

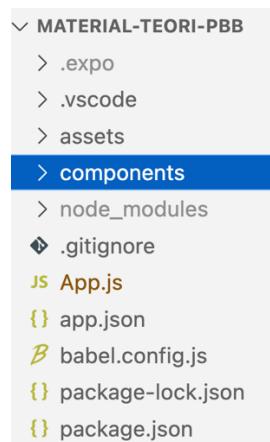


BAB II

REACT FUNDAMENTAL

1. Function Components (RFC) dan Class Components (RCC)

React Componen adalah bit kode yang independen dan dapat digunakan kembali. Komponen pada react memiliki tujuan yang sama dengan fungsi JavaScript, tetapi bekerja secara terpisah dan mengembalikan XHTML. Komponen react terdiri dalam dua jenis, komponen Class dan komponen Fungsi. Untuk membuat file-file komponen buatlah folder bernama components sejajar dengan file App.js.



Gambar 1. Menambahkan folder components

1.1. Class Component (RCC)

Komponen kelas harus menyertakan pernyataan `extends React.Component`. Pernyataan ini mengartikan bahwa class tersebut merupakan warisan untuk `React.Component`, dan memberikan akses komponen kepada fungsi React.Component. Komponen juga memerlukan metode `render()`, metode ini mengembalikan XHTML. Setiap component pada react selalu memiliki satu buah class. Berikut adalah contoh penerapan komponen CLASS pada react:



```
import { Text, View } from 'react-native'
import React, { Component } from 'react'

export class Car extends Component {
  render() {
    return (
      <View>
        <Text>Hi i'm a car</Text>
      </View>
    )
  }
}

export default Car
```



Import library pada JavaScript yang dapat digunakan pada project react. Library ini tersimpan pada folder node_module.

Render sebuah properties yang digunakan untuk menampilkan komponen yang telah terbentuk. Untuk dapat merender membutuhkan sebuah return statement untuk menampilkan expresi JSX

JSX JavaScript XML memungkinkan menuliskan syntax HTML kedalam React.

Export Default berfungsi untuk menginformasikan bahwa main programnya ada di komponen tersebut.

constructor() - Dipanggil selama fase kontruksi awal komponen React. Digunakan untuk mengatur status awal dan properti komponen.

Ubahlah script *App.js* untuk menguji file komponen yang telah anda buat dengan cara memanggil komponen class Car dalam bentuk JSX, komponen ini akan menjadi titik awal pada aplikasi anda.

```
JS App.js > ...
1 import Car from './components/Car';
2
3 export default function App() {
4   return (
5     <Car />
6   );
7 }
```

Setelah menambahkan *class Car* sebagai titik awal aplikasi maka tampilan pada simulator akan seperti berikut:



Gambar 1.1 Output React Class Component

1.2. Function Component (RFC)

RFC juga mengembalikan XHTML, dan memiliki sifat yang hampir sama dengan komponen Kelas, tetapi komponen Fungsi dapat ditulis menggunakan lebih sedikit kode, lebih mudah dipahami. Dalam membuat sebuah komponen fungsi pada react, dapat dibangun dalam dua bentuk RFC, berikut adalah contoh penerapan RFC dengan dua tipe scripting. Buatlah file bernama Motorcycle.js dan Bicycle.js didalam folder components, dan masukan script seperti dibawah ini:



Motorcycle.js	Bicycle.js
<pre>import { Text, View } from "react-native"; import React from "react"; const Motorcycle = () => { return (<View> <Text>Hi i'm a Motorcycle</Text> </View>); }; export default Motorcycle;</pre>	<pre>import { View, Text } from "react-native"; import React from "react"; function Bicycle() { return (<View> <Text>Hi i'm a Bicycle</Text> </View>); } export default Bicycle;</pre>
Mengubah Titik Awal Aplikasi	
<pre>JS App.js > ... 1 import Motorcycle from './components/Motorcycle'; 2 3 export default function App() { 4 return (5 <Motorcycle /> 6); 7 }</pre>	<pre>JS App.js > ... 1 import Bicycle from './components/Bicycle'; 2 3 export default function App() { 4 return (5 <Bicycle /> 6); 7 }</pre>
Output	

1.3. Perbedaan antara RCC dengan RFC

Function Component (RFC)	Class Component (RRC)
Komponen fungsional hanyalah fungsi murni JavaScript biasa yang menerima props sebagai argumen dan mengembalikan elemen React (JSX).	Komponen kelas mengharuskan Anda untuk memperluas dari React. Komponen dan buat fungsi render yang mengembalikan elemen React.
Tidak ada metode render yang digunakan dalam komponen fungsional.	Itu harus memiliki metode render() yang mengembalikan JSX (yang secara sintaksis mirip dengan HTML)
Komponen fungsional berjalan dari atas ke bawah dan setelah fungsi dikembalikan, itu tidak dapat tetap hidup.	Komponen kelas dibuat dan metode siklus hidup yang berbeda tetapi hidup dan dijalankan dan dipanggil tergantung pada fase komponen kelas.
Juga dikenal sebagai komponen Stateless karena mereka hanya menerima data dan menampilkannya dalam beberapa bentuk, bahwa mereka terutama bertanggung jawab	Juga dikenal sebagai komponen Stateful karena mereka menerapkan logika dan keadaan.



untuk merender UI.	
Metode React Lifecycle (misalnya, componentDidMount) tidak dapat digunakan dalam komponen fungsional.	Metode React Lifecycle dapat digunakan di dalam komponen kelas (misalnya, componentDidMount).
Hooks dapat dengan mudah digunakan dalam komponen fungsional untuk membuatnya stateful. Contoh: const [name, SetName]= React.useState('')	Ini membutuhkan syntax yang berbeda di dalam komponen kelas untuk mengimplementasikan hooks. Contoh: constructor(props){ super(props); this.state={name:''}}
Constructors tidak digunakan.	Constructors digunakan sesuai kebutuhan untuk menyimpan status.

