Mobile Application Development

Source codes for lectures

Dr. Osman Khalid

Updated on 13 December 2022

http://osman.pakproject.com

Table of Contents

Javascript classes

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Untitled Document</title>

<script>
    class Car
    {
        constructor(name)
        {
          this.brand = name;
        }

        display()
        {
            console.log(this.brand);
        }
    }


    mycar = new Car("Ford");
    mycar.display();

</script>
```

```
</head>

<body>
</body>

</html>
```

Inheritance Example

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Untitled Document</title>

<script>
    class Car
    {
        constructor(name)
        {
          this.brand = name;
        }

        display()
        {
            console.log("Brand: " + this.brand + " Model: " + this.model);
        }
    }


    class Model extends Car
    {
        constructor(name, mod)
        {
            super(name);
            this.model = mod;

        }
    }


   mycar = new Model("Ford", "Ferrari");

    mycar.display();

</script>
```

```
</head>

<body>
</body>

</html>
```

Javascript One argument function

```
<script>
function Show(mystring)
{
alert("This is " + mystring);
}

Show("Programming");
</script>
```

Return a value from a function

```
<script>
function Show()
{
return "hello";
}

alert( Show() );
</script>
```

Alternate way of defining a regular function

```
<script>
Show = function()
{
return "hello";
}

alert( Show() );
</script>
```

Arrow function

```
<script>
Show = () =>
{
```

```
return "hello";
}

alert( Show() );
</script>
```

Passing arguments to arrow functions

```
<script>
Show = (mystr, mystr2) =>
{
return mystr + " " + mystr2;
}

alert( Show("hello", "world") );
</script>
```

Arrow function with single statement

```
<script>
Show = () => "hello";


alert( Show() );
</script>
```

Passing an argument to an arrow function

```
<script>
Show = (mystring) => "hello" + mystring;


alert( Show(" world") );
</script>
```

We can remove parenthesis from arrow functions if single argument is passed

```
<script>
Show = mystring => "hello" + mystring;


alert( Show(" world") );
</script>
```

Example of javascript map function

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Untitled Document</title>
<script>
const myArray = [65, 44, 12, 4];

//Method 1
const newArr1 = myArray.map((item) => item*10);

for(let item of newArr1)
{
    console.log(item);
}

//Method 2
const newArr2 = myArray.map(myFunction);

function myFunction(num)
{
    return num * 10;
}

for(let item of newArr2)
{
    console.log(item);
}

</script>
</head>

<body>
```

```
</body>
</html>
```

Calling a method with a button

```
<body>
<script>
    myfunction = function()
    {
        alert("Button is clicked");
    }

    </script>

    <button onclick="myfunction()">Click here</button>
</body>
```

Calling method with a button using event listener

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Untitled Document</title>

</head>

<body>
    <button id="btn" >Click here</button>

    <script>
        class Car
        {
            constructor(name)
            {
              this.brand = name;
            }

            display()
            {
                console.log(this);
            }
        }

    mycar = new Car("Ford");
    mycar.display();
```

```
    myfunction = function()
    {
        alert("Button is clicked");
    }

    var btn = document.getElementById('btn');
    btn.addEventListener('click', myfunction);
    </script>
 </body>

</html>
```

Javascript map method example with a normal function

```
<html>
    <head><title></title></head>
    <body>

    <script>

        var myarr = [20, 30, 40, 50, 60];
       var newarr = myarr.map( makeSquare );

       function makeSquare(n)
        {
            return n * 10;
        }

       console.log(newarr);

    </script>
    </body>
</html>
```

Javascript map function example with an arrow function

```
<html>
    <head><title></title></head>
    <body>

    <script>

        var myarr = [20, 30, 40, 50, 60];

       var newarr = myarr.map( (n) => n * 20 );
```

```
      console.log(newarr);

    </script>
    </body>
</html>
```

Javascript map method usage in React JS

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


const colors = ['red', 'green', 'blue', 'orange'];

const newcolors = colors.map( (c) => <p>{c}</p>);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(newcolors);
```

Basic application of react-native, using a function component

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 * @flow strict-local
 */

import React from 'react';

import {
  Text,
  View
} from 'react-native';

const App = () => {


  return (
        <View style={{flex: 1, justifyContent: "center", alignItems:
"center"}}>
          <Text>Hello world</Text>
```

```
        </View>
  );
}

export default App;
```

Example Reactive Native with core components

```
import React from 'react';

import {
  View,
  Text,
  Image,
  ScrollView,
  TextInput

} from 'react-native';

const App = () => {


  return (
        <ScrollView>
          <Text>This is some text</Text>
          <View>
            <Text>This text is placed in View</Text>
            <Image source={require("./images/flower.jpg")}
            style={{width: 300, height: 300}}

            />
          </View>

          <TextInput
          style = {{
            height: 40,
            borderColor: 'gray',
            borderWidth: 1
          }}
          defaultValue = "Default input text"
          />
        </ScrollView>
  );
}

export default App;
```

Function component example

Here we replaced App with Cat and we are returning <Text></Text> from function component.

```
import React from 'react';
import { Text } from 'react-native';

const Cat = () => {
  return (
    <Text>Hello, I am your cat!</Text>
  );
}

export default Cat;
```

