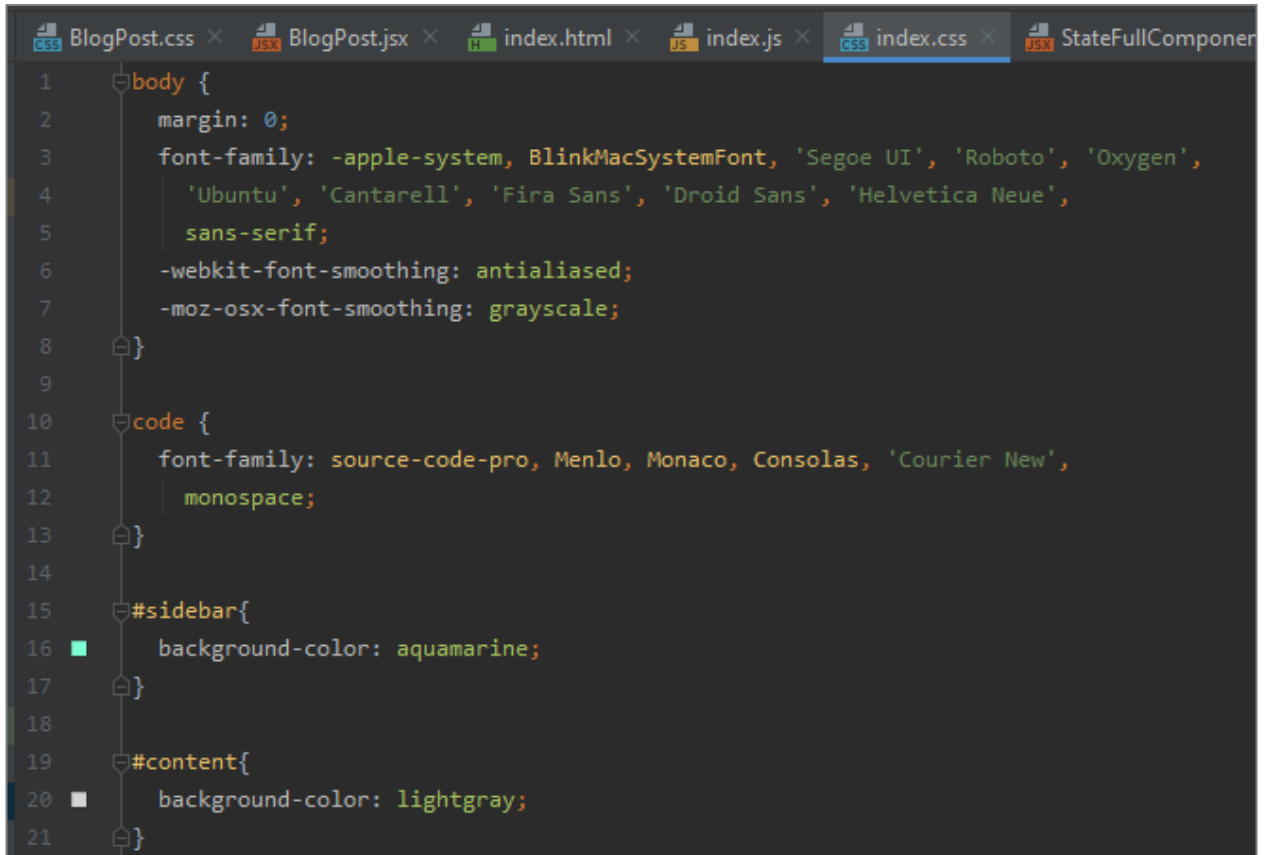
Gambar 1.7. Modifikasi index.html

9. Amati tampilan yang ada pada browser (seperti Gambar 1.8)



Gambar 1.8. Tampilan hasil modifikasi

10. Buka file `index.css` dan tambahkan code css seperti Gambar 1.9, untuk menambah sedikit style pada halaman web



Gambar 1.9. Penambahan code css

11. Perhatikan kembali browser, dan lihat hasil tampilan seperti Gambar 1.10.



Gambar 1.10. Hasil penambahan css

Kita ingin sebuah website memiliki tampilan seperti pada Gambar 1.1. Dengan minimal ada gambar artikel, judul, dan deskripsi artikel. Maka contoh *data dummy* yang akan kita pakai bisa menggunakan data dari <http://placeimg.com> contoh <http://placeimg.com/120/120/any>.

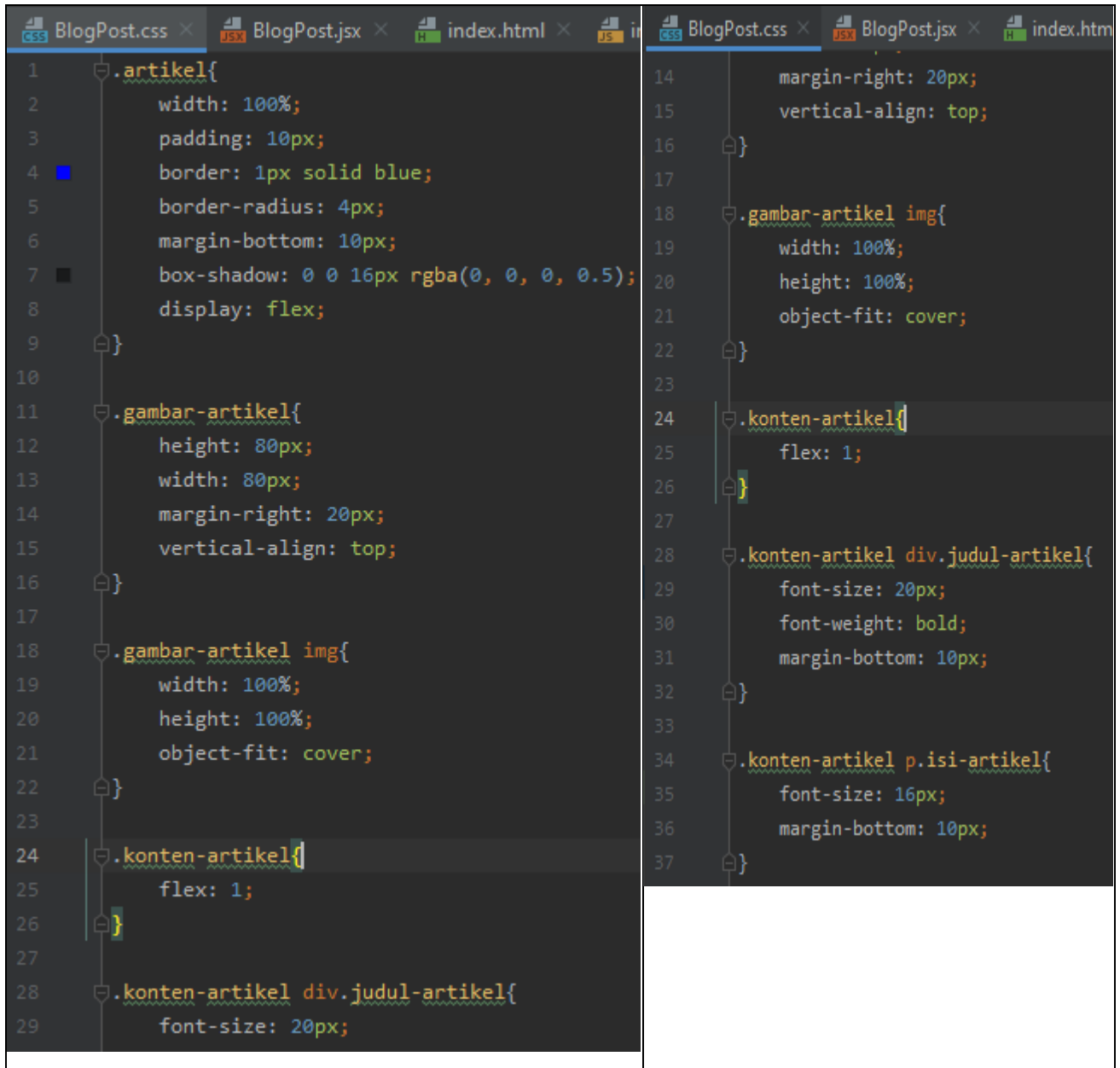
Tahapan edit tampilan post artikel:

12. Ubah kode program untuk *statefull component* **BlogPost.jsx** menjadi seperti Gambar 1.11


```
BlogPost.css x BlogPost.jsx x index.html x index.js x index.css x StateFullComponent.jsx x
1 import React, {Component} from "react";
2 import './BlogPost.css';
3
4 class BlogPost extends Component{
5   render() {
6     return(
7       <div class="post-artikel">
8         <h2>Daftar Artikel</h2>
9         <div class="artikel">
10           <div class="gambar-artikel">
11             
12           </div>
13           <div class="konten-artikel">
14             <div class="judul-artikel">Judul Artikel</div>
15             <p class="isi-artikel">Isi Artikel</p>
16           </div>
17         </div>
18       </div>
19     )
20   }
21 }
22
23 export default BlogPost;
```

Gambar 1.11. Edit kode program BlogPost

13. Tambahkan custom css ke [BlogPost.css](#) seperti Gambar 1.12



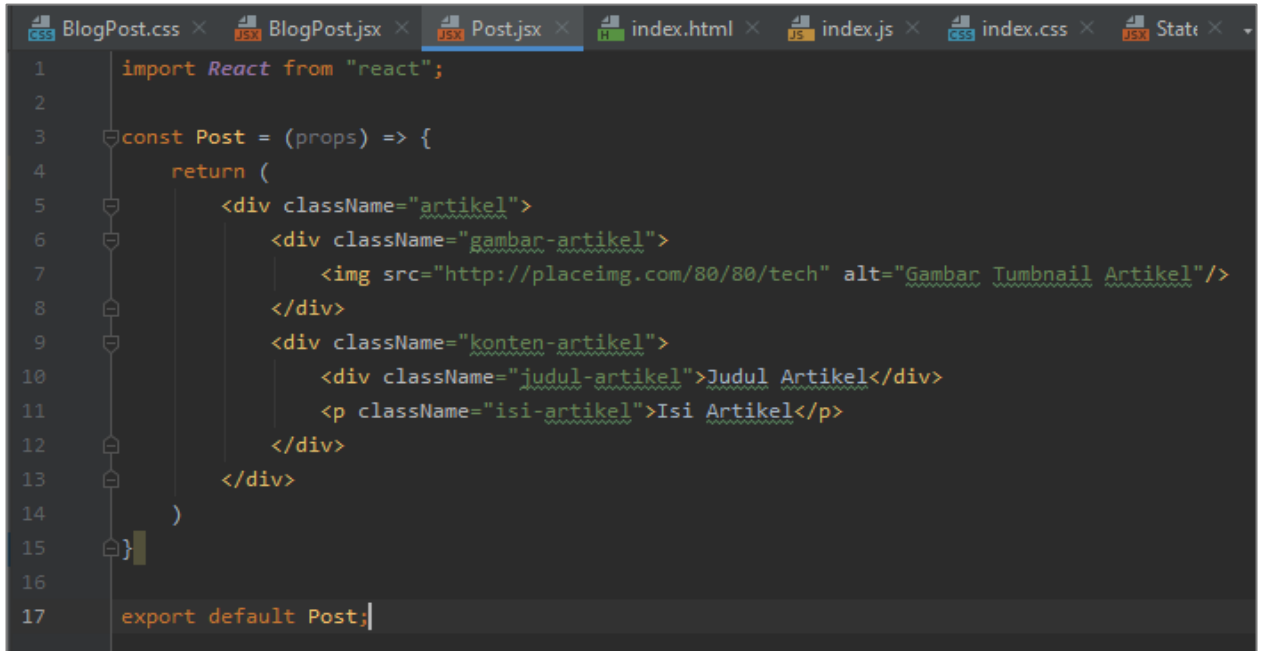
Gambar 1.12. Edit kode program BlogPost.css

14. Perhatikan tampilan browser.

Pemindahan dari *statefull component* ke *stateless component*

Pada component BlogPost (lihat Gambar 1.11), baris 9-17 merupakan daftar artikel yang bisa jadi dalam sebuah website berisi lebih dari 1 (satu) list artikel. Baris 9-17 dapat dipindah ke *stateless component* untuk dapat digunakan ulang (dipanggil kembali) karena fungsi dari bagian tersebut hanya mengembalikan deskripsi singkat artikel (bukan operasi logic).

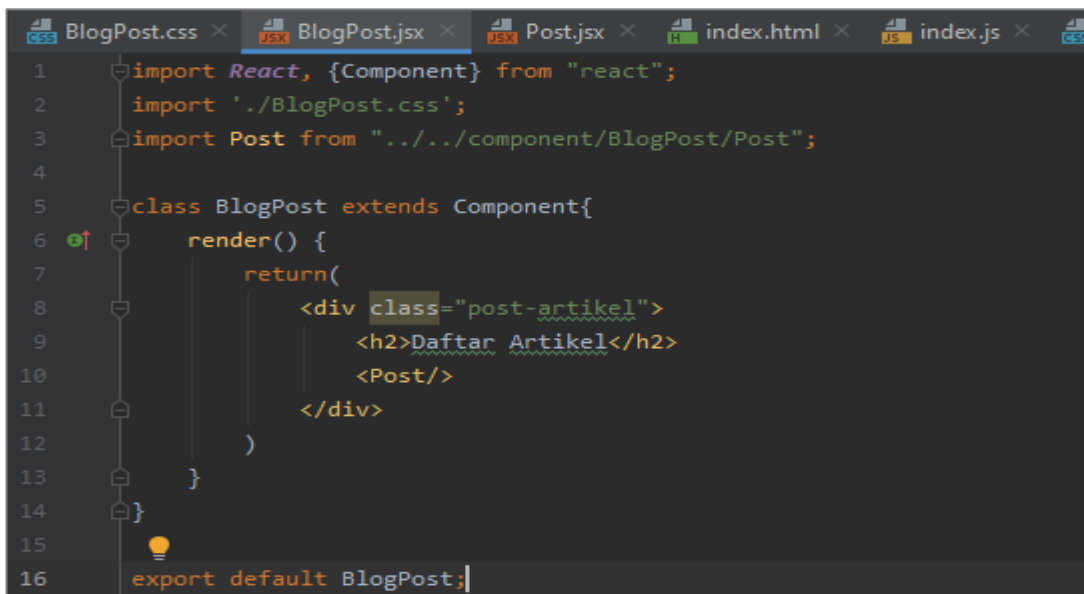
15. Buat folder **BlogPost** pada folder **component** (*stateless component*), lalu buat file **Post.jsx**
16. Potong (*cut*) baris 9-17 pada *statefull component* **BlogPost.jsx** ke *stateless component* **Post.jsx**, dan modifikasi **Post.jsx** seperti Gambar 1.13.



```
1 import React from "react";
2
3 const Post = (props) => {
4   return (
5     <div className="artikel">
6       <div className="gambar-artikel">
7         
8       </div>
9       <div className="konten-artikel">
10        <div className="judul-artikel">Judul Artikel</div>
11        <p className="isi-artikel">Isi Artikel</p>
12      </div>
13    </div>
14  )
15 }
16
17 export default Post;
```