2. Penggunaan State pada RFC dan RCC

State seperti penyimpanan data pribadi milik komponen. State berguna untuk menangani data yang berubah dari waktu ke waktu atau yang berasal dari interaksi pengguna. State akan menyiapkan sebuah memori komponen pada aplikasi.

2.1. Tipe data dan Variable

Tipe data pada Javascript ada tiga buah yaitu const, let, dan var. Berikut ini adalah contoh penggunaan variable terhadap tipe data tersebut:

Name	Scope	Desc
const	Block scope	Berisi nilai tetap dan tidak bisa diubah-ubah
let	Block scope	Nilai dapat diubah
var	Functional scope	Nilai dapat diubah dan diakses diluar block kecuali diluar function

Contoh:

```
const radian = 1
console.log(radian)
```

```
let isActive = true
console.log(isActive)
```

```
var total = 10.3
console.log(total)
```

```
const bulan = 'mei'
bulan = 'juni'
console.log(bulan)
```

```
let name = 'Febry'
console.log(name)
```

```
var obj ={title:'Pem Perangkat Bergerak', id:1}
console.log(obj)
```

```
const arrayObj = [{title:'Pem Perangkat Bergerak', id:1}, {title:'Pem Web', id:2}]
console.log(arrayObj)
```

2.2. State variable pada RFC

Berikut ini adalah contoh penerapan State pada RFC:

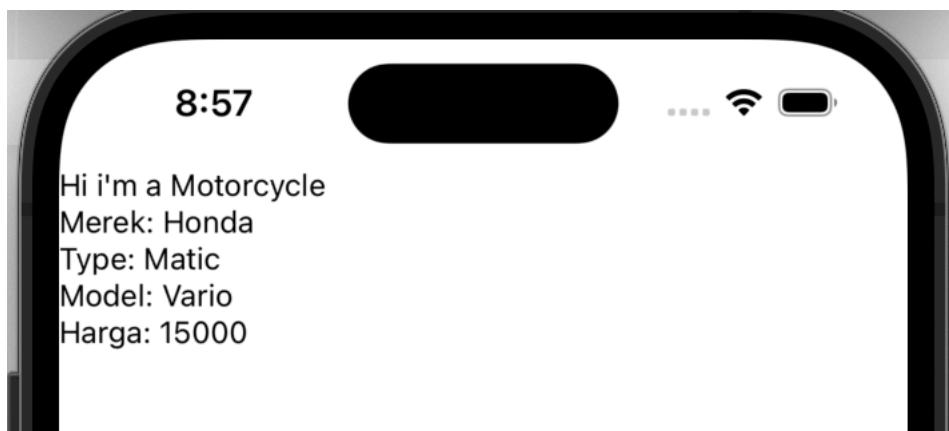
```
import { Text, View } from "react-native";
import React from "react";

var name = "Honda";
const Motorcycle = () => {
  return (
    <View>
      <Text>Hi i'm a Motorcycle</Text>
      <Text>Merek: {name}</Text>
      <Text>Type: {types.type}</Text>
      <Text>Model: {types.model}</Text>
      <Text>Harga: {types.harga}</Text>
    </View>
  );
};

export default Motorcycle;

const types = {type:"Matic", model:"Vario", harga:15000};
```

Pada script diatas jika ingin menampilkan state variable name dan types pada JSX maka harus menggunakan symbol Expression yaitu dengan menambahkan kurung kurawal {...}. Output dari script diatas sebagai berikut:



Gambar 2.2. Output penggunaan state variable pada RFC

2.3. State variable pada RCC

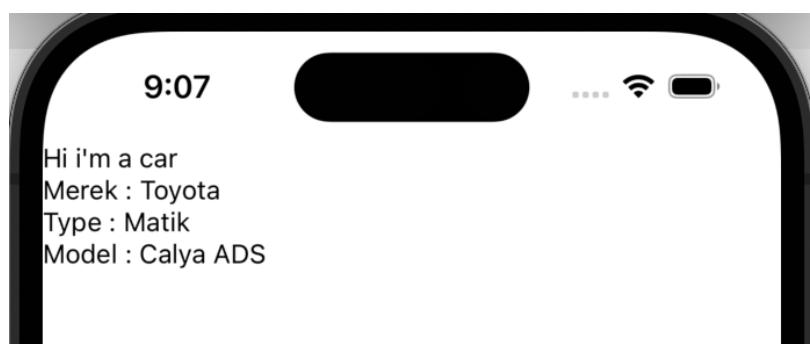
Penerapan state variable pada RCC berbeda dengan bentuk dari RFC. Bentuk penyimpanan data state di RCC lebih complex dan rumit. Berikut adalah contoh penerapan state variable pada RCC:

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'

export class Car extends Component {
  constructor(props){
    super(props);
    this.state={
      merek:"Toyota",
      types:{type:"Matik", model:"Calya ADS"}
    }
  }
  render() {
    return (
      <View>
        <Text>Hi i'm a car</Text>
        <Text>Merek : {this.state.merek}</Text>
        <Text>Type : {this.state.types.type}</Text>
        <Text>Model : {this.state.types.model}</Text>
      </View>
    )
  }
}

export default Car
```

Pada script diatas sama halnya dengan RFC untuk memanggil sebuah state kedalam JSX harus menggunakan symbol Expression yaitu menggunakan kurung kurawal {...}, namun pada RCC pemanggilan state variable harus diawali dengan `this.state.nama_variable`. Berikut adalah contoh dari output script diatas:



Gambar 2.3. Output penggunaan state variable pada RCC

2.4. State Function pada RFC

State function JavaScript merupakan blok kode yang dirancang untuk melakukan tugas tertentu. Fungsi JavaScript dijalankan ketika "sesuatu" memanggilnya (memanggilnya). Berikut adalah contoh penggunaan state function pada RFC:

```
import { View, Text } from "react-native";
import React from "react";

function Bicycle() {
  return (
    <View>
      <Text>Hi i'm a Bicycle</Text>
      <TakeARide />
      {Place2Go()}
    </View>
  );
}

export default Bicycle;

const TakeARide = () => {
  return <Text>Let's go riding with me</Text>;
};

function Place2Go() {
  return <Text>We're going to south west now, come on.</Text>;
}
```

Pada script diatas memiliki sebuah fungsi umum bernama *TakeARide()* dan *Place2Go()*, kedua fungsi tersebut dipanggil didalam JSX dengan dua acara yang berbeda. Untuk fungsi *TakeARide()* dipanggil dengan menggunakan tag XHTML, sedangkan untuk fungsi *Place2Go()* menggunakan Expression.



Gambar 2.4.a. Output penerapan state function pada RFC

Biasanya dalam sebuah fungsi selalu ada sebuah aksi berupa pengiriman sebuah data atau informasi dalam bentuk variable. Aksi passing parameter yang dapat digunakan pada RFC sebagai berikut:

```

function Bicycle() {
    const city = "south west";
    const peoples = [{name:"Erdiana", fams:"Sister"}, 
                    {name:"Emanuel", fams:"Brother"}];
    return (
        <View>
            <Text>Hi i'm a Bicycle</Text>
            <TakeARide peoples={peoples} />
            {Place2Go(city)}
        </View>
    );
}

export default Bicycle;

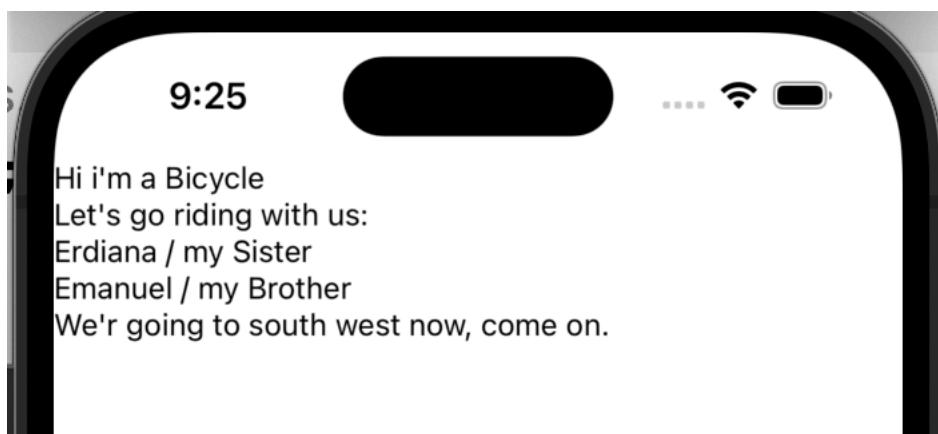
const TakeARide = ({peoples}) => {
    return (
        <View>
            <Text>Let's go riding with us:</Text>
            {peoples.map((v,index)=>(
                <View key={index}>
                    <Text>{v.name} / my {v.fams}</Text>
                </View>
            )));
        </View>
    );
};

function Place2Go(value) {
    return <Text>We'r going to {value} now, come on.</Text>;
}

```

Pada script diatas jika mengirimkan sebuah parameter kedalam bentuk XHTML seperti pada function `<TakeARide />`, parameter tersebut perlu disimpan dalam bentuk attribute. Dan ketika didalam fungsi `TakeARide()` nilai parameter harus dalam bentuk objek dan penamaannya harus sama dengan nama attribute yang dikirimkan.

Sedangkan pada fungsi `Place2Go()`, cukup mengirimkannya kedalam isian expression `Place2Go("value")`. Namun jika yang dikirimkan dalam bentuk objek anda dapat menggunakan symbol bracket objek seperti pada fungsi `TakeARide({variable})` atau dapat menggunakan *props*. Berikut adalah contoh penerapan state function pada RFC untuk passing parameters:



Gambar 2.4.b. Output penerapan state function untuk passing parameters

2.5. State Function pada RCC

Berbeda halnya dengan RFC, penerapan sebuah fungsi didalam RCC bisa dikatakan tidak sesimple di RFC. Berikut adalah contoh penerapan fungsi pada RCC:

```
export class Car extends Component {
  constructor(props) {
    super(props);
    this.Come2Go = this.Come2Go.bind(this);
    this.state = {
      merek: "Toyota",
      types: { type: "Matik", model: "Calya ADS" },
    };
  }

  Come2Go(value){
    return (
      <View>
        <Text>Let's go running away from duty</Text>
        <Text>with us only {value} IDR</Text>
      </View>
    )
  }

  render() {
    return (
      <View>
        <Text>Hi i'm a car</Text>
        <Text>Merek : {this.state.merek}</Text>
        <Text>Type : {this.state.types.type}</Text>
        <Text>Model : {this.state.types.model}</Text>
        {this.Come2Go(200000)}
      </View>
    );
  }
}

export default Car;
```



Dari script diatas untuk membuat sebuah fungsi di RCC, pertama kali yang perlu diperhatikan ialah menginisialisasi nama sebuah fungsi didalam `constructor()`. Dengan menuliskan script seperti berikut `this.Come2Go = this.Come2Go.bind(this)`. Didalam RCC untuk membuat sebuah function harus berada di bawah constructor dan bentuknya pun cukup dengan `NameOfFunction()`. Sedangkan untuk memanggil sebuah fungsi di RCC, sama halnya dengan state variable, kita perlu memanggilnya dengan bentuk Expression dengan diawali `this.NameOfFunction()`.

3. Core Component UI dalam JSX

3.1. Container UI

Container UI yang dapat digunakan untuk membangun sebuah output pada aplikasi terdiri dari View, SafeAreaView, ScrollView, dan ImageBackground. Penerapan container ini sama halnya seperti element block pada HTML yaitu `<div>`.

a. View

Container ini dapat digunakan lebih dari satu kali dalam JSX, ini membantu untuk membungkus tag JSX yang bersifat multiple element.

```
render() {
  return (
    <View>
      <View>
        <Text>Hi i'm a car</Text>
        <Text>Merek : {this.state.merek}</Text>
      </View>
      <View>
        <Text>Type : {this.state.types.type}</Text>
        <Text>Model : {this.state.types.model}</Text>
      </View>
      {this.Come2Go(200000)}
    </View>
  );
}
```

b. SafeAreaView

Container JSX dalam bentuk ini hanya digunakan hanya sekali dalam aplikasi, biasanya diinisialisasi didalam root atau titik awal pada component react. Tag JSX ini berguna untuk memposisikan layer UI mengikuti batas StatusBar. Jika tidak menggunakan JSX ini maka start awal layer akan berada di titik X dan Y sama dengan 0.

```
export default function App() {
  return (
    <SafeAreaView>
      <Car />
    </SafeAreaView>
  );
}
```



c. ImageBackground

Sama halnya dengan container SafeAreaView, container ini hanya dapat digunakan sekali saja dalam inisialisasinya. Container ini diperuntukan jika ingin memiliki background gambar pada aplikasinya.

```
export default function App() {
  return (
    <ImageBackground
      source={{
        url: "https://kis.ibik.ac.id/environment/ibik/images/background.jpg",
      }}
      resizeMode="cover"
      style={{ flex: 1}}
    >
      <SafeAreaView>
        <Car />
      </SafeAreaView>
    </ImageBackground>
  );
}
```

d. ScrollView

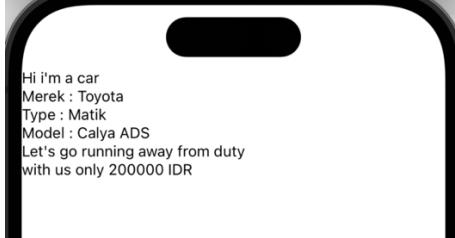
Jika anda ingin memiliki layout dalam bentuk scrolling anda dapat menggunakan container ini sebagai bentuk JSX yang digunakan pada elemen yang memiliki data atau informasi yang panjang.

```
return (
  <ScrollView>
    <View>
      <Text>Hi i'm a car</Text>
      <Text>Merek : {this.state.merek}</Text>
    </View>
    <View>
      <Text>Type : {this.state.types.type}</Text>
      <Text>Model : {this.state.types.model}</Text>
    </View>
    {this.Come2Go(200000)}
  </ScrollView>
);
```

3.2. Basic UI

a. StatusBar

Elemen JSX `<StatusBar />` ini untuk mengontrol status aplikasi seperti zona status, biasanya berada pada bagian atas layar, yang menampilkan waktu, Wi-Fi dan informasi jaringan seluler, level baterai, dan/atau ikon status lainnya. Status Bar terdapat pada library expo, yaitu `import { StatusBar } from "expo-status-bar"`.

Code	Output
<pre>export default function App() { return (<SafeAreaView> <StatusBar hidden={true} /> <Car /> </SafeAreaView>); }</pre>	 <p>Hi i'm a car Merek : Toyota Type : Matik Model : Calya ADS Let's go running away from duty with us only 200000 IDR</p>

Berikut adalah attribute yang dapat digunakan untuk element StatusBar:

Attribute	Value
backgroundColor	Color, default 'black'
hidden	boolean

b. Text

Sebuah element JSX untuk menampilkan sebuah tulisan kedalam aplikasi.

`<Text>Hi i'm a car</Text>`

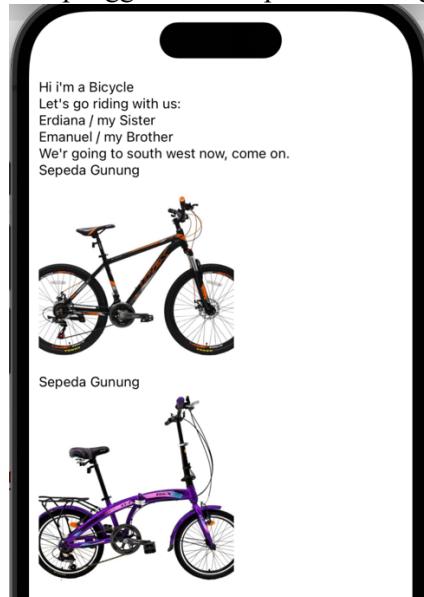
c. Image

Berikutnya ialah JSX UI untuk menampilkan berbagai jenis gambar, termasuk gambar jaringan, sumber daya statis, gambar lokal sementara, dan gambar dari disk lokal, seperti rol kamera.