Using JSX to declare a variable in react-native

```
import React from 'react';
import { Text } from 'react-native';

const Cat = () => {
  const name = "friend";

  return (
    <Text>Hello, I am your {name}</Text>
  );
}

export default Cat;
```

Using JSX and calling a function in curly braces

```
import React from 'react';
import { Text } from 'react-native';

const getFullName = (firstName, secondName, thirdName) => {
  return firstName + " " + secondName + " " + thirdName;
}

const Cat = () => {
  return (
    <Text>
      Hello, I am {getFullName("Rum", "Tum", "Tugger")}!
    </Text>
  );
}
```

```
export default Cat;
```

Custom components and nesting within each other

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

const Cat = () => {
  return (
    <View>
      <Text>Hello, I am...</Text>
      <TextInput
        style={{
          height: 40,
          borderColor: 'gray',
          borderWidth: 1
        }}
        defaultValue="Name me!"
      />
    </View>
  );
}

export default Cat;
```

Multiple components, calling components within components

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

const Cat = () => {
  return (
    <View>
      <Text>I am also a cat!</Text>
    </View>
  );
}

const Cafe = () => {
  return (
    <View>
      <Text>Welcome!</Text>
      <Cat />
      <Cat />
      <Cat />
    </View>
  );
}
```

```
export default Cafe;
```

Importing a component from an external file

App.js

```
import React from 'react';
import { Text } from 'react-native';
import Product from './Product';


const App = () => {
  return (
    <>
   <Text>This is App component</Text>
   <Product > </Product>
   </>
   );
}

export default App;
```

Product.js

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

const Product = () => {
    return(
    <View>
        <Text>This is product component</Text>
    </View>
    );
}

export default Product;
```

Using Props properties to pass values to react elements

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

```
const Cat = (props) => {
  return (
    <View>
      <Text>Hello, I am {props.name}!</Text>
    </View>
  );
}

const Cafe = () => {
  return (
    <View>
      <Cat name="Maru" />
      <Cat name="Jellylorum" />
      <Cat name="Spot" />
    </View>
  );
}

export default Cafe
```

Printing simple alert() with button

```
import React from 'react';
import { Button } from 'react-native';

const App = () => {
  return (
   <Button
   title="Click Here"
   onPress={()=>alert("hello")}
   />

   );
}

export default App;
```

Calling a simple function in button onPress event

```
import React from 'react';
import { Button } from 'react-native';

const myfunction = () =>
{
alert("hello");
```

```
}

const App = () => {
  return (
   <Button
   title="Click Here"
   onPress={() => myfunction()}
   />

   );
}

export default App;
```

Calling a simple function in button onPress event and passing an argument to the function

```
import React from 'react';
import { Button } from 'react-native';

const myfunction = (str) =>
{
alert(str);
}

const App = () => {
  return (
   <Button
   title="Click Here"
   onPress={() => myfunction("45")}
   />

   );
}

export default App;
```

Defining a function within the onPress event of a button

```
import React from 'react';
import { Button } from 'react-native';


const App = () => {
  return (
```

```
   <Button
   title="Click Here"
   onPress={
    function myfunction()
    {
      alert("hello");
    }
   }
   />

   );
}

export default App;
```

How to disable a button using Hook and useState, state variable

```
 import React, {useState} from 'react';
 import { Button, Text, View } from 'react-native';


 const Cat = (props) => {

  const [isHungry, setIsHungry] = useState(true);
  return (
    <View>
      <Text>I am {props.name}</Text>

      <Button
      onPress={ () => {
        setIsHungry(false);
      }}
      disabled = {!isHungry}
       title='Click Here' />

    </View>
  );
 }


 const App = () => {
   return (
  <>
<Cat name="first" />
</>
   );
```

```
 }

export default App;
```

Example of State variables and useState()

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

const Cat = (props) => {
  const [isHungry, setIsHungry] = useState(true);

  return (
    <View>
      <Text>
        I am {props.name}, and I am {isHungry ? "hungry" : "full"}!
      </Text>
      <Button
        onPress={() => {
          setIsHungry(false);
        }}
        disabled={!isHungry}
        title={isHungry ? props.name +  ": Pour me some milk, please!"  :
props.name +  ": Thank you!"}
      />
    </View>
  );
}

const Cafe = () => {
  return (
    <>
      <Cat name="Cat1" />
      <Cat name="Cat2" />
    </>
  );
}

export default Cafe;
```

Converting text to upper case using onChangeText and TextInput with state variable

In this example, we store `text` in the state, because it changes over time.

```
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, TextInput, View } from 'react-native';

const App = () => {

  // initialize 'text' with blank value ''
  const [text, setText] = useState('');

  return (
    <View style={styles.container}>

      <TextInput
        placeholder='Type here to translate'

      /*
      input argument is newText which is the text we are typing.
      Return argument is setText(newText) which is assigning
      newText to state variable text.
      onChangeText prop takes a function to be called every
      time the text changed, and an onSubmitEditing prop that
      takes a function to be called when the text is submitted.
       */


      onChangeText={newText => setText(newText)}
      defaultValue=''
      />

      <Text style={{fontSize: 42}} >
      {text.toUpperCase()}
      </Text>

      <StatusBar style="auto" />
    </View>
  );
}

export default App;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
```

```
});
```