Gambar 1.13. Kode program Post.jsx

17. Untuk *statefull component* **BlogPost.jsx** pada baris 10, panggil *stateless component* **Post.jsx** seperti Gambar 1.14.



```
1 import React, {Component} from "react";
2 import './BlogPost.css';
3 import Post from "../../component/BlogPost/Post";
4
5 class BlogPost extends Component{
6   render() {
7     return(
8       <div class="post-artikel">
9         <h2>Daftar Artikel</h2>
10        <Post/>
11      </div>
12    )
13  }
14 }
15
16 export default BlogPost;
```

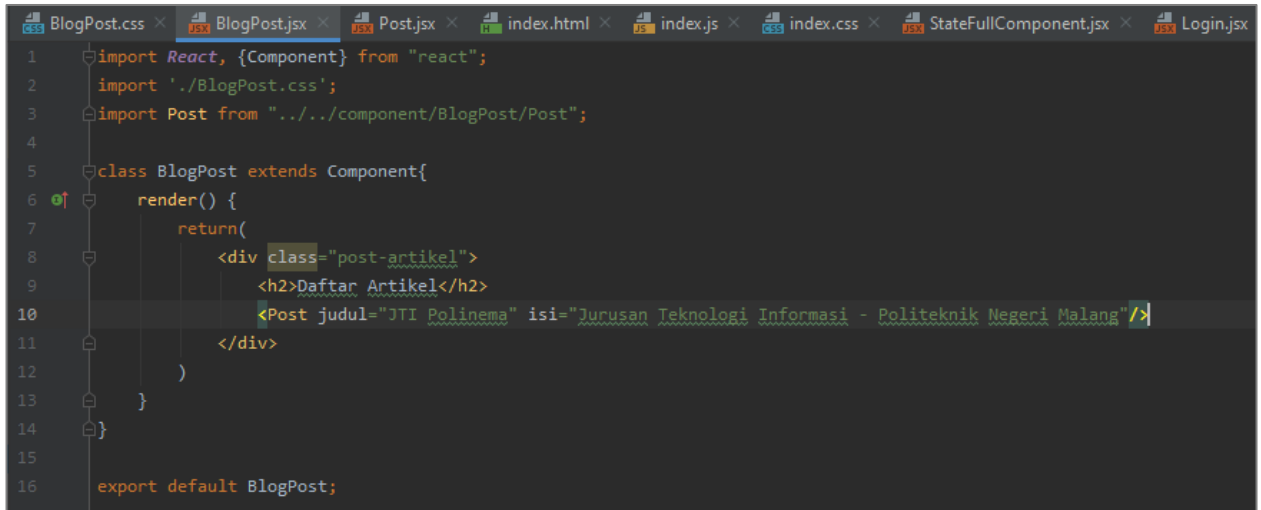
Gambar 1.14. Component BlogPost memanggil component Post

18. Perhatikan hasil tampilan browser, apa yang terjadi?

Muat Data Dinamis.

Bagaimana caranya untuk dapat membuat data dinamis (lebih dari 1 artikel) dimana data Judul dan Deskripsi pada artikel didapat dari API?

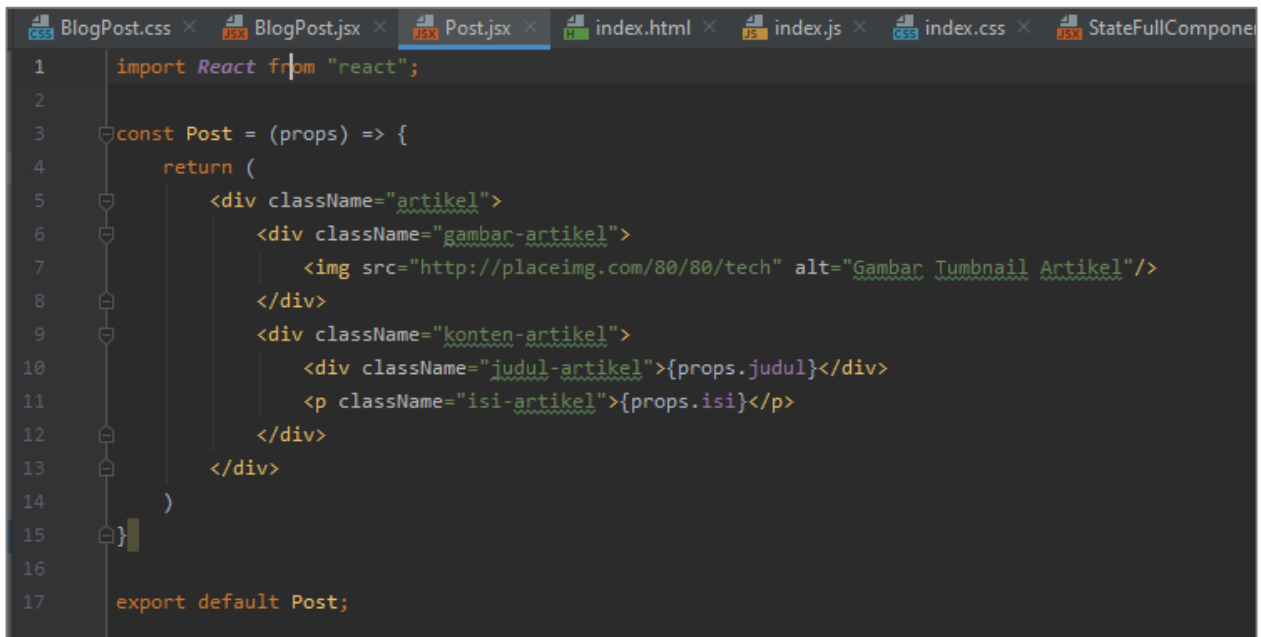
19. Pada *statefull component* **BlogPost.jsx**, tambahkan parameter yang ingin dilempar ke *stateless component* untuk ditampilkan. Kode program bisa dilihat pada Gambar 1.15.



```
1 import React, {Component} from "react";
2 import './BlogPost.css';
3 import Post from "../component/BlogPost/Post";
4
5 class BlogPost extends Component{
6   render() {
7     return(
8       <div class="post-artikel">
9         <h2>Daftar Artikel</h2>
10        <Post judul="JTI Polinema" isi="Jurusan Teknologi Informasi - Politeknik Negeri Malang"/>
11      </div>
12    )
13  }
14 }
15
16 export default BlogPost;
```