```
return (
  <View style={{padding:10}}>
    <Text>Hi i'm a Bicycle</Text>
    <TakeARide peoples={peoples} />
    {Place2Go(city)}
    <View>
      <Text>Sepeda Gunung</Text>
      <Image
        source={{
          uri: "https://trexsporting.com/images/products/11-KbmXViHodZ.jpg",
        }}
        style={{width:200, height:200}}
      />
    </View>

    <View>
      <Text>Sepeda Gunung</Text>
      <Image
        source={require("../assets/icons/sepeda-lipat.jpeg")}
        style={{width:200, height:200}}
      />
    </View>
  </View>
);
```

Berikut adalah contoh output dari penggunaan komponen ui Image:

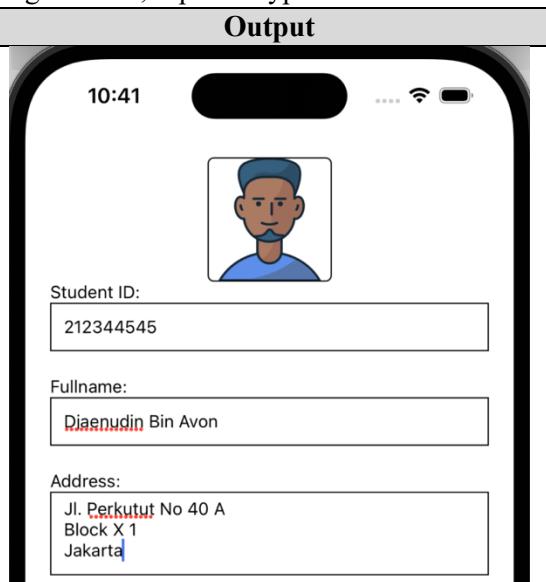


Gambar 3.2.c. Output penggunaan komponen UI Image

3.3. Forms UI

a. TextInput

Komponen dasar untuk memasukkan teks ke dalam aplikasi melalui keyboard. Komponen ini memiliki property konfigurasi untuk beberapa fitur, seperti koreksi otomatis, kapitalisasi otomatis, teks placeholder, dan jenis keyboard yang berbeda, seperti keypad numerik.

Code	Output
<pre> <View> <Text>Student ID:</Text> <TextInput style={styles.inputText} placeholder="Enter your NPM" keyboardType="numeric" /> </View> <View> <Text>Fullname:</Text> <TextInput style={styles.inputText} placeholder="Enter your name here" /> </View> <View> <Text>Address:</Text> <View> <TextInput editable multiline numberOfLines={4} maxLength={40} /> </View> </View> </pre>	 <p>10:41</p> <p>Student ID: 212344545</p> <p>Fullname: Diaeudin Bin Avon</p> <p>Address: Jl. Perkutut No 40 A Block X 1 Jakarta</p>

b. Buttons

Komponen tombol dasar yang seharusnya dirender dengan baik di platform apa pun. Mendukung tingkat penyesuaian minimal.

```
<Button  
    title="Button form OS"  
/>
```

Sebelum mencoba beberapa bentuk dari Touchable buatlah sebuah fungsi umum seperti dibawah ini:

```
const buttonAct = (title) => {  
  return (  
    <View  
      style={{  
        backgroundColor: "purple",  
        borderRadius: 10,  
        padding: 10,  
        alignItems: "center",  
        marginVertical: 5,  
      }}  
    >  
      <Text style={{ color: "white" }}>{title}</Text>  
    </View>  
  );  
};
```

- **TouchableOpacity**

Pembungkus untuk membuat tampilan merespons sentuhan dengan benar. Saat ditekan, opasitas tampilan terbungkus berkurang, meredupkannya.