Using a ScrollView

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

const App = () => {
  return (
    <ScrollView>
      <View style={styles.container}>
      <Text style={{fontSize:40}} >This is first paragraph. This is first
paragraph. This is first paragraph. This is first paragraph. This is first
paragraph. </Text>
      <Text style={{fontSize:60}}> This is second paragraph. This is second
paragraph. This is second paragraph. This is second paragraph. This is second
paragraph. </Text>
      <Text style={{fontSize: 80}}> This is third paragraph.  This is third
paragraph.  This is third paragraph.  This is third paragraph. </Text>
      </View>
    </ScrollView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
    paddingLeft: 2,
    paddingRight: 2,
    marginTop: 40
  }
});

export default App;
```

Spread syntax for arrays in javascript

```
const myArray1 = [3, 4, 5]

const myArray2 = [1, 2, ...myArray1, 6, 7]

console.log(myArray2)
```

```
const myObject1 = { name: 'Devin', hairColor: 'brown' }

const myObject2 = { ...myObject1, age: 29 }

console.log(myObject2)
```

Example of using a FlatList

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

export default function App() {
    return (
        <View>
         <FlatList
         data={
            [ {name:"Ali", age:"23"},
              {name:"noman", age:"44"},
              {name:"Faisal", age:"45"},
            ]
        }

        renderItem = {

            ({item}) =>
            <Text>Name: {item.name}, Age: {item.age}</Text>

    }
    />
        </View>

    );
}
```

Using a simple style in app

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

const App = () => {
  return (
    <View style={styles.container}>
    <Text style={styles.welcome}>Hello</Text>
```

```
    <Text style={styles.welcome}>World</Text>
    </View>
    );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },

  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 50,
  }
});


export default App;
```

How to get value of TextInput and show in alert() on button click

```
import React, { useState } from "react";
import { StyleSheet, View, Text, Button, TextInput } from 'react-native';

export default function App() {
    const [text,setText] = useState('');
    return (
        <View style={styles.maincontainer}>
            <Text style={styles.title}>How to get TextInput value on Button
Click into React Native</Text>
            <View style={styles.container}>
                <TextInput
                    style={styles.input}
                    placeholder="Enter Name"
                    onChangeText={(text) => setText(text)}
                    value={text}
                />
                <Button title="submit" onPress={() => alert(text)} />
            </View>
        </View>
    );
}

const styles = StyleSheet.create({
```

```
    maincontainer: {
        marginTop: 40,
    },
    input:{
        borderWidth:1,
        marginBottom:10,
        padding:10,
        width:'100%',
        borderRadius:10,
     },
    title: {
        backgroundColor: 'red',
        textAlign: 'center',
        padding: 10,
        fontSize: 20,
        color: '#FFFF',
        fontWeight:'bold',
    },
    container: {
        marginTop: 40,
        alignItems: 'center',
    },
});
```

Get value from TextInput by pressing go button on software keyboard

```
import React, { useState } from "react";
import { StyleSheet, View, Text, Button, TextInput } from 'react-native';

export default function App() {
    const [txtEmail, setEmail] = useState('');
    const [txtName, setName] = useState('');


    return (
        <View style={styles.maincontainer}>
            <Text style={styles.title}>How to get TextInput value on Clicking
Go button of soft keyboard</Text>
            <View style={styles.container}>

            <TextInput
                    style={styles.input}
                    placeholder="Enter email"
```

```
                    onSubmitEditing={(value) =>
setEmail(value.nativeEvent.text)}
                />


                <TextInput
                    style={styles.input}
                    placeholder="Enter Name"
                    onSubmitEditing={(value) =>
setName(value.nativeEvent.text)}
                />

                <Text>E-Mail: {txtEmail}</Text>
                <Text>Name: {txtName}</Text>

            </View>
        </View>
    );
}

const styles = StyleSheet.create({
    maincontainer: {
        marginTop: 40,
    },
    input:{
        borderWidth:1,
        marginBottom:10,
        padding:10,
        width:'100%',
        borderRadius:10,
     },
    title: {
        backgroundColor: 'red',
        textAlign: 'center',
        padding: 10,
        fontSize: 20,
        color: '#FFFF',
        fontWeight:'bold',
    },
    container: {
        marginTop: 40,
        alignItems: 'center',
    },
});
```

Assign value of one TextInput to another on button click

```
import React, { useState } from "react";
import { StyleSheet, View, Text, Button, TextInput } from 'react-native';

export default function App() {
    const [text,setText] = useState('');
    const [newText, setNewText] = useState('');

    return (
        <View style={styles.maincontainer}>
            <Text style={styles.title}>How to get TextInput value on Button
Click into React Native</Text>
            <View style={styles.container}>

            <Text> This is the first text input </Text>
                <TextInput style={styles.input} placeholder="Enter name"
                    onChangeText={(text) => setText(text)} />

                <Text> This is the second text input </Text>

                <TextInput style={styles.input}
                defaultValue = {newText}
                >


                </TextInput>
                <Button title="submit" onPress={() => setNewText(text)} />
            </View>
        </View>
    );
}

const styles = StyleSheet.create({
    maincontainer: {
        marginTop: 40,
    },
    input:{
        borderWidth:1,
        marginBottom:10,
        padding:10,
        width:'100%',
        borderRadius:10,
     },
    title: {
        backgroundColor: 'red',
        textAlign: 'center',
        padding: 10,
        fontSize: 20,
```