Gambar 1.15. Penambahan parameter pada BlogPost

20. Setelah itu pada *stateless component* **Post.jsx** tangkap parameter yang dilempar oleh *statefull component* seperti pada Gambar 1.16 dan lihat pada browser apa yang terjadi!.



```
1 import React from "react";
2
3 const Post = (props) => {
4   return (
5     <div className="artikel">
6       <div className="gambar-artikel">
7         
8       </div>
9       <div className="konten-artikel">
10        <div className="judul-artikel">{props.judul}</div>
11        <p className="isi-artikel">{props.isi}</p>
12      </div>
13    </div>
14  )
15 }
16
17 export default Post;
```

Gambar 1.16. Kode program Post

21. Simpan, dan amati apa yang terjadi pada browser kalian!.

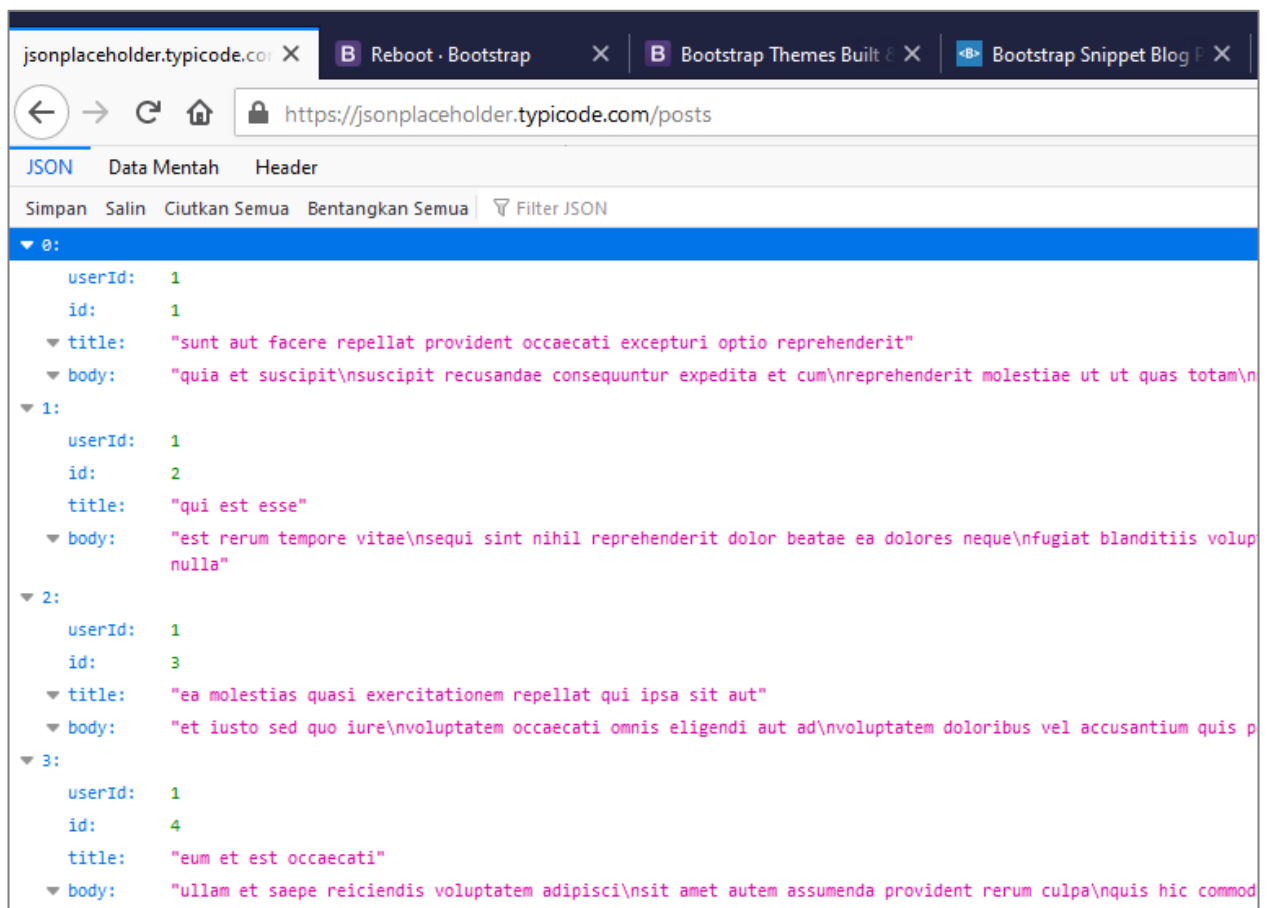
Mengambil data Post/Artikel dari API.

Bagaimana caranya untuk mendapatkan list artikel berdasarkan data json dari web API (contohnya: <https://jsonplaceholder.typicode.com/posts>) ?

Kita gunakan *life cycle component* yaitu `componentDidMount()` dimana ketika komponen selesai di-mount-ing, program akan memanggil API.

22. Gunakan `state` untuk menyimpan data hasil *request* dari API

23. data API yang akan kita gunakan adalah data *dummy* dari <https://jsonplaceholder.typicode.com/posts>, dimana memiliki 4 element data yaitu *userid*, *id*, *title*, *body* (seperti pada Gambar 1.17)



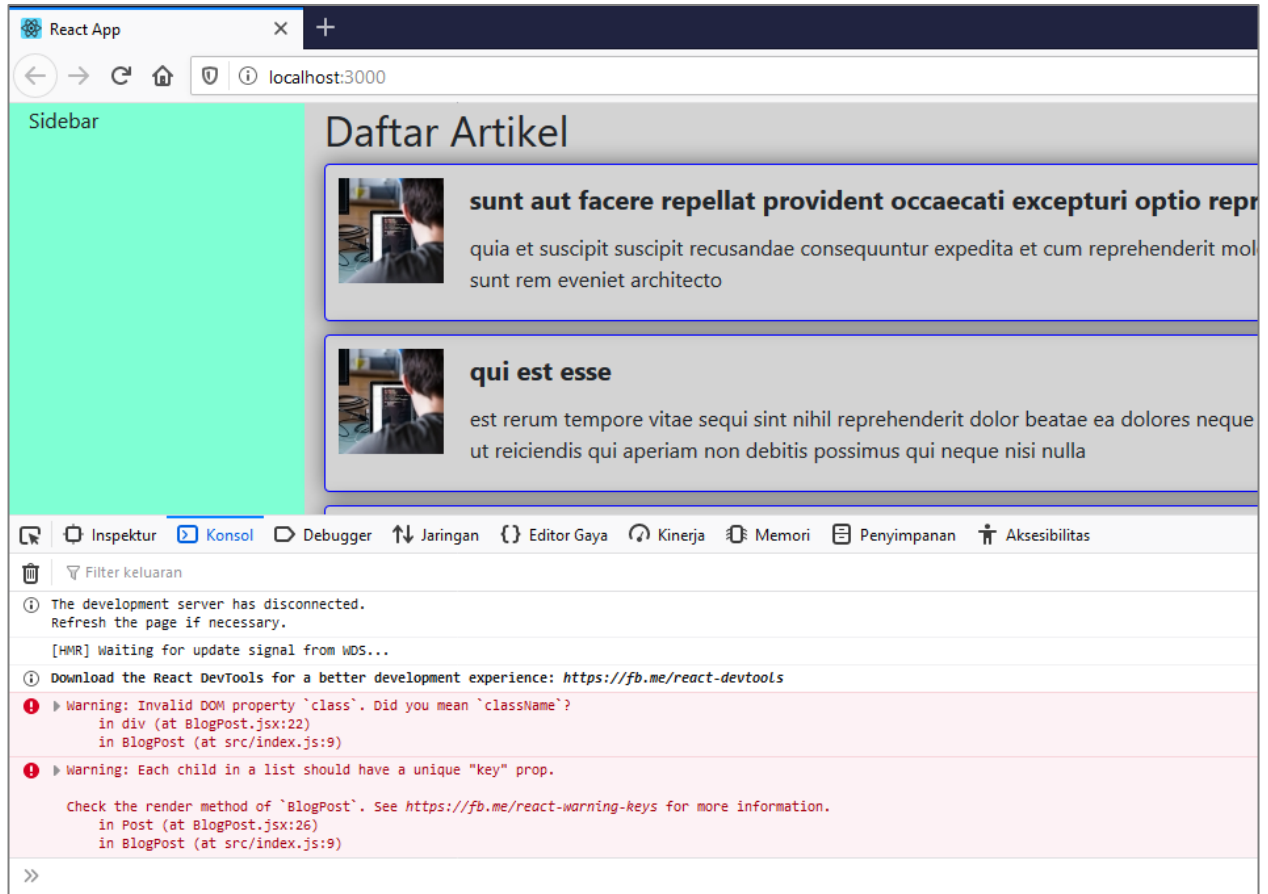
Gambar 1.17. Data response json dari web API