```
<TouchableOpacity  
    activeOpacity={0.6}  
>  
  {buttonAct("Touchable Opacity")}  
</TouchableOpacity>
```

- **TouchableHighlight**

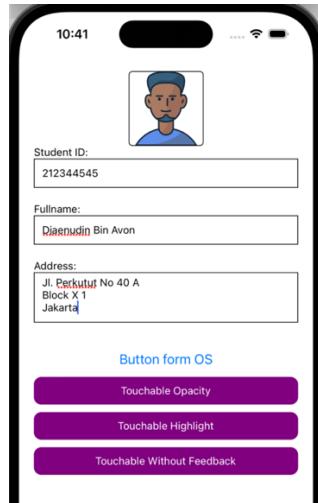
Pembungkus untuk membuat tampilan merespons sentuhan dengan benar. Saat ditekan, opasitas tampilan yang dibungkus akan berkurang, yang memungkinkan warna lapisan bawah terlihat, menggelapkan, atau mewarnai tampilan.

```
<TouchableHighlight  
    activeOpacity={0.6}  
>  
  {buttonAct("Touchable Highlight")}  
</TouchableHighlight>
```

- **TouchableWithoutFeedback**

Jangan gunakan kecuali Anda memiliki alasan yang sangat bagus. Semua elemen yang merespons pers harus memiliki umpan balik visual saat disentuh.

```
<TouchableWithoutFeedback  
    activeOpacity={0.6}  
>  
  {buttonAct("Touchable Without Feedback")}  
</TouchableWithoutFeedback>
```



Gambar 3.3. Output komponen UI Forms

3.4. Stylesheet

StyleSheet adalah abstraksi yang mirip dengan CSS StyleSheets. React Native menyediakan sejumlah komponen dasar yang dapat digunakan secara langsung tetapi menurut tema aplikasi, kita harus menyesuaikan komponen kadang-kadang dan itulah mengapa kita menggunakan stylesheet. Intinya Stylesheet adalah sebuah abstraksi yang mirip dengan stylesheet css. Pada CSS kode `background-color` digunakan untuk merubah *background color* sedangkan pada React Native menggunakan kode `backgroundColor`.

Terdapat 3 cara untuk membuat style pada tampilan aplikasi, antara lain :

3.4.1. Inline

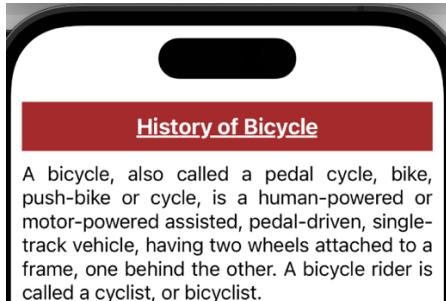
Kita bisa membuat style secara langsung pada component yang ingin kita beri style, untuk membuat style secara inline kita menggunakan props yang sudah disediakan . Akan tetapi cara ini tidak disarankan , karena akan mengotori kode . Dengan metode Inline kita tidak perlu import component StyleSheet dari react native.

```
import { View, Text, StyleSheet } from "react-native";
import React from "react";

const ItemTyphograph = () => {
  return (
    <View>
      <Text
        style={{
          fontSize: 20,
          fontWeight: "bold",
          backgroundColor:"brown",
          color:"#fff",
          textAlign:"center",
          textDecorationLine: "underline",
          padding:10,
          marginBottom:10
        }}
      >
        History of Bicycle
      </Text>
      <Text style={styles.paragraph}>
        A bicycle, also called a pedal cycle, bike, push-bike or cycle, is a human-powered or
        motor-powered assisted, pedal-driven, single-track vehicle, having two wheels attached
        to a frame, one behind the other. A bicycle rider is called a cyclist.
      </Text>
    </View>
  );
};

export { ItemTyphograph };
```

Untuk membuat stylesheet secara inline pada script diatas kita dapat memasang attribute style pada tag element JSX. Seperti pada `<Text style={{key:value}}>...</Text>`. Property lengkap dari stylesheet react native dapat dilihat di <https://reactnative.dev/docs/text#style>. Berikut adalah hasil output dari script diatas:



Gambar 3.4.1. Output penerapan stylesheet inline

3.4.2. Embed

Kita bisa juga membuat style tanpa mengotori kode salah satunya dengan cara embed. Embed adalah cara membuat style di dalam satu file , tetapi tidak secara langsung di dalam component. Kita bisa menggunakan built in component yang sudah disediakan oleh React Native yaitu dengan StyleSheet. Pada penerapan kali ini kita membutuhkan library dari react-native untuk menggunakan StyleSheet seperti `import { StyleSheet } from "react-native"`. Masih pada file yang sama yaitu `ItemTyphograph()`. Pada JSX Text kedua yang berupa paragraph, disini akan ditambahkan sebuah style embedded.

```
const ItemTyphograph = () => {
  return (
    <View>
      <Text>...
      </Text>

      <Text style={styles.paragraph}>
        A bicycle, also called a
        <Text style={{...styles.paragraph, color:"red"}}>pedal cycle</Text>,
        <Text style={{...styles.paragraph, fontWeight:"bold"}}>bike</Text>,
        <Text style={{...styles.paragraph, fontStyle:"italic"}}>push-bike or cycle</Text>,
        is a human-powered or motor-powered assisted, pedal-driven, single-track vehicle,
      </Text>
    </View>
  );
};

export { ItemTyphograph };