```
        color: '#FFFF',
        fontWeight:'bold',
    },
    container: {
        marginTop: 40,
        alignItems: 'center',
    },
});
```

Create a simple login page

```
import React, {useState} from 'react';
import { Text, View, Button, StatusBar, StyleSheet } from 'react-native';
import {TextInput} from 'react-native-paper';


export default function App() {
const [userError, setUserError] = useState(false);
  return (
    <View>

    <TextInput
    label="Username"
    left={<TextInput.Icon name="account" />}
    mode="outlined"
    style={{ margin: 10 }}
    activeUnderlineColor="green" //when this TextInput is active, change its
accent color to green
    underlineColor="purple" //when inactive, set color to purple
    error={userError}
  />


  <TextInput
    label="Email"
    left={<TextInput.Icon name="email" />}
    mode="flat"
    style={{ margin: 10 }}
    activeUnderlineColor="yellow"
    underlineColor="red"
    error={userError}
  />
  <Button title="Submit"
   onPress={() => setUserError((value) => !value)}>Press</Button>
    <StatusBar />
```

```
    </View>
  );
}
```

Get value of two InputText and print in alert

```
import React, { useState } from "react";
import { StyleSheet, View, Text, Button, TextInput } from 'react-native';


const login = (email, name) => {
  alert(email + " " + name);
}

export default function App() {
    const [txtEmail, setEmail] = useState('');
    const [txtName, setName] = useState('');



    return (
        <View style={styles.maincontainer}>
            <Text style={styles.title}>How to get TextInput value on Clicking
Go button of soft keyboard</Text>
            <View style={styles.container}>

            <TextInput
                    style={styles.input}
                    placeholder="Enter email"
                    onChangeText={email => setEmail(email)}

                />

              <TextInput
                    style={styles.input}
                    placeholder="Enter name"
                    onChangeText={name => setName(name)}

                />

              <Button title="Click Here"
              onPress={ () => login(txtEmail, txtName) }
              ></Button>


            </View>
        </View>
```

```
    );
}

const styles = StyleSheet.create({
    maincontainer: {
        marginTop: 40,
    },
    input:{
        borderWidth:1,
        marginBottom:10,
        padding:10,
        width:'100%',
        borderRadius:10,
     },
    title: {
        backgroundColor: 'red',
        textAlign: 'center',
        padding: 10,
        fontSize: 20,
        color: '#FFFF',
        fontWeight:'bold',
    },
    container: {
        marginTop: 40,
        alignItems: 'center',
    },
});
```

Sent value of two InputTexts to a function and return the value

In this program, we will send values of two InputTexts to a function. The function will do some processing and return the value that we will store in a state variable.

```
import React, { useState } from "react";
import { StyleSheet, View, Text, Button, TextInput } from 'react-native';

const login = (email, name) => {
  alert(email + " " + name);
  return "Input should be in correct format";
}

export default function App() {
    const [txtEmail, setEmail] = useState('');
    const [txtName, setName] = useState('');
    const [errMsg, seterrMsg] = useState('');
```

```
    return (
        <View style={styles.maincontainer}>
            <Text style={styles.title}>How to get TextInput value on Clicking
Go button of soft keyboard</Text>
            <View style={styles.container}>

            <TextInput
                    style={styles.input}
                    placeholder="Enter email"
                    onChangeText={email => setEmail(email)}

                />

              <TextInput
                    style={styles.input}
                    placeholder="Enter name"
                    onChangeText={name => setName(name)}

                />

              <Button title="Click Here"
              onPress={ () => seterrMsg(login(txtEmail, txtName)) }
              ></Button>

             <Text>{errMsg}</Text>
            </View>
        </View>
    );
}

const styles = StyleSheet.create({
    maincontainer: {
        marginTop: 40,
    },
    input:{
        borderWidth:1,
        marginBottom:10,
        padding:10,
        width:'100%',
        borderRadius:10,
     },
    title: {
        backgroundColor: 'red',
        textAlign: 'center',
        padding: 10,
        fontSize: 20,
        color: '#FFFF',
        fontWeight:'bold',
```

```
    },
    container: {
        marginTop: 40,
        alignItems: 'center',
    },
});
```

Defining an arrow function within an arrow function

```
import React, {useState} from 'react';
import { Text, TextInput, View, Button } from 'react-native';


const App = () => {
  const [email, setEmail] = useState('');
  const [name, setName] = useState('');
  const [response, setResponse] = useState('');

// this arrow function is called within arrow function
const Displayvalues = () => (
<View>
    <Text>{email}</Text>
    <Text>{name}</Text>
</View>
  )

  return (
   <View>
   <TextInput
   style={ {borderColor: 'black', borderWidth:2, margin: 5}}
   onChangeText={(text)=>setEmail(text)}
   >
   </TextInput>

   <TextInput
   style={ {borderColor: 'black', borderWidth:2, margin: 5}}
   onChangeText={(text)=>setName(text)}
   >
   </TextInput>

   <Button
   title="Click Here"
   onPress={()=>setResponse(Displayvalues)}
   >

   </Button>

   <View>
```

```
    {response}
   </View>
   </View>
   );
}

export default App;
```

Changing values of variables in function component called in main component

