M Reza Faisal
Ilmu Komputer, FMIPA, ULM

Component & Design

Instruksi

1. File Utama Aplikasi

Ukuran file sebuah project React Native adalah relatif besar karena didalamnya berisi banyak node module dan juga hasil konversi kode React Native menjadi kode native Android dan iOS. Sebagai contoh ukuran folder `node_modules` adalah sekitar 200MB. Sedangkan ukuran folder `android` 276MB dan ukuran folder `ios` adalah 34MB. Sehingga ini menjadi pertimbangan dalam membuat project baru.

Karena hal itu kita cukup menggunakan sebuah project saja untuk latihan-latihan yang dilakukan. Pada hands on labs sebelumnya kode latihan dilakukan dengan mengubah kode file `App.js`. Pada hands on labs ini akan dibuat file seperti `App.js` untuk menulis kode latihan yang baru.

Pada hands on labs sebelumnya telah dijelaskan fungsi file `index.js` yang berisi cara untuk menentukan file utama aplikasi yang akan digunakan. Pada Kode 1 adalah isi dari file `index.js`.

Kode 1. index.js.

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import App from './App';
7  import {name as appName} from './app.json';
8
9  AppRegistry.registerComponent(appName, () => App);
```

Pada baris ke-6 adalah cara untuk mengimport file `App.js` yang dapat diakses dengan nama App. Kemudian pada baris ke-9 dapat dilihat class `AppRegistry` yang merupakan titik masuk untuk menjalankan aplikasi React Native. Dengan method `registerComponent` untuk meregister komponen utama.

Dengan mengetahui informasi di atas maka kita dapat membuat file aplikasi utama baru kemudian didaftarkan agar file tersebut dijalankan sebagai aplikasi. Sebagai contoh kita buat file aplikasi utama baru dengan nama `AppButton.js` seperti pada Kode 2.

Kode 2. AppButton.js

AppButton.js

```
1  import React from 'react';
2  import { Text, View, Image, Button } from 'react-native';
3
4  const AppButton = () => {
5    return (
6      <View>
7        <Text>App Button</Text>
8        <Button title="Click Me!"/>
```

```
9       </View>
10    )
11  }
12
13  export default AppButton;
```

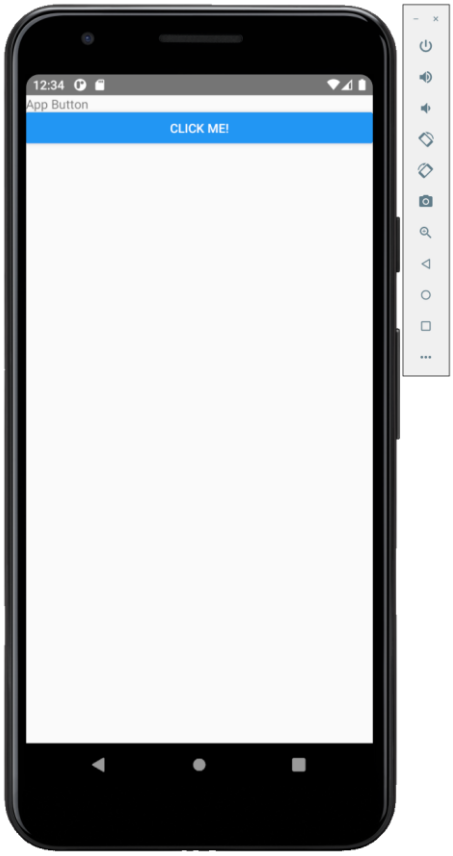
Selanjutnya adalah mengedit file `index.js` agar file `AppButton.js` ini dijadikan sebagai file aplikasi utama. Pada Kode 3 dapat dilihat hasil file `index.js` yang dimodifikasi untuk maksud di atas.

Kode 3. index.js untuk AppButton.js

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import AppButton from './AppButton';
7  import {name as appName} from './app.json';
8
9  AppRegistry.registerComponent(appName, () => AppButton);
```

Hasilnya dapat dilihat pada gambar sebelah kiri di Gambar 1.



Gambar 1. Hasil AppButton.

Contoh berikutnya dibuat file aplikasi utama dengan nama `AppTextInput.js` seperti pada Kode 4.

Kode 4. AppTextInput.js

AppTextInput.js

```
1  import React from 'react';
2  import { Text, View, Image, TextInput, Button, StyleSheet } from 'react-
3  native';
4
5  const styles = StyleSheet.create({
6      input: {
7          height: 40,
8          margin: 12,
9          borderWidth: 1,
10         padding: 10,
11     },
12 });
13
14 const AppTextInput = () => {
15   return (
16     <View>
17       <Text>Name</Text>
18       <TextInput placeholder='Write your name' style={styles.input}/>
19       <Text>Age</Text>
20       <TextInput placeholder='Write your age' keyboardType='numeric' style
21 ={styles.input}/>
22       <Text>Email</Text>
23       <TextInput placeholder='Write your email' keyboardType='email-
24 address' style={styles.input}/>
25       <Button title="Save"/>
26     </View>
27   )
28 }
29
30 export default AppTextInput;
```

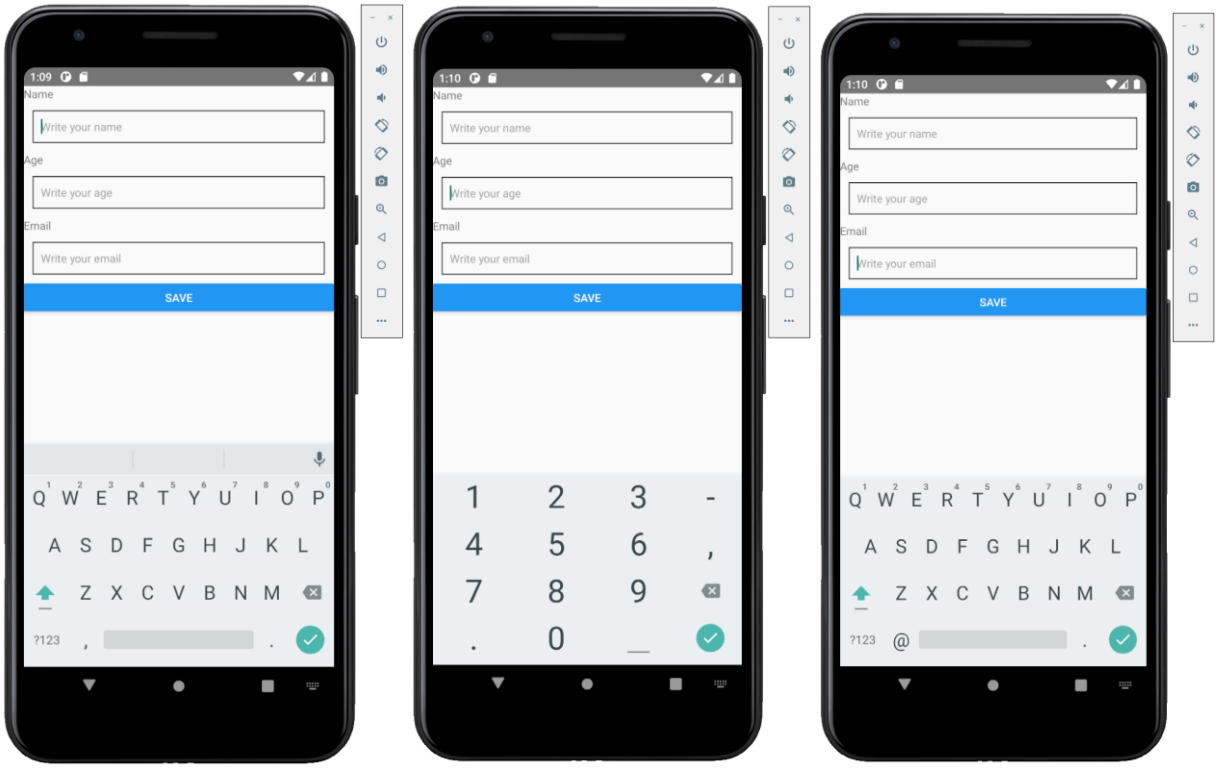
Kemudian agar file `AppTextInput.js` ini menjadi file utama aplikasi yang ditampilkan maka dilakukan modifikasi file `index.js` seperti pada Kode 5. Hasilnya dapat dilihat pada Gambar 2.