24. Edit pada *statefull component* `BlogPost.jsx` seperti pada Gambar 1.18 dan perhatikan dengan seksama akan penjelasan di beberapa baris kode program tersebut.

```
BlogPost.css x BlogPost.jsx x Post.jsx x index.html x index.js x index.css x StateFullComponent.jsx x Login.jsx x
5 class BlogPost extends Component{
6   state = { // komponen state dari React untuk statefull component
7     listArtikel: [] // variabel array yang digunakan untuk menyimpan data API
8   }
9
10  componentDidMount() { // komponen untuk mengecek ketika component telah di-mount-ing, maka panggil API
11    fetch( input: 'https://jsonplaceholder.typicode.com/posts') // alamat URL API yang ingin kita ambil datanya
12      .then(response => response.json()) // ubah response data dari URL API menjadi sebuah data json
13      .then(jsonHasilAmbilDariAPI => { // data json hasil ambil dari API kita masukkan ke dalam listArtikel pada state
14        this.setState( state: {
15          listArtikel: jsonHasilAmbilDariAPI
16        })
17      })
18  }
19
20  render() {
21    return(
22      <div class="post-artikel">
23        <h2>Daftar Artikel</h2>
24        {
25          this.state.listArtikel.map(artikel => { // looping dan masukkan untuk setiap data yang ada di listArtikel ke variabel artikel
26            return <Post judul={artikel.title} isi={artikel.body}/> // mappingkan data json dari API sesuai dengan kategorinya
27          })
28        }
29      </div>
30    )
31  }
32 }
```

Gambar 1.18. Penambahan componentDidMount pada *statefull component* BlogPost

25. Lihat hasilnya pada browser. Kemudian **klik kanan** pada browser pilih "**inspect element**" kemudian pilih tab "**console**". *Refresh* browser dan amati apa yang terjadi.
26. Jika terlihat seperti pada Gambar 1.19, maka terjadi kesalahan pada program yang kita buat.



Gambar 1.19. Error pada browser

27. Jika terjadi hal demikian, hal ini terjadi karena dalam react **"class"** dalam tag html harus ditulis menjadi **"className"**. selain itu, pada *statefull component* yang dinamis, harus ada **"UNIQUE KEY"** pada tiap komponen yang diproses sehingga komponen perlu diberi **UNIQUE KEY**.
28. **UNIQUE KEY** dapat diambil dari element yang ada pada data API yang sudah kita ambil (contoh saat ini adalah element **id** pada data API (userid, **id**, title, body) yang akan kita gunakan untuk UNIQUE KEY. Lihat Gambar 1.20.

```

20  render() {
21    return(
22      <div className="post-artikel">
23        <h2>Daftar Artikel</h2>
24        {
25          this.state.listArtikel.map(artikel => { // looping dan masuk
26            return <Post key={artikel.id} judul={artikel.title} isi=
27          })
28        }
29      </div>
30    )
31  }

```

Gambar 1.20. Penambahan key pada stateless component

29. Simpan dan lihat apa yang terjadi pada *console* browser (Gambar 1.21).



Gambar 1.21. Hasil akhir

1.4 Pertanyaan Praktikum 1

- a. Pada langkah 8, sekarang coba kalian ganti class `container` dengan `container-fluid` atau sebaliknya pada file "`public/index.html`" dan lihat apa perbedaannya.
 1. Tampilan seperti apa yang kalian temukan setelah mencoba mengganti nama class tersebut?
 2. Apa perbedaan dari `container` dan `container-fluid` ?
- b. Jika kita ingin meng-*import* suatu *component* contoh *component bootstrap*, akan tetapi *component* dalam tersebut belum terdapat pada module ReactJS. Apa yang akan dilakukan untuk dapat menggunakan component tersebut? Bagaimana caranya?

Praktikum 2

Interaksi dengan API menggunakan *Fake API*

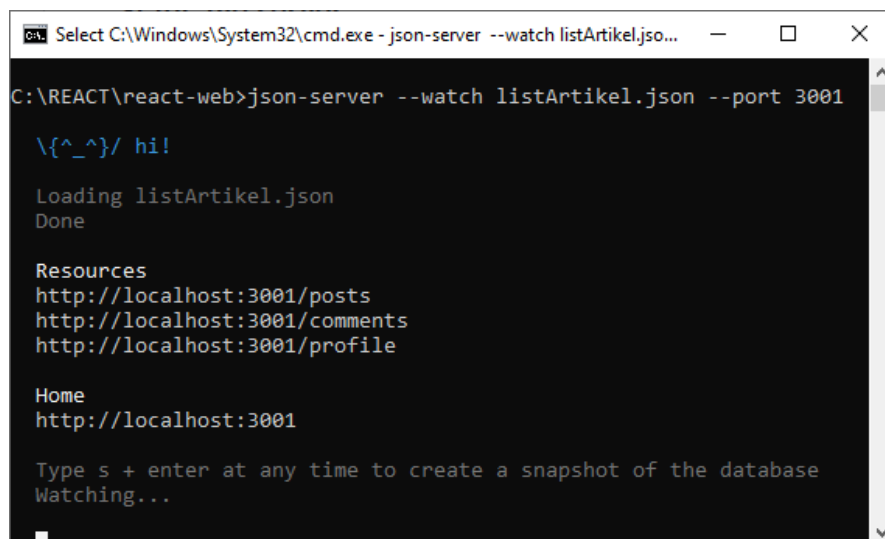
Saat kita mengakses API dengan method GET seperti Praktikum 1. Kita langsung menembak API dari server *jsonplaceholder* yaitu <https://jsonplaceholder.typicode.com/posts>. Data yang akan kita dapat sesuai dengan data yang disediakan oleh server tersebut.

Bagaimana jika kita ingin mendapatkan atau mengolah data API sendiri sehingga data yang akan kita pakai sesuai dengan yang kita inginkan? Solusinya bisa menggunakan *Fake API* yang kita install di local project ReactJS.