```
import React, {useState} from 'react';
import { View, Button, Text, TextInput } from 'react-native';


const App = () => {

const [myvariable, setmyvariable] = useState("pakistan");

  return (
    <>
<Text>This is main component</Text>
    <AssignData
    myvar = {myvariable}
    setvariable = {setmyvariable}
    >

<Text>Printing in App: {myvariable}</Text>
    </AssignData>

   </>
   );
  }

const AssignData = (
{
  myvar,
  setvariable,
  children
}

) => {

    return(
      <View>
```

```
        <Text>Printing in AssignData: {myvar}</Text>
        <View>{children}</View>
     <Button
     title="Click here"
     onPress={()=>setvariable('lahore')}
     >

     </Button>
     </View>

    )
    }

export default App;
```

Simple example of map function

```
<html>
    <head><title></title></head>
    <body>
        <script>
    let myarray = [10, 20, 30, 40, 50];

    myarray.map(

function myfunction(num)
{
   let ans = num * 10;
    console.log(ans);
}

    );
        </script>
    </body>
</html>
```

map function with short notation

```
<html>
    <head><title></title></head>
    <body>
        <script>
    let myarray = [10, 20, 30, 40, 50];

    myarray.map(
```

```
(num) => console.log(num * 10)

    );
        </script>
    </body>
</html>
```

map function simple example

```
<html>
    <head><title></title></head>
    <body>

    <script>
    let myarray = [10, 20, 30, 40, 50];
    var mystring = "";
    let newarray = myarray.map( (num) => num*num );

   console.log(newarray);
        </script>
    </body>
</html>
```

map function example

NOTE: First make a simple example of usage in javascript

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

const App = () => {
  let boxesarray = ["Box 1", "Box 2", "Box 3"];
  const [boxes] = useState(boxesarray);

  return (
    <View >
      {
      boxes.map(
      (value) => <Text key={value}>{value}</Text>
      )
}
</View>
      );
    }

export default App;
```

```
    </View>


   );

}

const styles = StyleSheet.create(
{
  box: {
    width: 50,
    height: 50,
  }
}

)

export default App;
```

map function to show dynamic Text

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


const App = () => {
  const [fruits] = useState([
    {name:"banana", color:"red"},
    {name:"apple", color: "blue"},
    {name:"orange", color:"green"},
    ]);

  return (
    <View>
{
  fruits.map(
(fruit) => (

<Text
style = {[styles.box, {backgroundColor:fruit.color}]}

>{fruit.name}</Text>

)
)

}
```

Example of function component

```
import React from 'react';
import { Text } from 'react-native';

const App = () => {
  return (
    <>
    <MyLayout> </MyLayout>
    <MyLayout> </MyLayout>
    <MyLayout> </MyLayout>
    </>
   );
}

const MyLayout = () =>
{
return(
<Text style={{width:50, height:50, backgroundColor: 'red'}}>Hello</Text>

);
}
export default App;
```

Function component without using return keyword

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

const App = () => {
  return (
    <>
    <MyLayout> </MyLayout>

    </>
   );
}

const MyLayout = () =>
(
<View>
  <Text style={{width:50, height:50, backgroundColor: 'red'}}>Hello</Text>
  <Text style={{width:50, height:50, backgroundColor: 'blue'}}>Hello</Text>
</View>
)
```

```
export default App;
```

Passing children to function component

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

const App = () => {
  return (

    <MyLayout>
      <View>
    <Text style={{width:50, height:50, backgroundColor: 'red'}}>Box1</Text>
    <Text style={{width:50, height:50, backgroundColor: 'green'}}>Box2</Text>
    <Text style={{width:50, height:50, backgroundColor: 'blue'}}>Box3</Text>
    </View>
    </MyLayout>
   );
}

const MyLayout = (
  { children }
) =>
  (
<View>
{children}
</View>

)
export default App;
```

Passing arguments to function component

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


const App = () => {
  const [myvariable] = useState("abc");


  return (

   <MyLayout
   myvar = {myvariable}
```

```
   country = "Pakistan"
   >
<View>
    <Text
     style = {{width:50, height:50, backgroundColor: 'green'}}
     >
       BOX1
       </Text>
     <Text
     style = {{width:50, height:50, backgroundColor: 'blue'}}
     >
       BOX2
       </Text>
       </View>
    </MyLayout>

   );
}

const MyLayout = (
{
  children,
  myvar,
  country
}
) =>
  (
    <View>
      {children}
      <Text>{myvar}</Text>
      <Text>{country}</Text>
    </View>
  )


export default App;
```

Passing object to function component

```
import React, { useState } from 'react';
import { Text, StyleSheet, View, TouchableOpacity, Button } from 'react-
native';


const App = () => {
```

```
  const [powderblue, setPowderblue] = useState({flexGrow: 0, flexShrink: 1,
flexBasis: "auto", });

  return (
    <View>
  <BoxInfo color="powderblue" {...powderblue} setStyle={setPowderblue} >
  </BoxInfo>

   </View>
   );
}


const BoxInfo = ({
  color,
  flexBasis,
  flexShrink,
  flexGrow,
  setStyle,
}) => (
  <View>
    <Text>{color}</Text>
    <Text>{flexBasis}</Text>
    <Text>{flexShrink}</Text>
    <Text>{flexGrow}</Text>
    </View>)

export default App;
```