Kode 5. index.js untuk AppTextInput.js.

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import AppTextInput from './AppTextInput';
7  import {name as appName} from './app.json';
```

```
8
9   AppRegistry.registerComponent(appName, () => AppTextInput);
```



Gambar 2. Hasil AppTextInput

Pada Gambar 2 dapat dilihat atribut-atribut dari component `TextInput` seperti `keyboardType` yang memberi perubahan terhadap format keyboard saat kolom input dipilih.

2. Event Handlers

Event handlers atau event listeners adalah fungsi yang menanggapi aksi dari pengguna. Sebagai contoh ketika tombol ditekan oleh user maka biasanya ada fungsi yang dijalankan sebagai contoh untuk alert atau mengirim dan menyimpan data. Atau ketika user mengisi input pada component `TextInput`, maka dimungkinkan komponen tersebut menjalankan fungsi yang dijalankan sebagai contoh memeriksa apakah isinya adalah angka atau bukan.

Untuk menambahkan event handler pada component umumnya dengan memberikan atribut pada komponen. Nama atribut event handler pada setiap komponen dapat berbeda-benda. Sebagai contoh adalah sebagai berikut:

- Komponen `Button` memiliki atribut `onPress` sebagai event handler ketika tombol ditekan.
- Komponen `TextInput` memiliki atribut `onChangeText` sebagai event handler ketika nilai pada input berubah.

Seperti yang telah kita ketahui bahwa setiap atribut pada komponen memiliki nilai, nilai untuk atribut yang berfungsi sebagai event handler adalah nama fungsi yang akan dipanggil.

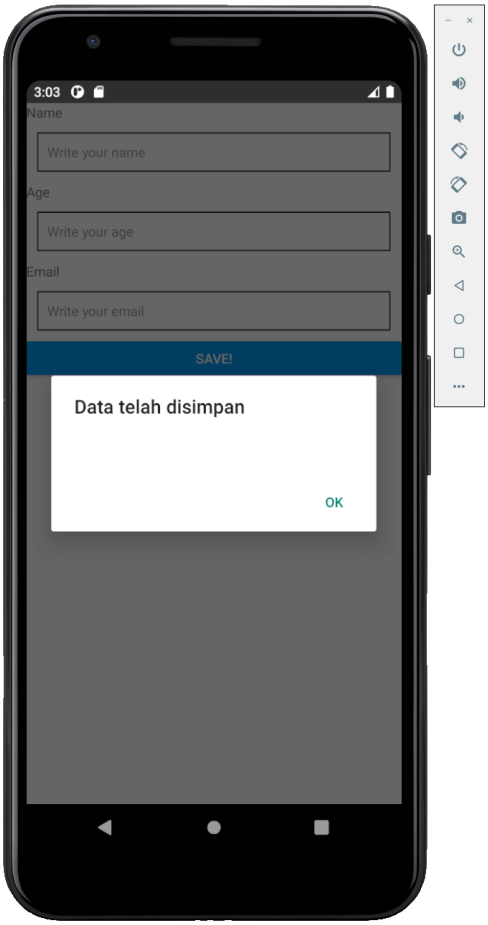
Sebagai contoh adalah event handler pada komponen `Button` dapat dilihat pada Kode 6.

Kode 6. Event handler onPress pada komponen Button.

AppTextInput.js

```
1  import React from 'react';
2  import { Text, View, Image, TextInput, Button, StyleSheet, Alert } from 'r
3  eact-native';
4
5  const styles = StyleSheet.create({
6      input: {
7          height: 40,
8          margin: 12,
9          borderWidth: 1,
10         padding: 10,
11     },
12 });
13
14 const AppTextInput = () => {
15   return (
16     <View>
17       <Text>Name</Text>
18       <TextInput placeholder='Write your name' style={styles.input}/>
19       <Text>Age</Text>
20       <TextInput placeholder='Write your age' keyboardType='numeric' style
21 ={styles.input}/>
22       <Text>Email</Text>
23       <TextInput placeholder='Write your email' keyboardType='email-
24 address' style={styles.input}/>
25       <Button title="Save!" onPress={() => Alert.alert('Data telah disimpa
26 n')}/>
27     </View>
28   )
29 }
30
31 export default AppTextInput;
```

Pada baris ke-25 dapat dilihat pada komponen `Button` terdapat atribut `onPress`. Kemudian dapat dilihat nilai dari atribut ini adalah kode JS untuk menampilkan alert. Untuk menampilkan alert digunakan method `alert` dari class `Alert`. Agar class `Alert` dapat digunakan maka perlu dilakukan import seperti yang terlihat pada baris ke-2. Hasilnya dapat dilihat pada Gambar 3.



Gambar 3. Hasil event handler onPress.

Dari kode di atas dapat dilihat kode untuk menangani event ditulis bukan dalam bentuk nama fungsi, namun berupa perintah lengkap. Berikut ini adalah contoh lain yang menggunakan fungsi dengan menggunakan kata kunci `function`. Contoh kodenya dapat dilihat pada `AppTextInput.js` dengan function `_onPressButton`.

Kode 7. AppTextInput.js dengan function _onPressButton.

AppTextInput.js

```
1  import React from 'react';
2  import { Text, View, Image, TextInput, Button, StyleSheet, Alert } from
3  'react-native';
4
5  const styles = StyleSheet.create({
6      input: {
7          height: 40,
8          margin: 12,
9          borderWidth: 1,
10         padding: 10,
11     },
12 });
13
```

```
14  function _onPressButton() {
15      Alert.alert('Data telah disimpan');
16  }
17
18  const AppTextInput = () => {
19      return (
20      <View>
21          <Text>Name</Text>
22          <TextInput placeholder='Write your name' style={styles.input}/>
23          <Text>Age</Text>
24          <TextInput placeholder='Write your age' keyboardType='numeric'
25  style={styles.input}/>
26          <Text>Email</Text>
27          <TextInput placeholder='Write your email' keyboardType='email-
28  address' style={styles.input}/>
29          <Button title="Save!" onPress={_onPressButton}/>
30      </View>
31      )
32  }
33
34  export default AppTextInput;
```

Fungsi `_ onPressButton` dapat dilihat pada baris ke-14 sampai dengan ke-16. Menulis fungsi seperti ini merupakan cara sederhana yang umum digunakan pada JavaScript.

Contoh berikutnya adalah dengan membuat class yang didalamnya terdapat method static yang nanti dapat diakses seperti pada contoh Kode 6. Pada Kode 8 adalah contoh penggunaan class yang didalamnya terdapat method static. Sebagai informasi, pada pemrograman berbasis obyek method static dapat digunakan tanpa harus membuat obyek dari class yang memiliki method tersebut.

Kode 8. AppTextInput.js dengan class ComponentAction.

AppTextInput.js