2.1 Install Fake API (JSON Server)

Fake API/JSON Server bisa kita dapatkan di halaman <https://github.com/typicode/json-server>. Tahapan install dan membuat data json sendiri

1. Install pada direktori project reactjs kita dengan perintah npm `install -g json-server`
2. *Copy*-kan file json `listArtikel.json` yang sudah ada pada direktori project reactjs kita.
3. Buka cmd baru pada direktori project, lalu ketik perintah `json-server --watch listArtikel.json --port 3001`.
4. Apabila pada cmd tampil seperti Gambar 2.1, maka server *Fake API* local kita telah siap.



```

C:\REACT\react-web>json-server --watch listArtikel.json --port 3001

\{^_^}/ hi!

Loading listArtikel.json
Done

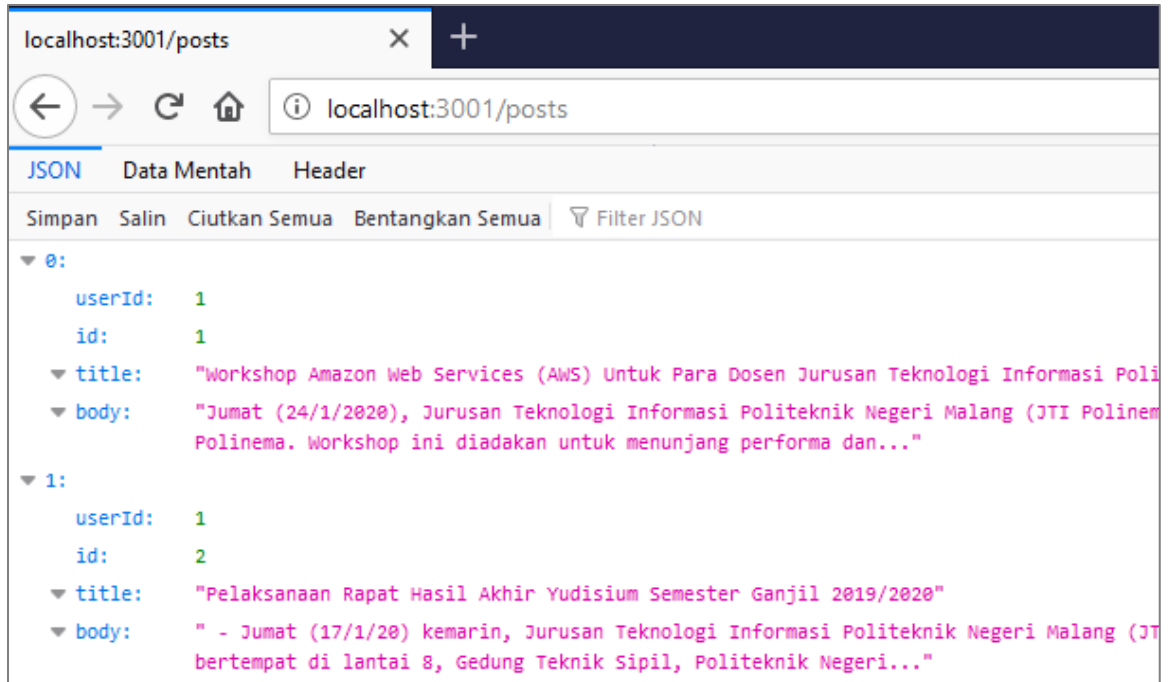
Resources
http://localhost:3001/posts
http://localhost:3001/comments
http://localhost:3001/profile

Home
http://localhost:3001

Type s + enter at any time to create a snapshot of the database
Watching...
```

Gambar 2.1 Tampilan response json-server

5. Kita cek *url resource* yang ada pada Fake API server ke browser apakah bisa diakses. Ketik url <http://localhost:3001/posts> pada browser.



Gambar2.2. Tampilan response json-server

6. Untuk memastikan lagi, kita edit *statefull component* **BlogPost** (Gambar 1.18) pada baris 11. Kita ganti url API dari <https://jsonplaceholder.typicode.com/posts> menjadi <http://localhost:3001/posts>
7. Simpan perubahan dan amati apa yang terjadi.

2.2 Pertanyaan Praktikum 2

- a. Kenapa *json-server* dijalankan pada port 3001? Kenapa tidak sama-sama dijalankan pada *port* 3000 seperti project react yang sudah kita buat?
- b. Bagaimana jadinya kalua kita ganti *port json-server* menjadi 3000?

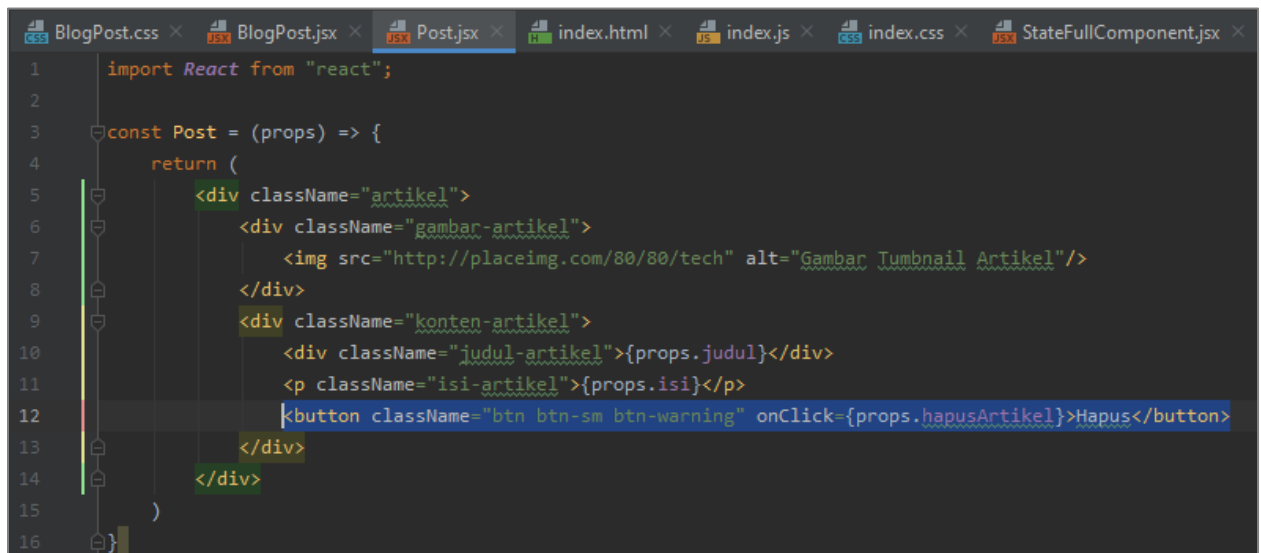
Praktikum 3

Interaksi dengan API menggunakan method DELETE

Method DELETE secara umum digunakan untuk melakukan proses hapus data. Saat kita ingin menghapus data, kita akan melakukan *request* ke server API dengan menggunakan *method* DELETE. Secara otomatis, server akan mengetahui bahwa *request* yang kita lakukan adalah untuk melakukan penghapusan data karena *request* kita menggunakan *method* DELETE.