Passings arrays as arguments to function components

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


const App = () => {

  const [veges] = useState(["carrot", "radish", "turnip"]);

  return (
    <>
  <MyLayout
  country="Pakistan"
  city="Abbottabad"
  fruits={["apple", "banana", "pear"]}
  vegetables = {veges}
  >
```

```
  <View>
  <Text style={{width:50, height:50, backgroundColor:'blue'}}>
      Hello
    </Text>
    <Text style={{width:50, height:50, backgroundColor:'green'}}>
      Hello
    </Text>
  </View>

  </MyLayout>

  </>
   );
}

const MyLayout = (
{
  country,
  city,
  fruits,
  vegetables
}
) =>
(
  <View>

<Text>{country}</Text>
<Text>{city}</Text>

{
  fruits.map( (fruit) => <Text>{fruit}</Text>)
}

{
  vegetables.map( (veg) => <Text>{veg}</Text>)
}



  </View>
  )



export default App;
```

Applying style conditionally

```
import React, {useState} from 'react';
import { Text, StyleSheet } from 'react-native';

const App = () => {
  const [str] = useState("red");
  return (
    <>
  <Text
  style={ [styles.box, str==="green" && styles.font,
 {backgroundColor:'blue'}]}>
      Hello
   </Text>

   <Text
   style={ [styles.box,  str==="red" && styles.font,
 {backgroundColor:'green'}]}>
      World
    </Text>
  </>
   );
}

const styles = StyleSheet.create(
{
  box:{
    width: 100,
    height: 100,
  },

  font:{
    fontWeight:'bold',
    fontSize: 30,

  }
}

);

export default App;
```

Applying multiple style classes conditionally

```
import React, {useState} from 'react';
import { Text, StyleSheet } from 'react-native';

const App = () => {
```

```
  const [names] = useState(["Ali", "Noman", "Faisal", "Javed"]);
  const [person] = useState("Faisal");

  return (
    <>
    {
      names.map(

(name) => (
  <Text
  key={name}
  style={ [styles.box, name===person && [styles.font, styles.coloring],
 {backgroundColor:'blue'}]}>
      {name}
   </Text>
   )
   )
  }

  </>
   );
}

const styles = StyleSheet.create(
{
  box:{
    width: 100,
    height: 100,
    margin: 3,
  },

  font:{
    fontWeight:'bold',
    fontSize: 30,

  },
  coloring: {
    color:'red',
  },
}

);

export default App;
```

Multiple conditions and multiple style classes

```
import React, { useState } from 'react';
import { Text, StyleSheet } from 'react-native';

const App = () => {
  const [names] = useState(
    [
      {name:"Ali", number:10},
      {name: "Noman", number:50},
      {name: "Faisal", number:40}]);

  const [person] = useState("Noman");
  const [num] = useState(150);

  return (
    <>
   {
    names.map( (obj) =>
    (<Text
    key={obj.name}
    style={[styles.box,
      obj.name===person &&
      obj.number==num && [styles.font, styles.fontcolor]]}
    >{obj.name}</Text>))
   }

   </>
   );
}

const styles = StyleSheet.create(
{
  box:{
    width: 200,
    height: 50,
  },

  font:{
    fontWeight:'bold',
    fontSize:30,
  },
  fontcolor:{
    color: 'red'

  }
}

)
```

```
export default App;
```

Change the style property dynamically

```
import React, {useState} from 'react';
import { Text, TextInput, View, Button } from 'react-native';


const App = () => {
  const [property, setProperty] = useState('backgroundColor');
  const [propertyvalue, setPropertyValue] = useState('red');

  return (
   <View>
   <TextInput
   style={ {[property]:propertyvalue, borderWidth:2, margin: 5}}
   >
   </TextInput>



   <Button
   title="Set Border color propery"
   onPress={()=>
        {
            setProperty('width');
            setPropertyValue(100);
        }
    }
   >
  </Button>

  <Button

   title="Set Background color propery"
   onPress={()=>
    {
        setProperty('backgroundColor');
        setPropertyValue('red');
    }
}
   >
  </Button>


   </View>
   );
```

```
}

export default App;
```

Simple example of class component

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

class Person extends Component {
  state = {
    email: 'ali@gmail.com',
    name: 'Ali Khan'
  }

  render() {

    return(
      <View>
<Text>{this.state.email}</Text>
<Text>{this.state.name}</Text>
<Button
title = 'Click Here'
onPress = {() => this.setState({name: 'Shahid'})}
></Button>

      </View>
    )


  }
}


export default Person;
```

Example of class component

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

class Inputs extends Component {
   state = {
      email: '',
      password: '',
```

```
      msg: ''
   }



   login = (email, pass) => {
      alert('email: ' + email + ' password: ' + pass)
      this.setState({ msg: 'The input is incorrect' })
   }
   render() {
      return (
         <View style = {styles.container}>
            <TextInput style = {styles.input}
               underlineColorAndroid = "transparent"
               placeholder = "Email"
               placeholderTextColor = "#9a73ef"
               autoCapitalize = "none"
               onChangeText = { (text) => this.setState( {email: text} )  }/>

            <TextInput style = {styles.input}
               underlineColorAndroid = "transparent"
               placeholder = "Password"
               placeholderTextColor = "#9a73ef"
               autoCapitalize = "none"
               onChangeText = { (text) => this.setState( {password: text}) }/>