```
1   import React from 'react';
2   import { Text, View, Image, TextInput, Button, StyleSheet, Alert } from
3   'react-native';
4
5   const styles = StyleSheet.create({
6       input: {
7           height: 40,
8           margin: 12,
9           borderWidth: 1,
10          padding: 10,
11      },
12  });
13
14  class ComponentAction {
15      static onPressButton(){
16          Alert.alert('Data telah disimpan.');
17      }
18  }
19
20  const AppTextInput = () => {
```

```
21      return (
22      <View>
23          <Text>Name</Text>
24          <TextInput placeholder='Write your name' style={styles.input}/>
25          <Text>Age</Text>
26          <TextInput placeholder='Write your age' keyboardType='numeric'
27  style={styles.input}/>
28          <Text>Email</Text>
29          <TextInput placeholder='Write your email' keyboardType='email-
30  address' style={styles.input}/>
31          <Button title="Save!" onPress={() =>
32  ComponentAction.onPressButton()}/>
33      </View>
34      )
35  }
36
37  export default AppTextInput;
```

Penulisan class pada Javascript ES6 dapat dilihat pada baris ke-14 sampai dengan baris ke-18. Kemudian pada baris ke-15 dapat dilihat bagaimana method static dibuat. Selanjutnya untuk menggunakan method static tersebut dapat dilihat pada baris ke-32.

Penjelasan lebih jauh tentang penulisan dan penggunakan bahasa pemrograman JavaScript ES6 akan diberikan pada materi hands on labs berikutnya.

3. props

Sebagian komponen dapat dikustomisasi saat dibuat dengan menggunakan parameter yang berbeda. parameter pembuatan ini dikenal dengan istilah `props`. `props` dapat berfungsi sebagai variable yang digunakan untuk mengirimkan nilai dari komponen induk ke komponen anak. `props` bersifat read-only dan seharusnya tidak dapat dimodifikasi.

Custom component yang kita buat juga dapat menggunakan `props`. Dengan adanya `props` maka sebuah komponen dapat digunakan pada banyak tempat di dalam aplikasi atau project. Untuk mengakses variable `props` dapat dilakukan dengan dua cara yaitu:

- `props.NAMA_PROP`, jika diakses dari komponen pada JSX.
- `this.props.NAMA_PROP`, jika diakses dari class komponen yang sama.

Untuk melihat penggunaan `props` maka terlebih dahulu bisa dibuat custom componen sederhana dengan menggunakan `AppCustomComponents.js` yang pernah dibuat pada hands on labs sebelumnya. Pada kode ini ditambahkan custom component baru dengan `props`.

Kode 9. AppCustomComponent.js

AppCustomComponents.js

```
1   import React from 'react';
2   import { Text, View, Image } from 'react-native';
3
4   export const AppHeader = () => {
5       return (
```

```
6           <View style={{height:60, backgroundColor: 'gray'}}>
7               <Text style={{fontWeight: 'bold',fontSize: 32}}>App
8   Title</Text>
9           </View>
10      );
11  }
12
13  export const AppContent = () => {
14      return (
15          <View>
16              <Text style={{fontWeight: 'bold',fontSize: 23}}>App
17  Content</Text>
18          <Image source={require('./images/flutter-logo.png')}
19  style={{width: 300, height:50, resizeMode:'contain'}} />
20          </View>
21      );
22  }
23
24  export const AppFooter = (props) => {
25      return (
26          <View style={{height:40, backgroundColor: 'gray'}}>
27              <Text style={{fontWeight: 'bold',fontSize: 23,
28  color:'white'}}>{props.footerContent}</Text>
29          </View>
30      );
31  }
```

Custom component baru yang ditambahkan dapat dilihat pada baris ke-24 sampai dengan ke-31. Selanjutnya adalah membuat file aplikasi utama yang digunakan untuk memanfaatkan custom component yang dibuat tersebut. Untuk kasus ini nama file aplikasi utama adalah `AppProps.js` dengan isi seperti terlihat pada Kode 10.

Kode 10. AppProps.js

AppProps.js

```
1   import React from 'react';
2   import { Text, View, Image } from 'react-native';
3   import {AppHeader, AppContent, AppFooter} from './AppCustomComponents'
4
5   const AppProps = () => {
6     return (
7       <View>
8         <AppHeader/>
9         <AppContent/>
10        <AppFooter footerContent="footer content"/>
11      </View>
12    )
13  }
14
15  export default AppProps;
```

Contoh penggunaan props pada custom component dapat dilihat pada baris ke-10. Nama antribut `footerContent` pada component pada Kode 10 sesuai dengan nilai `props.footerContent` pada Kode 9 baris ke-28.

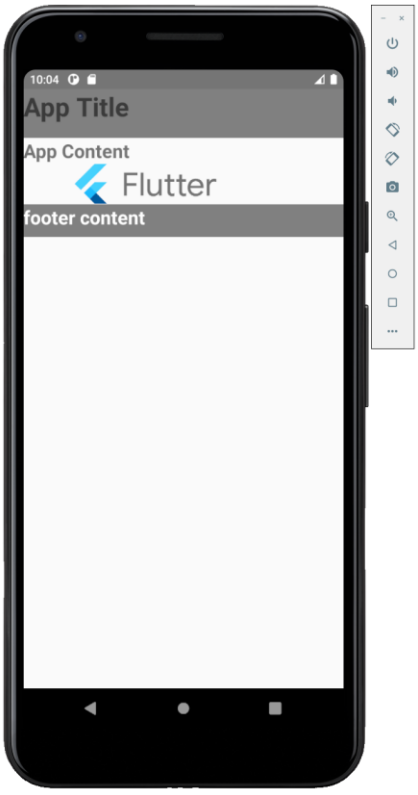
Agar file `AppProps.js` menjadi file utama aplikasi maka file `index.js` perlu dimodifikasi menjadi seperti pada Kode 11.

Kode 11. index.js untuk AppProps.js

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import AppProps from './AppProps';
7  import {name as appName} from './app.json';
8
9  AppRegistry.registerComponent(appName, () => AppProps);
```

Hasilnya dapat dilihat pada Gambar 4.



Gambar 4. AppProps.js.

4. state

Berbeda dengan `props`, `state` mengijinkan komponen untuk mengubah nilainya berdasarkan respon dari aksi user, respon network dan lain-lain. Jika `props` digunakan dengan cara mengirimkannya sebagai parameter maka `state` tidak dikirimkan sebagai parameter namun komponen akan melakukan inisialisasi dan memodifikasi nilainya secara internal.

Pada Kode 12 adalah contoh implementasi `state` pada halaman aplikasi. Untuk menggunakan `state` pada halaman aplikasi maka terlebih dahulu harus dideklarsikan kata kunci `useState` seperti yang terlihat pada baris ke-1.

Kode 12. AppState.js

AppState.js

```
1   import React, { useState } from 'react';
2   import { View, Text, Button, StyleSheet } from 'react-native';
3
4   const AppState = () => {
5     const [count, setCount] = useState(0);
6
7     return (
8       <View style={styles.container}>
9         <Text>You clicked {count} times</Text>
10        <Button
11          onPress={() => setCount(count + 1)}
12          title="Click me!"
13        />
14      </View>
15    );
16  };
17
18  export default AppState;
19
20  // React Native Styles
21  const styles = StyleSheet.create({
22    container: {
23      flex: 1,
24      justifyContent: 'center',
25      alignItems: 'center'
26    }
27  });
```

Kode class `AppState` dapat dilihat dimulai dari baris ke-4 sampai baris ke-16. Di dalam class `AppState` perlu dilakukan inisialisasi variable yang akan menyimpan `state` dan melakukan inisialisasi, selain itu juga ditentukan nama method yang digunakan untuk mengupdate nilai pada variable tersebut, kode untuk maksud ini dapat dilihat pada baris ke-5.