const styles = StyleSheet.create({
  paragraph :{
    fontSize: 18, textAlign:"justify"
  }
})
```

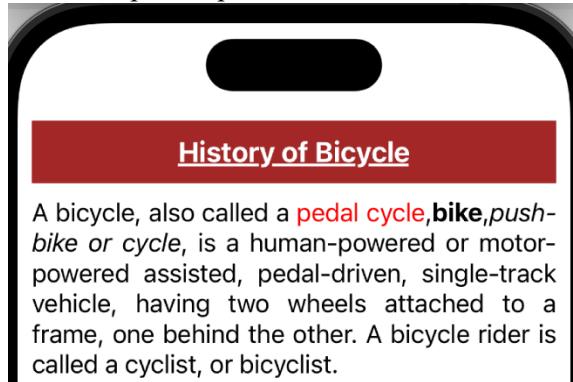
Pada script diatas untuk menggunakan style embedded kita perlu mendeklarasikannya kedalam bentuk variables. Disini variable stylesheet disimpan dengan nama styles, dimana variable ini memiliki bentuk objek dari library StyleSheet. Key paragraphf disini menyimpan 1 bentuk objek, yang berisi property fontSize dan textAlign. Dan key paragraphf akan digunakan untuk elemen JSX Text (sesuai pada gambar kotak). Pada script diatas, tulisan pedal cycle, bike, dan push-bike or cycle,



memiliki custome styling, di mana jika kita ingin mengupdate state tanpa perlu mengubah inti value pada state, kita dapat menggunakan metode *Spread Operator*, yaitu dengan cara seperti berikut:

{...object, value}

Berikut adalah contoh hasil dari output script diatas:



Gambar 3.4.2. Output penerapan stylesheet embed

3.4.3. External

Kita bisa juga membuat style tanpa mengotori kode dengan cara selain embed yaitu dengan cara external. External adalah cara membuat style tidak dalam satu file melainkan pada file terpisah. Contoh, pada source code sebelumnya variable styles dapat kita simpan dalam bentuk file js baru dan memanggil file tersebut di ItemTyphograph().

3.4.4. Stylesheet untuk Layout

Berikut ini terdapat beberapa property yang umum digunakan untuk membangun sebuah Layouting pada desain UI:

- Flex

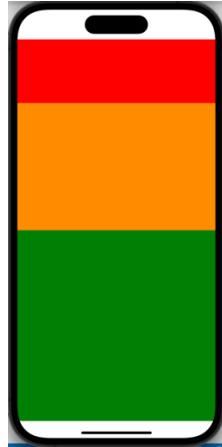
flex akan menentukan bagaimana item akan "mengisi" ruang yang tersedia di sepanjang sumbu utama layar. Space akan dibagi menurut properti flex masing-masing elemen.

```
import { StatusBar } from "expo-status-bar";
import { SafeAreaView, StyleSheet, View } from "react-native";

export default function App() {
  return (
    <SafeAreaView style={styles.container}>
      <StatusBar hidden={true} />
      <View style={{ flex: 1, backgroundColor: "red" }} />
      <View style={{ flex: 2, backgroundColor: "darkorange" }} />
      <View style={{ flex: 3, backgroundColor: "green" }} />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  paragraph: {
    fontSize: 18,
    textAlign: "justify",
  },
  container: {
    flex: 1
  },
});
```

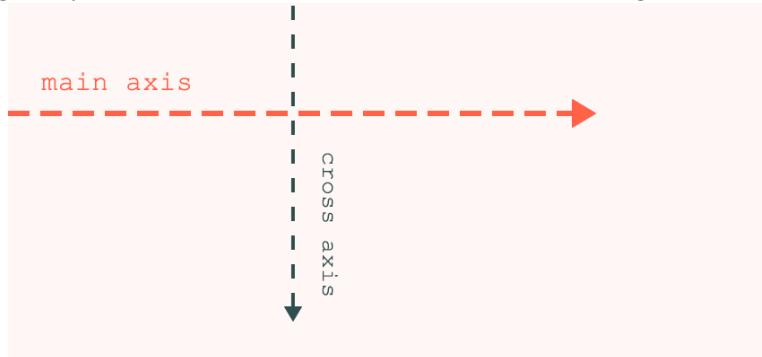
Dalam contoh berikut, tampilan merah, oranye, dan hijau adalah semua anak dalam tampilan kontainer yang memiliki set flex: 1. Tampilan merah menggunakan flex: 1 , tampilan oranye menggunakan flex: 2, dan tampilan hijau menggunakan flex: 3 . $1+2+3 = 6$, artinya tampilan merah akan mendapatkan $1/6$ ruang, oranye $2/6$ ruang, dan hijau $3/6$ ruang.



Gambar 3.4.4. Output penerapan Flex pada stylesheet

- Flex Direction

FlexDirection mengatur arah Main Axis. Dimana nilai default dari FlexDirection ialah column, yang artinya Cross Axis akan berada di sumbu vertical, bergerak dari atas ke bawah.



column - (nilai default) mengatur posisi node-node berada dari atas ke bawah. Jika pembungkus diaktifkan, baris berikutnya akan dimulai di sebelah kanan item pertama di bagian atas area.

row – mengatur posisi node-node dari kiri ke kanan. Jika pembungkus diaktifkan, baris berikutnya akan dimulai di bawah item pertama di sebelah kiri wadah.

column-reverse - mengatur posisi node-node dari bawah ke atas. Jika pembungkus diaktifkan, baris berikutnya akan dimulai di sebelah kanan item pertama di bagian bawah wadah.

row-reverse - mengatur posisi node-node dari kanan ke kiri. Jika pembungkus diaktifkan, baris berikutnya akan dimulai di bawah item pertama di sebelah kanan wadah.

Berikut adalah contoh script awal tanpa menggunakan flexdirection pada stylesheet:

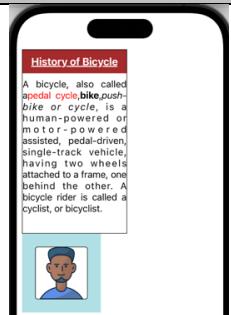