3.1 Langkah Praktikum 3

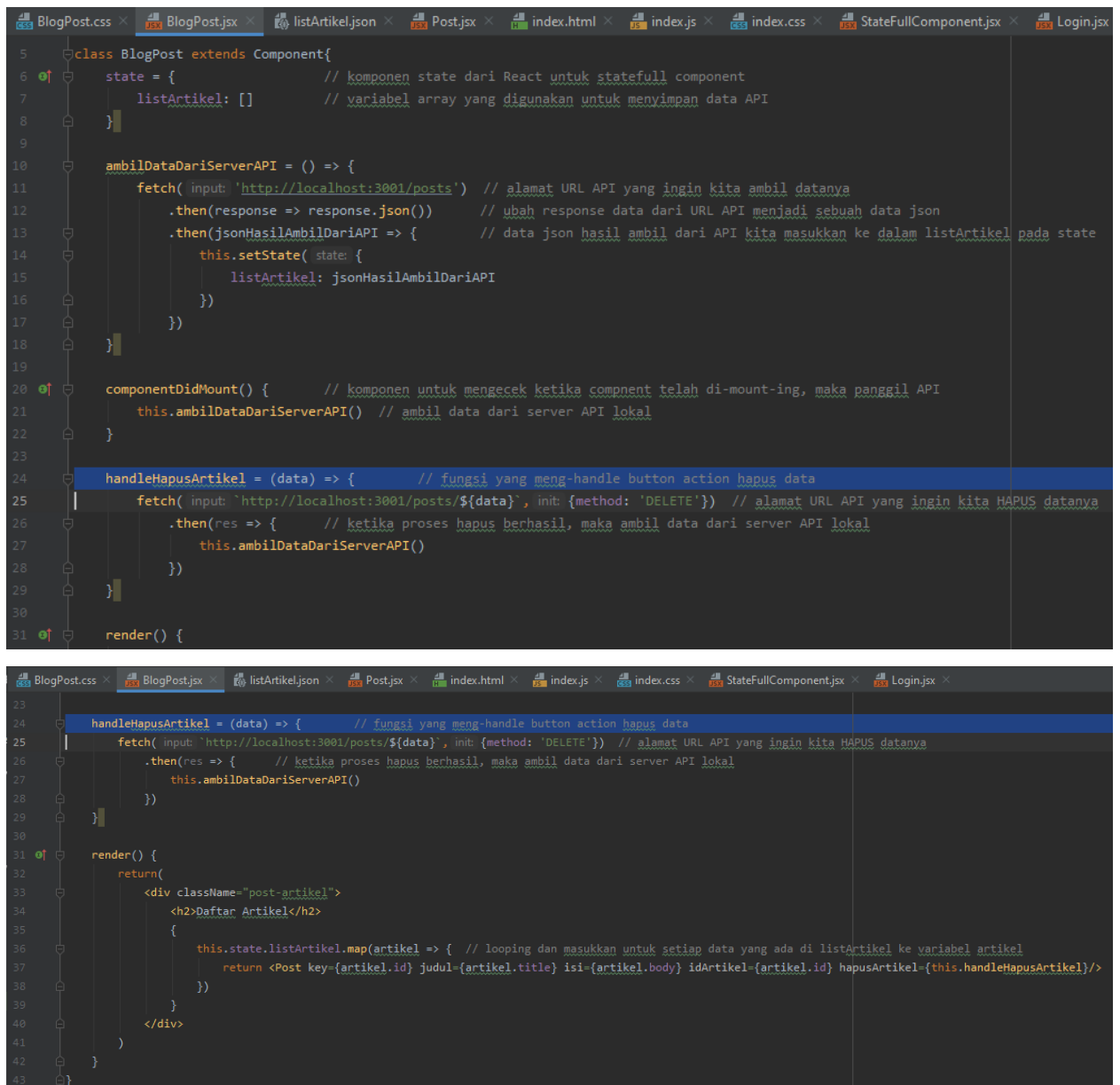
1. Buka *stateless component* **Post**. Tambahkan 1 baris kode program pada baris 10 seperti pada Gambar 3.1



```
1 import React from "react";
2
3 const Post = (props) => {
4   return (
5     <div className="artikel">
6       <div className="gambar-artikel">
7         
8       </div>
9       <div className="konten-artikel">
10        <div className="judul-artikel">{props.judul}</div>
11        <p className="isi-artikel">{props.isi}</p>
12        <button className="btn btn-sm btn-warning" onClick={props.hapusArtikel}>Hapus</button>
13      </div>
14    </div>
15  )
16 }
```

Gambar 3.1 Tambah kode program Post.jsx

2. Kemudian pada *statefull component* **BlogPost**, modifikasi kode program sebelumnya sesuai dengan Gambar 3.2



Gambar 3.2 Modifikasi kode program BlogPost

3. Klik tombol hapus pada list artikel di browser. Amati apa yang terjadi.

3.2 Pertanyaan Praktikum 3

- a. Apa yang terjadi setelah kalian klik tombol hapus?
- b. Perhatikan file `listArtikel.json`, apa yang terjadi pada file tersebut? Kenapa demikian?
- c. Fungsi `handleHapusArtikel` itu untuk apa?
- d. Jelaskan perbedaan fungsi `componentDidMount()` pada Gambar 1.18 dengan fungsi `componentDidMount()` pada Gambar 3.2 ?

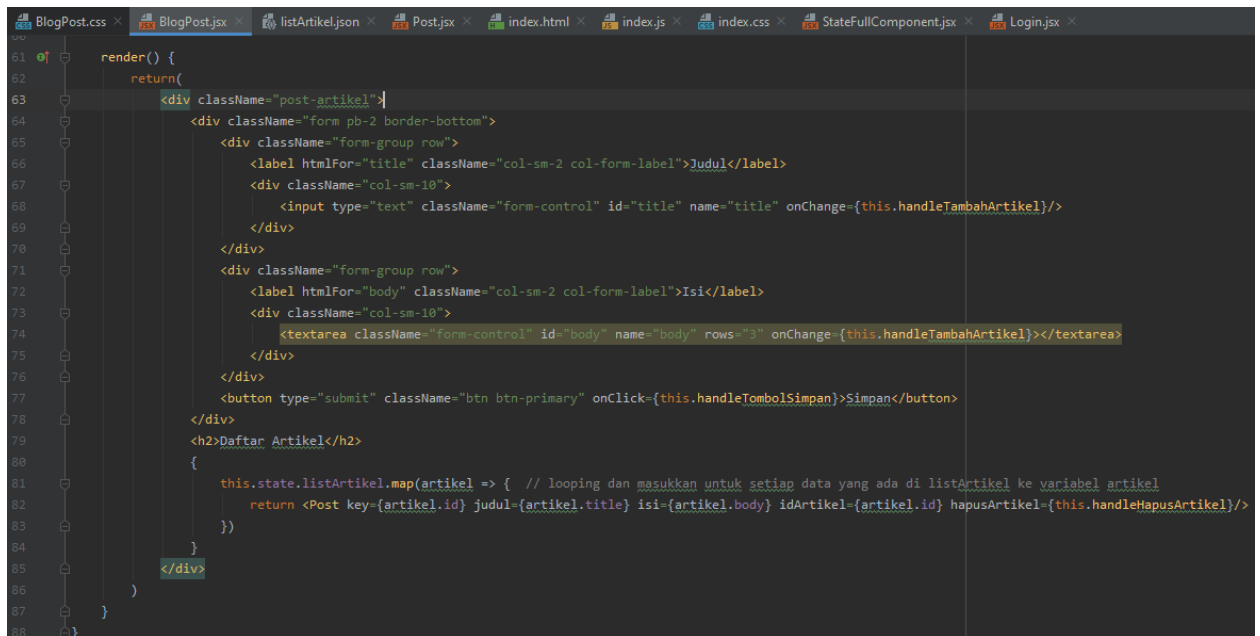
Praktikum 4

Interaksi dengan API menggunakan method POST

Method POST sering digunakan dalam mengirimkan form *request* ke server. Dalam API method POST biasa digunakan untuk melakukan insert/tambah data pada server.