Kemudian pada baris ke-11 dapat dilihat bagaimana method `setCount` digunakan untuk mengupdate nilai variable `count` yang menyimpan nilai `state`. Sedangkan pada baris ke-9 dapat dilihat bagaimana menampilkan nilai variable count ke layar.

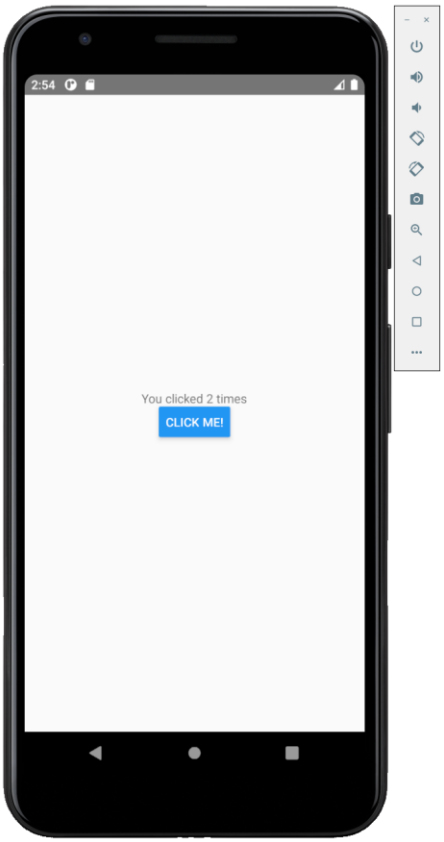
Untuk membuat file `AppState.js` menjadi file aplikasi utama maka seperti yang dilakukan pada pembahasan sebelumnya perlu dilakukan perubahan pada file `index.js` seperti pada Kode 13.

Kode 13. index.js untuk AppState.

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import AppState from './AppState';
7  import {name as appName} from './app.json';
8
9  AppRegistry.registerComponent(appName, () => AppState);
```

Hasilnya dapat dilihat pada Gambar 5, pada aplikasi ini jika user mengklik tombol CLICK ME! maka nilai angka pada teks di atas tombol tersebut akan bertambah.



Gambar 5. Hasil AppState.

5. Design

Pada hands on labs sebelumnya sudah dijelaskan penggunaan style dengan menggunakan class `StyleSheet`. Dengan menggunakan style maka komponen-komponen dapat dipercantik. Selain itu style

juga dapat digunakan untuk mengatur antarmuka atau layout. Pada sub bab ini diberikan beberapa hal-hal dasar yang terkait dengan tujuan tersebut.

Pada contoh-contoh kode sebelumnya telah digunakan komponen `View` yang fungsinya sebagai container. Bagi pembaca yang telah terbiasa dengan pembuatan aplikasi web, komponen `View` ini mempunyai fungsi seperi `<div>`. Bentuk komponen ini adalah kotak yang mempunyai lebar dan tinggi. Umumnya komponen ini digunakan sebagai wadah untuk menampung komponen-komponen lainnya. Komponen ini juga menjadi komponen yang dapat digunakan untuk antarmuka atau layout.

Fixed Dimension

Cara umum yang digunakan untuk menentukan dimensi adalah dengan memberikan nilai tetap untuk lebar dan tinggi. Nilai atribut komponen pada React Native tidak memiliki unit artinya cukup diberikan nilai angka saja. Untuk memberikan nilai dimensi yang tetap dapat dilihat pada contoh seperti pada Kode 14.

Kode 14. AppFixedDimensions.js

AppFixedDimensions.js

```
1  import React from 'react';
2  import { View } from 'react-native';
3
4  const AppFixedDimensions = () => {
5    return (
6      <View>
7        <View style={{width: 100, height: 100, backgroundColor: 'powderblue'}} />
8        <View style={{width: 200, height: 200, backgroundColor: 'skyblue'}} />
9        <View style={{width: 300, height: 300, backgroundColor: 'steelblue'}} />
10     </View>
11   );
12 };
13
14 export default AppFixedDimensions;
```

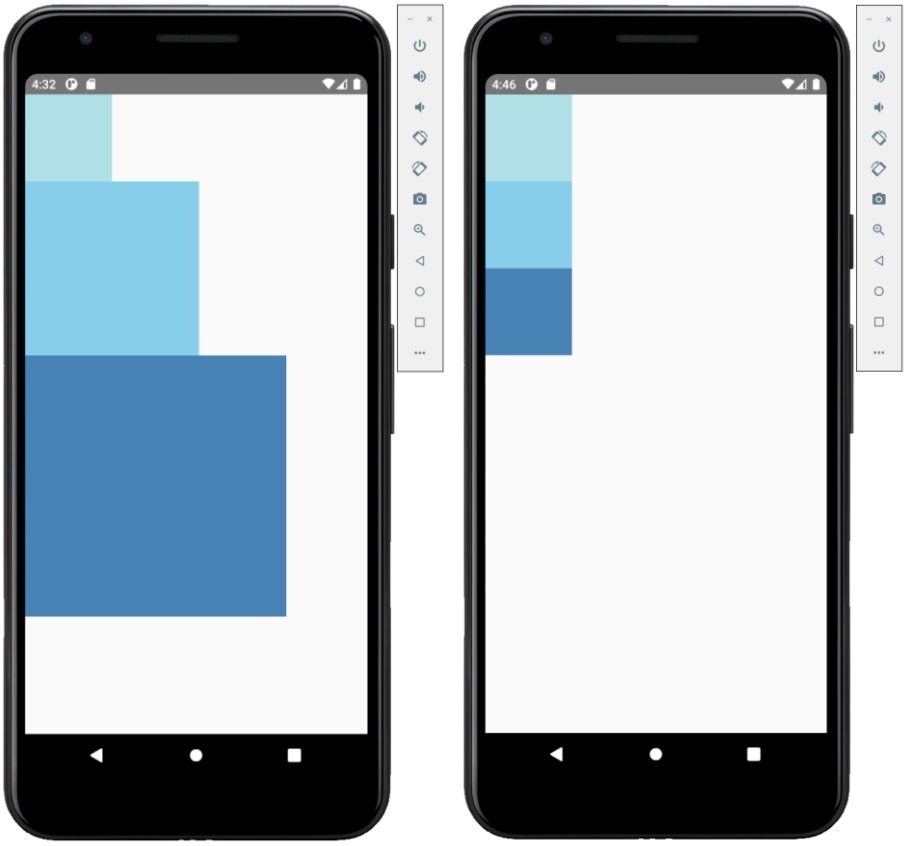
Kemudian modifikasi file `index.js` menjadi seperti pada Kode 15.

Kode 15. index.js untuk AppFixedDimensions.js.

index.js

```
1  /**
2   * @format
3   */
4
5  import {AppRegistry} from 'react-native';
6  import AppFixedDimensions from './AppFixedDimensions';
7  import {name as appName} from './app.json';
8
9  AppRegistry.registerComponent(appName, () => AppFixedDimensions);
```

Hasilnya dapat dilihat pada Gambar 6. Secara umum jika komponen-komponen `View` yang digunakan akan tersusun ke bawah seperti terlihat pada gambar sebelah kiri.



Gambar 6. Hasil AppFixedDimensions.

Jika ukuran lebar dan tinggi komponen `View` dibuat sama seperti yang terlihat pada potongan kode yang terlihat pada Kode 16. Hasilnya dapat dilihat pada gambar di sebelah kanan pada Gambar 6. Terlihat pada gambar walau ukuran komponen-komponen kecil namun tetap tersusun secara vertikal.

Kode 16. Komponen View dengan ukuran lebar dan tinggi sama.

AppFixedDimensions