            <TouchableOpacity
               style = {styles.submitButton}
               onPress = { () => this.login(this.state.email,
this.state.password) }>
               <Text style = {styles.submitButtonText}> Submit </Text>

               <Text style = {styles.errorMsg}>{this.state.msg}</Text>
            </TouchableOpacity>
         </View>
      )
   }
}
export default Inputs

const styles = StyleSheet.create({
   container: {
      paddingTop: 23
   },
   input: {
      margin: 15,
      height: 40,
      borderColor: '#7a42f4',
      borderWidth: 1
```

```
   },
   submitButton: {
      backgroundColor: '#7a42f4',
      padding: 10,
      margin: 15,
      height: 40,
   },
   submitButtonText:{
      color: 'white'
   },
   errorMsg: {
margin:10,
height:30
   }
})
```

Touchable Opacity Example

This component fades out when pressed, and fades back in when released. We can style it however we want, just like a `View`.
We can configure the pressed opacity with the `activeOpacity` prop.
This is typically the most common kind of button in a React Native app.

```
import React, { useState } from 'react'
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native'

export default function App() {
  const [count, setCount] = useState(0)

  return (
    <View style={styles.container}>
      <TouchableOpacity
        style={styles.button}
        activeOpacity={0.7}
        onPress={() => {
          setCount(count + 1)
        }}
      >
        <Text style={styles.text}>Press me!</Text>
      </TouchableOpacity>
      <Text style={styles.text}>{`Pressed ${count} times`}</Text>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
```

```
    alignItems: 'center',
    justifyContent: 'center',
  },
  button: {
    padding: 40,
    borderRadius: 4,
    borderWidth: 1,
    borderColor: 'green',
    backgroundColor: 'lightgreen',
  },
  text: {
    fontSize: 18,
    padding: 12,
  },
})
```

Touchable Highlight example

This component changes color when pressed, and changes back in when released. We can configure the color with the `underlayColor` prop.

```
import React, { useState } from 'react'
import { StyleSheet, Text, TouchableHighlight, View } from 'react-native'

export default function App() {
  const [count, setCount] = useState(0)

  return (
    <View style={styles.container}>
      <TouchableHighlight
        style={styles.button}
        underlayColor="#FAB"
        onPress={() => {
          setCount(count + 1)
        }}
      >
        <Text style={styles.text}>Press me!</Text>
      </TouchableHighlight>
      <Text style={styles.text}>{`Pressed ${count} times`}</Text>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
```

```
    justifyContent: 'center',
  },
  button: {
    padding: 40,
    borderRadius: 4,
    backgroundColor: '#F88',
  },
  text: {
    fontSize: 18,
    padding: 12,
  },
})
```

Flex example

In the following example, the red, yellow, and green views are all children in the container view that has flex: 1 set. The red view uses flex: 1 , the yellow view uses flex: 2, and the green view uses flex: 3 . 1+2+3 = 6, which means that the red view will get 1/6 of the space, the yellow 2/6 of the space, and the green 3/6 of the space.

NOTE: flexDirection, justifyContent, alignItems, are always used on parent element, they won't work on child element. To stretch an element, we will use alignSelf: "stretch" property

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

const Flex = () => {
  return (
    <View style={[styles.container, {
      // Try setting `flexDirection` to `"row"`.
      flexDirection: "column"
    }]}>
      <View style={{ flex: 1, backgroundColor: "red" }} />

      <View style={{ flex: 2, backgroundColor: "darkorange" }} />

      <View style={{ flex: 3, backgroundColor: "green" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
```

```
    padding: 20,
  },
});

export default Flex;
```

flexDirection: “column” with flexWrap: ‘nowrap’

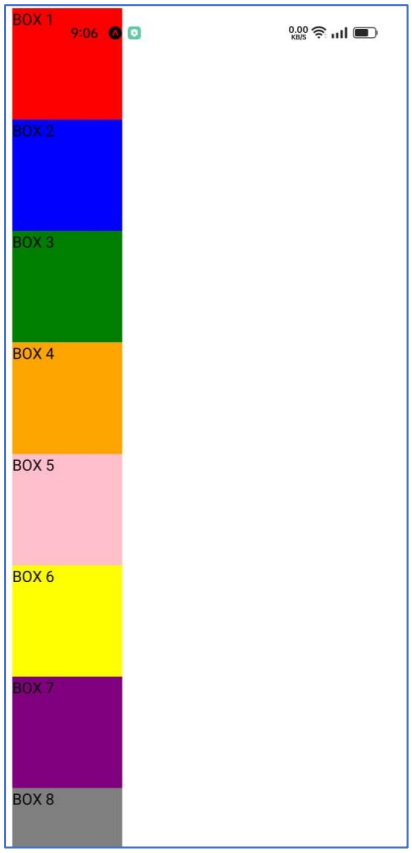
(Run this example on your mobile or emulator to see correct output)