4.1 Langkah Praktikum 4

1. Buka *statefull component* **BlogPost**, dan modifikasi pada fungsi **render()** untuk menampilkan *form input* artikel yang berisi judul dan isi berita. seperti pada Gambar 4.1



```
61 render() {
62   return(
63     <div className="post-artikel">
64       <div className="form pb-2 border-bottom">
65         <div className="form-group row">
66           <label htmlFor="title" className="col-sm-2 col-form-label">Judul</label>
67           <div className="col-sm-10">
68             <input type="text" className="form-control" id="title" name="title" onChange={this.handleTambahArtikel}/>
69           </div>
70         </div>
71         <div className="form-group row">
72           <label htmlFor="body" className="col-sm-2 col-form-label">Isi</label>
73           <div className="col-sm-10">
74             <textarea className="form-control" id="body" name="body" rows="3" onChange={this.handleTambahArtikel}></textarea>
75           </div>
76         </div>
77         <button type="submit" className="btn btn-primary" onClick={this.handleTambahArtikel}>Simpan</button>
78       </div>
79       <h2>Daftar Artikel</h2>
80       {
81         this.state.listArtikel.map(artikel => { // looping dan masukkan untuk setiap data yang ada di listArtikel ke variabel artikel
82           return <Post key={artikel.id} judul={artikel.title} isi={artikel.body} idArtikel={artikel.id} hapusArtikel={this.handleHapusArtikel}/>
83         })
84       }
85     </div>
86   )
87 }
88 }
```

Gambar 4.1 modifikasi component BlogPost

2. Kemudian modifikasi **BlogPost** untuk bagian **state** dan *request* API dari server, seperti Gambar 4.2

```

1 import React, {Component} from "react";
2 import './BlogPost.css';
3 import Post from "../../component/BlogPost/Post";
4
5 class BlogPost extends Component{
6   state = { // komponen state dari React untuk statefull component
7     listArtikel: [], // variabel array yang digunakan untuk menyimpan data API
8     insertArtikel: { // variabel yang digunakan untuk menampung sementara data yang akan di insert
9       userId: 1, // kolom userId, id, title, dan body sama, mengikuti kolom yang ada pada listArtikel.json
10      id: 1,
11      title: "",
12      body: ""
13    }
14  }
15
16  ambilDataDariServerAPI = () => { // fungsi untuk mengambil data dari API dengan penambahan sort dan order
17    fetch( input: 'http://localhost:3001/posts? sort=id& order=desc') // penambahan sort dan order berdasarkan parameter
18      .then(response => response.json()) // ubah response data dari URL API menjadi sebuah data json
19      .then(jsonHasilAmbilDariAPI => { // data json hasil ambil dari API kita masukkan ke dalam listArtikel pada state
20        this.setState( state: {
21          listArtikel: jsonHasilAmbilDariAPI
22        })
23      })
24  }
25
26  componentDidMount() { // komponen untuk mengecek ketika component telah di-mount-ing, maka panggil API
27    this.ambilDataDariServerAPI() // ambil data dari server API lokal
28  }

```

Gambar 4.2 penambahan state pada BlogPost

3. Tambahkan untuk *handle* form tambah data artikel seperti Gambar 4.3

```

26 componentDidMount() { // komponen untuk mengecek ketika component telah di-mount-ing, maka panggil API
27   this.ambilDataDariServerAPI() // ambil data dari server API lokal
28 }
29
30 handleHapusArtikel = (data) => { // fungsi yang meng-handle button action hapus data
31   fetch( input: 'http://localhost:3001/posts/${data}', init: {method: 'DELETE'}) // alamat URL API yang ingin kita HAPUS datanya
32     .then(res => { // ketika proses hapus berhasil, maka ambil data dari server API lokal
33       this.ambilDataDariServerAPI()
34     })
35 }
36
37 handleTambahArtikel = (event) => { // fungsi untuk meng-handle form tambah data artikel
38   let formInsertArtikel = {...this.state.insertArtikel}; // cloning data state insertArtikel ke dalam variabel formInsertArtikel
39   let timestamp = new Date().getTime(); // digunakan untuk menyimpan waktu (sebagai ID artikel)
40   formInsertArtikel['id'] = timestamp;
41   formInsertArtikel[event.target.name] = event.target.value; // menyimpan data onchange ke formInsertArtikel sesuai dengan target yg diisi
42   this.setState( state: {
43     insertArtikel: formInsertArtikel
44   });
45 }
46
47 handleTombolSimpan = () => { // fungsi untuk meng-handle tombol simpan
48   fetch( input: 'http://localhost:3001/posts', init: {
49     method: 'post', // method POST untuk input/insert data
50     headers: {
51       'Accept': 'application/json',
52       'Content-Type': 'application/json'
53     },

```

Gambar 4.3 Handle tambah artikel

4. Langkah terakhir tambahkan fungsi untuk handle tombol simpan artikel, seperti pada Gambar 4.4



```
46
47 handleTombolSimpan = () => { // fungsi untuk meng-handle tombol simpan
48   fetch( input: 'http://localhost:3001/posts', init: {
49     method: 'post', // method POST untuk input/insert data
50     headers: {
51       'Accept': 'application/json',
52       'Content-Type': 'application/json'
53     },
54     body: JSON.stringify(this.state.insertArtikel) // kirimkan ke body request untuk data artikel yang akan ditambahkan (insert)
55   })
56   .then( (response : Response ) => {
57     this.ambilDataDariServerAPI(); // reload / refresh data
58   });
59 }
60
61 render() {
62   return(
```

Gambar 4.4 Handle tombol simpan

5. Simpan, lakukan percobaan penambahan data, dan amati perubahannya.

4.2 Pertanyaan Praktikum 4

- a. Jelaskan apa yang terjadi pada file `listArtikel.json` sebelum dan setelah melakukan penambahan data?
- b. Data yang ditampilkan di browser adalah data terbaru berada di posisi atas dan data lama berada di bawah, sedangkan pada file `listArtikel.json` data terbaru malah berada di bawah. Jelaskan mengapa demikian?

TUGAS PRAKTIKUM

Buatlah program menggunakan Fake API (JSON Server) tentang pendataan Mahasiswa aktif/cuti/lulus di Jurusan Teknologi Informasi. Atribut-atribut yang ada dari mahasiswa adalah NIM, nama, alamat, no hp, tahun Angkatan, dan status. Buatlah aplikasi yang menggunakan API dengan method GET, DELETE, dan POST.

Contoh data json yang digunakan.

```
{
  "mahasiswa": [
    {
      "NIM": 180823453,
      "nama": "Mahasiswa 1",
      "alamat": "Jl. Soekarno Hatta No. 9 Malang",
      "hp": "081802023249",
      "angkatan": 2018,
      "status": "aktif"
    },
    {
      "NIM": 140823453,
      "nama": "Mahasiswa X",
      "alamat": "Jl. Menur No. 9 Surabaya",
      "hp": "081802023523",
      "angkatan": 2014,
      "status": "lulus"
    }
  ]
}
```

~ ~ ~ Selamat Mengerjakan ~ ~ ~