```
...
const AppFixedDimensions = () => {
  return (
    <View>
      <View style={{width: 100, height: 100, backgroundColor: 'powderblue'}} />
      <View style={{width: 100, height: 100, backgroundColor: 'skyblue'}} />
      <View style={{width: 100, height: 100, backgroundColor: 'steelblue'}} />
    </View>
  );
};
...
```

Percentage Dimension

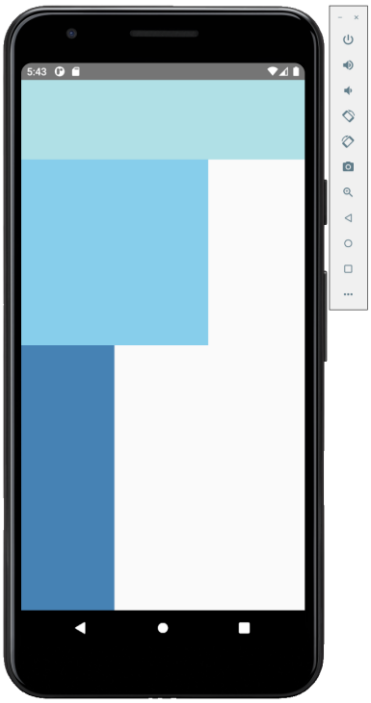
Untuk memenuhi layar dengan komponen dapat juga menggunakan memberikan nilai prosentase pada atribut `width` dan `height`. Contohnya dapat dilihat pada Kode 17.

Kode 17. AppPercentageDimensions.js

AppPercentageDimensions.js

```
1  import React from 'react';
2  import { View } from 'react-native';
3
4  const AppPercentageDimensions = () => {
5    return (
6      <View style={{ height: '100%' }}>
7        <View style={{height: '15%', backgroundColor: 'powderblue'}} />
8        <View style={{width: '66%', height: '35%', backgroundColor: 'skyblue'}} />
9        <View style={{width: '33%', height: '50%', backgroundColor: 'steelblue'}} />
10     </View>
11   );
12 };
13
14 export default AppPercentageDimensions;
```

Hasilnya dapat dilihat pada Gambar 7.



Gambar 7. Hasil AppPercentageDimensions.

Flex Dimension

Komponen yang menggunakan flex dimension akan memenuhi ruang yang tersedia secara dinamis. Untuk menggunakan dimensi ini digunakan atribut `flex` pada `style`. Sebagai contoh dengan memberikan nilai

1 pada atribut `flex` (`flex: 1`) maka komponen akan memenuhi seluruh ruang yang tersedia namun akan berbagi dengan komponen yang memiliki induk komponen yang sama.

Untuk melihat perbedaan antara penggunaan atribut `flex` pada `style` dapat dibandingkan kedua kode berikut ini.

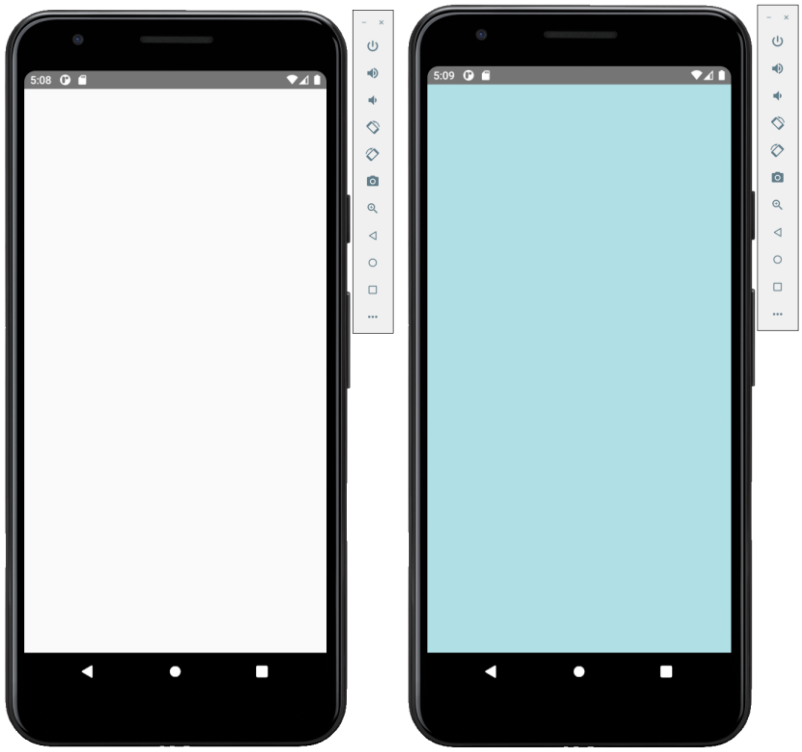
AppFlexDimensions

```
1  import React from 'react';
2  import { View } from 'react-native';
3
4  const AppFlexDimensions = () => {
5    return (
6      <View style={{backgroundColor: 'powderblue'}}>
7      </View>
8    );
9  };
10
11 export default AppFlexDimensions;
```

Kemudian ganti baris ke-6 dengan kode berikut.

```
<View style={{flex:1, backgroundColor: 'powderblue'}}>
```

Hasilnya dapat dilihat pada kedua gambar pada Gambar 8.



Gambar 8. Perbandingan dimensi antara menggunakan flex dan tidak.

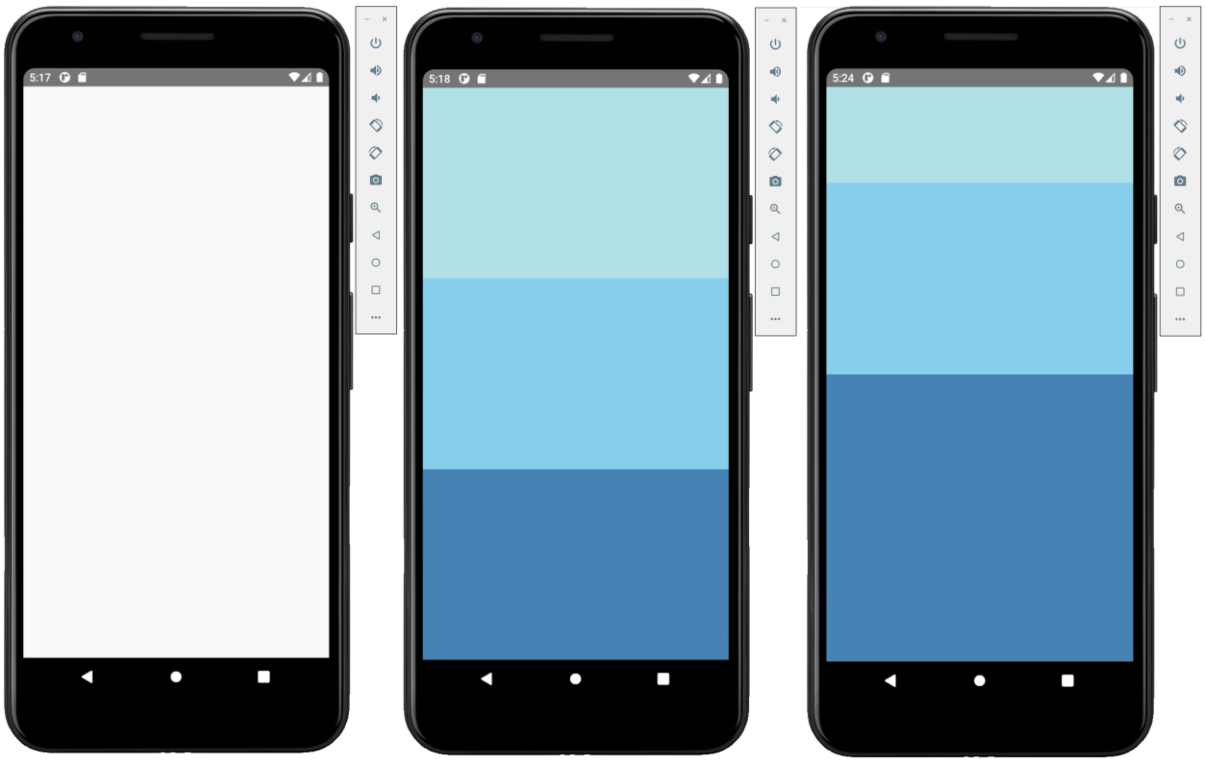
Selanjutnya akan ditambahkan 3 komponen `View` sebagai komponen anak dari komponen `View` yang telah ada seperti yang terlihat pada kode berikut.

Kode 18. AppFlexDimensions.js dengan 3 komponen anak tanpa style flex.

AppFlexDimensions.js