```
import { StatusBar } from "expo-status-bar";
import { SafeAreaView, StyleSheet, View } from "react-native";
import { ItemImage, ItemTypograph } from "./components/stylesheets/Items";

export default function App() {
  return (
    <SafeAreaView style={{ flex: 1 }}>
      <StatusBar hidden={true} />
      <View style={styles.container}>
        <ItemTypograph />
        <ItemImage />
      </View>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  paragraph: {
    fontSize: 18,
    textAlign: "justify",
  },
  container: {
    padding: 10,
    flex: 1,
  },
});
```

Dari script diatas kita akan menambahkan property flexDirection pada key container, sehingga dapat melihat bagaimana node-node didalamnya ItemTypograph dan ItemImage diatur sesuai bentuk dari flexDirection.

Flex Direction	Output
<pre>container: [padding: 10, flex: 1, flexDirection:"column"],</pre>	
<pre>container: { padding: 10, flex: 1, flexDirection:"column-reverse" },</pre>	



<pre>container: { padding: 10, flex: 1, flexDirection:"row" },</pre>	
<pre>container: { padding: 10, flex: 1, flexDirection:"row-reverse" },</pre>	

- Justify Content

JustifyContent menjelaskan cara menyusun node-node di dalam poros utama pembungkusnya. Misalnya, menggunakan properti ini untuk memusatkan node secara horizontal di dalam pembungkus dengan flexDirection diatur ke row atau vertikal di dalam pembungkus dengan flexDirection diatur ke column.

Justify Content	Output
<pre>container: { padding: 10, flex: 1, justifyContent:"center" },</pre>	
<pre>container: { padding: 10, flex: 1, justifyContent:"flex-end" },</pre>	

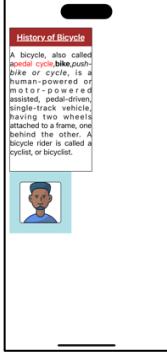


<pre>container: { padding: 10, flex: 1, justifyContent:"flex-start" },</pre>	
<pre>container: { padding: 10, flex: 1, justifyContent:"space-between" },</pre>	
<pre>container: { padding: 10, flex: 1, justifyContent:"space-around" },</pre>	

- Align Items

alignItems menjelaskan cara menyelaraskan node-node di sepanjang Cross Axis pembungkusnya. Ini sangat mirip dengan justifyContent tetapi alih-alih diterapkan ke Main Axis, alignItems diterapkan ke Cross Axis.



Align Items	Output
<pre>container: { padding: 10, flex: 1, alignItems:"baseline" },</pre>	
<pre>container: { padding: 10, flex: 1, alignItems:"center" },</pre>	
<pre>container: { padding: 10, flex: 1, alignItems:"flex-end" },</pre>	
<pre>container: { padding: 10, flex: 1, alignItems:"flex-start" },</pre>	



4. Latihan Praktikum

1. Buatlah script untuk memposisikan node-node berada di tengah-tengah layar monitor seperti berikut:



2. Dalam membuat script nomor 1, buatlah sebuah RCC sebagai titik awal aplikasi dan untuk node-node nya menggunakan RFC (kotak tulisan history bicycle & kotak gambar). Panggil kedua RFC tersebut sebagai bagian dari node RCC yang akan anda buat.
3. Buatlah sebuah splash screen sederhana seperti gambar dibawah ini dan buatlah dengan menggunakan StyleSheet Inline dan Embeded.



4. Buatlah sebuah halaman untuk menampilkan informasi data diri anda, namun data mengenai informasi anda disimpan dalam bentuk sebuah Object sebagai berikut:

```
const MyBio = {  
    identity:{  
        npm:212310056 ,  
        firstname: "MUHAMMAD",  
        middlename:"ZIDAN",  
        lastname:"AKBAR",  
    },  
    educations:[  
        {id:1, school:"SDN 1 Kota Bogor"},  
    ]  
}
```



```
{id:2, school:"SMPN 1 Kota Bogor"},  
{id:3, school:"SMAN 1 Kota Bogor"},  
,  
mobile_phone: 0812345679,  
email:"212310056@student.ibik.ac.id",  
gender:'M',  
tall_body:175,  
weight_body:64.5  
}
```

Implementasikanlah variable diatas dengan menggunakan state variable RCC dan state variable RFC. Dan berikan output dari masing-masing component.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PPB-TI-21-[PA/KA]-NPM. File source code disimpan sesuai nama project-praktikum dan masukan kedalam repositori tersebut. Buatkanlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository anda kepada:

@FebryFairuz dan (@IrvanRizkyAriansyah atau @raiyanawinata)

Disusun Oleh	Disetujui Oleh
<u>Irvan Rizky Ariansyah</u> Penyusun I	<u>Raiyana Jan Winata</u> Penyusun II