```
1  import React from 'react';
2  import { View } from 'react-native';
3
4  const AppFlexDimensions = () => {
5    return (
6      <View style={{flex:1}}>
7          <View style={{backgroundColor: 'powderblue' }} />
8          <View style={{backgroundColor: 'skyblue' }} />
9          <View style={{backgroundColor: 'steelblue' }} />
10     </View>
11   );
12 };
13
14 export default AppFlexDimensions;
```

Pada kode di atas ketiga komponen `View` tidak memiliki antribut `flex`. Hasilnya dapat dilihat pada gambar sebelah kiri di Gambar 9, tidak terlihat ada komponen `View` dengan warna yang berbeda.



Gambar 9. Hasil AppFlexDimensions.

Selanjutnya tambahkan atribut `flex:1` pada setiap komponen tersebut seperti pada potongan kode berikut. Hasilnya dapat dilihat pada gambar di tengah pada Gambar 9.

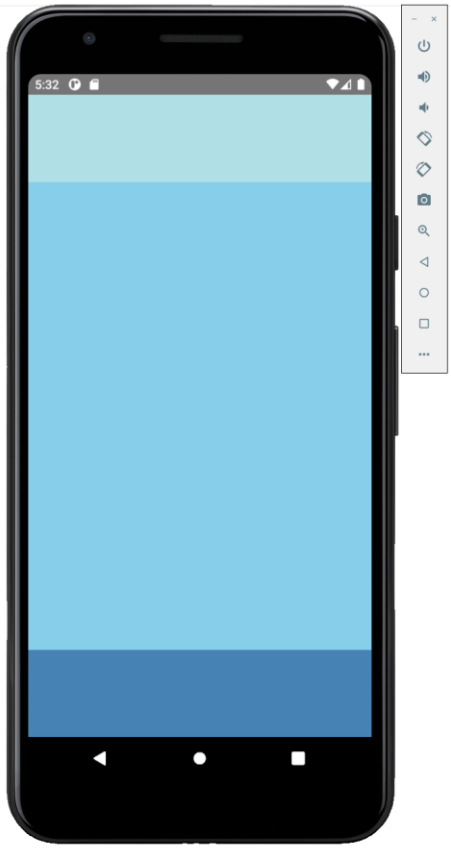
```
1  <View style={{flex:1}}>
2      <View style={{ flex: 1, backgroundColor: 'powderblue' }} />
3      <View style={{ flex: 1, backgroundColor: 'skyblue' }} />
```

```
4       <View style={{ flex: 1, backgroundColor: 'steelblue' }} />
5   </View>
```

Nilai atribut `flex` ini berpengaruh terhadap ukuran dari komponen, misal nilai atribut `flex` untuk komponen `View` anak yang pertama adalah 1, kemudian yang kedua adalah 2 dan yang terakhir adalah 3 atau seperti yang terlihat pada potongan kode di bawah ini. Maka hasilnya dapat dilihat di gambar sebelah kanan pada Gambar 9.

```
1   <View style={{flex:1}}>
2       <View style={{ flex: 1, backgroundColor: 'powderblue' }} />
3       <View style={{ flex: 2, backgroundColor: 'skyblue' }} />
4       <View style={{ flex: 3, backgroundColor: 'steelblue' }} />
5   </View>
```

Dari ketiga contoh kode di atas dapat dilihat bagaimana penggunaan `flex` untuk membuat dimensi komponen.



Gambar 10. Gabungan penggunaan fixed dan flex dimension.

Contoh berikut adalah gabungan dari penggunaan cara fixed dan flex dimension seperti yang terlihat pada potongan kode berikut ini. Hasilnya dapat dilihat pada Gambar 10.

```
<View style={{flex:1}}>
    <View style={{ height:100, backgroundColor: 'powderblue' }} />
    <View style={{ flex: 1, backgroundColor: 'skyblue' }} />
```

```
    <View style={{ height:100, backgroundColor: 'steelblue' }} />
</View>
```

Flex Direction

Pada contoh di atas dapat dilihat komponen tersusun secara vertikal atau berdasarkan kolom. Untuk mengubah susunan komponen-komponen anak pada komponen induk dapat memanfaatkan atribut `flexDirection`. Nilai atribut `flexDirection` ini adalah sebagai berikut:

- `column`, adalah nilai default yang digunakan. Nilai ini membuat susunan komponen secara vertikal dimulai dari atas ke bawah.
- `row`, jika nilai ini digunakan maka komponen-komponen akan tersusun dari kiri ke kanan.
- `column-reverse`, jika menggunakan nilai ini maka susunan komponen secara vertikal dimulai dari bawah ke atas.
- `row-reverse`, jika menggunakan nilai ini maka susunan komponen secara horisontal dimulai dari kanan ke kiri.

Berikut adalah kode yang dapat digunakan untuk contoh penggunaan `flexDirection` dengan menggunakan nilai `column`.

Kode 19. AppFlexDimensions dengan flexDirection:'column'

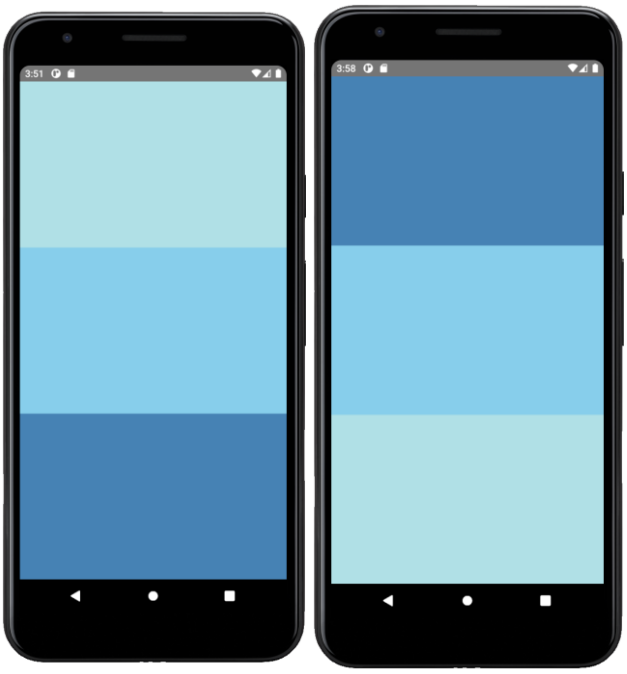
AppFlexDimensions.js

```
1  import React from 'react';
2  import { View } from 'react-native';
3
4  const AppFlexDimensions = () => {
5    return (
6      <View style={{flex:1, flexDirection:'column'}}>
7          <View style={{ flex: 1, backgroundColor: 'powderblue' }} />
8          <View style={{ flex: 1, backgroundColor: 'skyblue' }} />
9          <View style={{ flex: 1, backgroundColor: 'steelblue' }} />
10     </View>
11   );
12 };
13
14 export default AppFlexDimensions;
```

Penggunakan atribut `flexDirection` ini diberikan pada komponen induk seperti yang terlihat pada baris ke-6. Hasil dari kode ini dapat dilihat pada gambar sebelah kiri di Gambar 11. Kemudian ganti pada baris ke-6 untuk nilai `flexDirection` dengan `column-reverse` dengan cara berikut ini.

```
<View style={{flex:1, flexDirection:'column-reverse'}}>
```

Hasilnya dapat dilihat gambar sebelah kanan di Gambar 11. Jika dibandingkan antara kedua hasil pada gambar maka dapat dilihat susunan komponen terurut terbalik jika menggunakan `flexDirection: 'column-reverse'`.

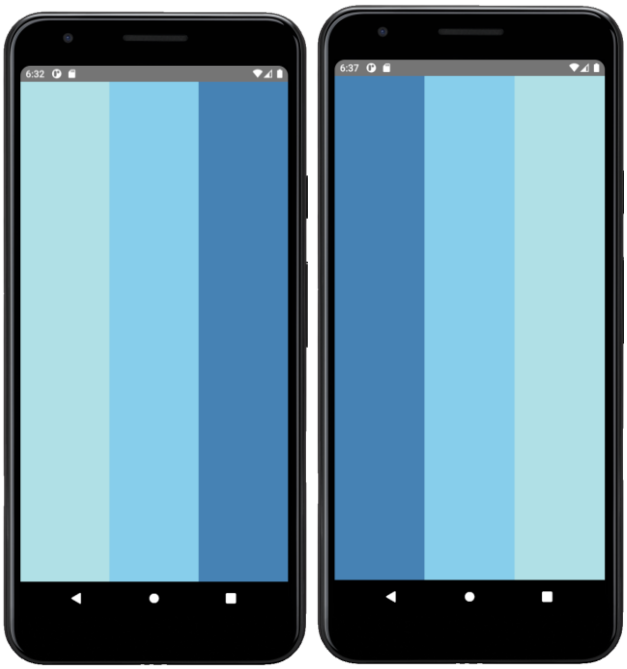


Gambar 11. Hasil penggunaan flexDirection dengan nilai column dan column-reverse.

Selanjutnya ubah nilai atribut `flexDirection` dengan nilai `row` dan `row-reverese` dengan memodifikasi pada baris ke-6. Untuk penggunaan nilai `flexDirection:'row'` dapat dilihat contoh potongan kode berikut.

```
<View style={{flex:1, flexDirection:'row'}}>
```

Hasilnya dapat dilihat pada gambar di sebelah kiri di Gambar 12.



Gambar 12. Hasil penggunaan flexDirection dengan nilai row dan row-reverse.

Selanjutnya ubah nilai `flexDirection` dengan `row-reverse` dengan kode ini dan hasilnya dapat dilihat pada gambar sebelah kanan di Gambar 12.

```
<View style={{flex:1, flexDirection:'row-reverse'}}>
```

Dari gambar tersebut dapat dilihat jika menggunakan nilai `row` untuk atribut `flexDirection` maka komponen akan disusun secara horisontal dari kiri ke kanan, dan akan tersusun dengan arah sebaliknya jika menggunakan nilai `row-reverse`.

Justify Content

Untuk mengatur posisi alignment komponen anak pada suatu ruang dapat digunakan atribut `justifyContent`. Implementasi atribut ini dapat dilihat pada Gambar 13. Pada contoh ini juga digunakan control `ScrollView` yang membuat area putih di tengah dapat dapat dinaik-turunkan. Pada gambar Gambar 13 di sebelah kiri dapat dilihat masih terdapat sisa konten pada bagian bawah, jika area konten warna putih tersebut dinaikkan maka dapat dilihat sisa konten tersebut seperti pada gambar di sebelah kanan.

Gambar 13. Hasil AppJustifyContent

Ada beberapa nilai yang dapat digunakan atribut ini yaitu:

- `flex-start`, jika dilihat pada gambar di atas maka nilai atribut ini membuat komponen anak tersusun dari posisi awal dan merapat.

- `flex-end`, nilai atribut ini membuat komponen anak tersusun ke posisi akhir dan merapat seperti yang terlihat pada gambar di atas.
- `center`, nilai atribut ini membuat komponen anak memposisi dirinya di tengah dan merapat.
- `space-between`, nilai atribut ini membuat komponen anak mengisi seluruh ruang yang ada dan terpisah oleh area kosong di antara dua komponen seperti yang terlihat pada gambar di atas.
- `space-around`, nilai atribut ini membuat komponen memiliki ruang kosong pada sisi kiri dan kanannya.
- `space-evenly`, nilai atribut ini membuat komponen memiliki area kosong antara komponen dengan ukuran yang sama.

Untuk implementasi atribut ini dapat dilihat pada Kode 20.

Kode 20. AppJustifyContent.js

AppJustifyContent.js

```
1  import React from 'react';
2  import { Text, View, Image, StyleSheet, ScrollView } from 'react-native';
3
4  const AppJustifyContent = () => {
5      return (
6          <View style={{flex:1, flexDirection:'column'}}>
7              <View style={{ height:75, backgroundColor: 'powderblue' }} />
8              <ScrollView style={{ flex: 1, backgroundColor: 'white' }}>
9                  <Text style={[styles.subTitle]}>flex-start</Text>
10                 <View style={[styles.countainer, {justifyContent:'flex-start'}]}>
11                     <View style={[styles.box, { backgroundColor: "powderblue" }]} />
12                     <View style={[styles.box, { backgroundColor: "skyblue" }]} />
13                     <View style={[styles.box, { backgroundColor: "steelblue" }]} />
14                 </View>
15
16                 <Text style={[styles.subTitle]}>flex-end</Text>
17                 <View style={[styles.countainer, {justifyContent:'flex-end'}]}>
18                     <View style={[styles.box, { backgroundColor: "powderblue" }]} />
19                     <View style={[styles.box, { backgroundColor: "skyblue" }]} />
20                     <View style={[styles.box, { backgroundColor: "steelblue" }]} />
21                 </View>
22
23                 <Text style={[styles.subTitle]}>center</Text>
24                 <View style={[styles.countainer, {justifyContent:'center'}]}>
25                     <View style={[styles.box, { backgroundColor: "powderblue" }]} />
26                     <View style={[styles.box, { backgroundColor: "skyblue" }]} />
27                     <View style={[styles.box, { backgroundColor: "steelblue" }]} />
28                 </View>
29
30                 <Text style={[styles.subTitle]}>space-between</Text>
31                 <View style={[styles.countainer, {justifyContent:'space-between'}]}>
32                     <View style={[styles.box, { backgroundColor: "powderblue" }]} />
33                     <View style={[styles.box, { backgroundColor: "skyblue" }]} />
34                     <View style={[styles.box, { backgroundColor: "steelblue" }]} />
35                 </View>
36
37                 <Text style={[styles.subTitle]}>space-around</Text>
38                 <View style={[styles.countainer, {justifyContent:'space-around'}]}>
39                     <View style={[styles.box, { backgroundColor: "powderblue" }]} />
40                     <View style={[styles.box, { backgroundColor: "skyblue" }]} />
41                     <View style={[styles.box, { backgroundColor: "steelblue" }]} />
42                 </View>
```

```
43
44                  <Text style={[styles.subTitle]}>space-evenly</Text>
45                  <View style={[styles.countainer, {justifyContent:'space-evenly'}]}>
46                      <View style={[styles.box, { backgroundColor: "powderblue" }]} />
47                      <View style={[styles.box, { backgroundColor: "skyblue" }]} />
48                      <View style={[styles.box, { backgroundColor: "steelblue" }]} />
49                  </View>
50              </ScrollView>
51              <View style={{ height:75, backgroundColor: 'steelblue' }} />
52          </View>
53      );
54  }
55  export default AppJustifyContent;
56
57  const styles = StyleSheet.create({
58      box: {
59          width: 50,
60          height: 50,
61      },
62      countainer: {
63          margin:10,
64          flexDirection:'row'
65      },
66      subTitle: {
67          marginLeft: 10,
68          marginTop:10,
69          fontSize: 20,
70          fontWeight:'bold'
71      }
72  })
```

Pada baris ke-8 dan ke-50 dapat dilihat penggunaan komponen `ScrollView`. Sebelum menggunakan komponen ini maka perlu ditambahkan `ScrollView` untuk diimport seperti yang terlihat pada baris ke-2. Selanjutnya penggunaan atribut `justifyContent` dapat dilihat pada baris ke-10, ke-17, ke-24, ke-31, ke-38 dan ke-45.

Layout Absolute & Relative

Posisi komponen dengan komponen lain adalah bersifat relatif, artinya posisi relatif kedua adalah relatif dengan posisi komponen pertama. Posisi yang bersifat relatif ini sudah kita lihat dan gunakan pada sub bab di atas.

Untuk implementasi layout ini digunakan atribut `position` untuk setiap komponen. Contoh penerapannya dapat dilihat pada Kode 21.

Kode 21. AppRelativeAbsolute.js

```
    AppRelativeAbsolute.js
1   import React, {useState} from 'react';
2   import { Text, View, Image, StyleSheet, ScrollView, Button } from 'react-native';
3
4   const AppRelativeAbsolute = () => {
5       const [position, setPosition] = useState('relative');
6
7       return (
8           <View style={{flex:1, flexDirection:'column'}}>
9               <View style={{ height:75, backgroundColor: 'powderblue' }} />
```

```
10              <View style={{ flex: 1, backgroundColor: 'white' }}>
11                  <Text style={[styles.subTitle]}>{position}</Text>
12                  <View style={[styles.countainer]}>
13                        <View style={[styles.box, { backgroundColor: "powderblue", top:0,
14  left:0, position:position }]} />
15                         <View style={[styles.box, { backgroundColor: "skyblue", top:25,
16  left:25, position:position }]} />
17                        <View style={[styles.box, { backgroundColor: "steelblue", top:50,
18  left:50, position:position }]} />
19                  </View>
20              </View>
21              <View style={{ height:75, backgroundColor: 'steelblue'}}>
22                        <View  style={[styles.countainer,  {justifyContent:'space-evenly',
23  marginTop: 10}]}>
24                         <Button title="Relative" onPress={() => setPosition('relative')}
25  style={{backgroundColor: "powderblue" }} />
26                         <Button title="Absolute" onPress={() => setPosition('absolute')}
27  style={{ backgroundColor: "powderblue" }} />
28                  </View>
29              </View>
30          </View>
31      );
32  }
33  export default AppRelativeAbsolute;
34
35  const styles = StyleSheet.create({
36      box: {
37          width: 50,
38          height: 50,
39      },
40      countainer: {
41          margin:10,
42          flexDirection:'row'
43      },
44      subTitle: {
45          marginLeft: 10,
46          marginTop:10,
47          fontSize: 20,
48          fontWeight:'bold'
49      }
50  })
```

Hasilnya dapat dilihat pada Gambar 14. Pada kode ini digunakan state untuk menyimpan dan memberikan nilai pada atribut `position`. Untuk menggunakan state maka perlu dilakukan import `{useState}` seperti yang terlihat pada baris ke-1. Kemudian di dalam fungsi `AppRelativeAbsolute` dideklarasikan nama variable state, fungsi untuk mengubah nilai state dan status awal state seperti yang terlihat pada baris ke-5. Nilai state `position` ditampilkan pertama kali sebagai nilai pada komponen `Text` pada baris ke-11.

Pada Gambar 14 dapat dilihat terdapat tiga kotak. Kotak pertama dibuat dengan kode pada baris ke-13. Kotak ini memiliki nilai posisi dengan menggunakan atribut `top` dan `left`. Nilai `top:0` artinya posisinya adalah 0 dari atas dan nilai `left:0` artinya posisinya 0 dari kiri. Jika diperhatikan titik 0,0 tersebut adalah dari titik awal komponen `View` induk pada baris ke-12 tersebut dibuat. komponen `View` ini juga memiliki atribut `position` dengan nilai adalah dari variable state `position`. Sebagai yang kita lihat pada baris ke-5 nilai awal dari variable ini adalah relative.

Selanjutnya kotak pertama dibuat dengan kode pada baris ke-15. Komponen ini memiliki atribut-atribut yang sama namun nilai untuk atribut `top` dan `left` berbeda yaitu 25. Jika nilai atribut `position` adalah `relative` maka posisi `top` dan `left` akan relatif dengan posisi sebelumnya. Namun jika atribut `position` adalah `absolute` maka perhitungannya akan dimulai dari titik awal komponen induk. Selanjutnya untuk posisi komponen berikutnya akan menyesuaikan dengan komponen sebelumnya jika menggunakan nilai `relative`.

Gambar 14. Hasil AppRelativeAbsolute.

Kemudian pada gambar di atas dapat dilihat terdapat dua tombol pada area bawah yaitu:

- Tombol RELATIVE, tombol ini berfungsi untuk mengubah nilai state menjadi `relative` dengan kode yang dapat dilihat pada baris ke-24. Pada tombol ini dapat dilihat terdapat event `onPress` yang berisi fungsi `setPosition('relative')`.
- Tombol ABSOLUTE, tombol ini berfungsi untuk mengubah nilai state menjadi `absolute` dengan kode yang dapat dilihat pada baris ke-26. Pada tombol ini dapat dilihat terdapat event `onPress` yang berisi fungsi `setPosition('absolute')`.

Jika tombol RELATIVE ditekan maka hasilnya seperti pada gambar kiri di Gambar 14. Dan jika tombol ABSOLUTE ditekan maka hasilnya seperti pada gambar kanan di Gambar 14.

Tugas

Lakukan tugas-tugas berikut:

1. Ikuti seluruh instruksi di atas.
2. Modifikasi AppJustifyContent.js agar memiliki 6 tombol untuk mengubah tampilannya dengan mengubah nilai justifyContent sesuai dengan tombol yang ditekan. (seperti yang dicontohkan pada sub bab Layout Absolute & Relative.

Referensi

1. shttps://reactjs.org/docs/hello-world.html
2. https://reactnative.dev/docs/components-and-apis
3. https://reactnative.dev/docs/stylesheet
4. https://reactnative.dev/docs/intro-react
5. https://reactnative.dev/docs/height-and-width