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box1}>BOX 1</Text>
      <Text style={styles.box2}>BOX 2</Text>
      <Text style={styles.box3}>BOX 3</Text>
      <Text style={styles.box4}>BOX 4</Text>
      <Text style={styles.box5}>BOX 5</Text>
      <Text style={styles.box6}>BOX 6</Text>
      <Text style={styles.box7}>BOX 7</Text>
      <Text style={styles.box8}>BOX 8</Text>
      <Text style={styles.box9}>BOX 9</Text>
      <Text style={styles.box10}>BOX 10</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 40,
    flex: 1,
    padding: 1,
    flexDirection: 'column',
    flexWrap: 'nowrap',
    // alignItems: 'center', (works with primary axis)
    // justifyContent: 'center', (works with secondary axis)
  },

  box1: {
    backgroundColor: 'red',
    height: 100,
    width: 100,
    //align-self: flex-start or flex-end;
  },
  box2: {
    backgroundColor: 'blue',
```

```
    height: 100,
    width: 100,
  },

  box3: {
    backgroundColor: 'green',
    height: 100,
    width: 100,
  },
  box4: {
    backgroundColor: 'orange',
    height: 100,
    width: 100,
  },
box5: {
    backgroundColor: 'pink',
    height: 100,
    width: 100,
  },

box6: {
    backgroundColor: 'yellow',
    height: 100,
    width: 100,
  },

  box7: {
    backgroundColor: 'purple',
    height: 100,
    width: 100,
  },

  box8: {
    backgroundColor: 'gray',
    height: 100,
    width: 100,
  },

box9: {
    backgroundColor: 'lightblue',
    height: 100,
    width: 100,
  },

box10: {
    backgroundColor: 'magenta',
    height: 100,
    width: 100,
```

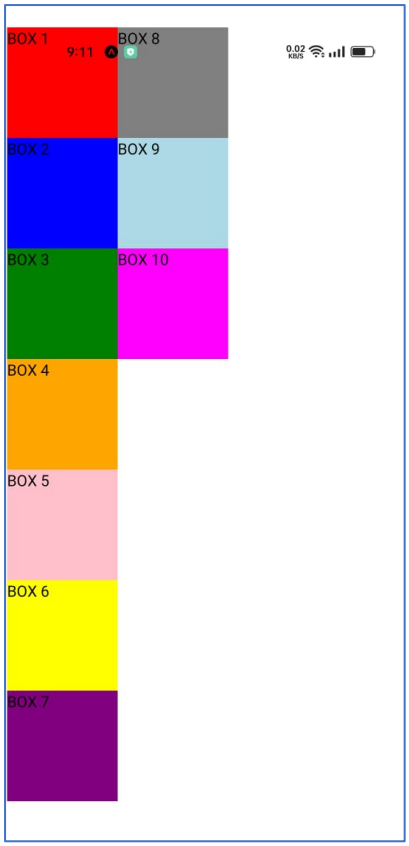
```
  },
});

export default App;
```

Output:



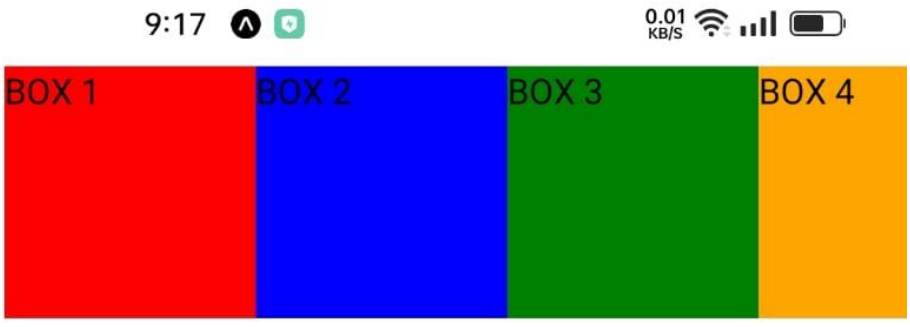
The following will be the output with

```
flexWrap: 'wrap',
```



The following will be the output with:

```
flexDirection: 'row',
    flexWrap: 'nowrap',
```



The following will be the output with:

```
    flexDirection: 'row',
    flexWrap: 'wrap',
```

Layout direction example

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box1}>BOX 1</Text>
      <Text style={styles.box2}>BOX 2</Text>
      <Text style={styles.box3}>BOX 3</Text>
      <Text style={styles.box4}>BOX 4</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 40,
    flex: 1,
    padding: 1,
    flexDirection: 'column',
    direction: 'rtl',
    flexWrap: 'wrap',
    // alignItems: 'center', (works with primary axis)
    // justifyContent: 'center', (works with secondary axis)
  },

  box1: {
```

```
    backgroundColor: 'red',
    height: 100,
    width: 100,
    //align-self: flex-start or flex-end;
  },
  box2: {
    backgroundColor: 'blue',
    height: 100,
    width: 100,
  },

  box3: {
    backgroundColor: 'green',
    height: 100,
    width: 100,
  },
  box4: {
    backgroundColor: 'orange',
    height: 100,
    width: 100,
  },
  /*
box5: {
    backgroundColor: 'pink',
    height: 100,
    width: 100,
  },

box6: {
    backgroundColor: 'yellow',
    height: 100,
    width: 100,
  },

  box7: {
    backgroundColor: 'purple',
    height: 100,
    width: 100,
  },

  box8: {
    backgroundColor: 'gray',
    height: 100,
    width: 100,
  },

box9: {
    backgroundColor: 'lightblue',
```

```
    height: 100,
    width: 100,
  },

box10: {
    backgroundColor: 'magenta',
    height: 100,
    width: 100,
  },
  */
});

export default App;
```

Flexbox justifyContent property

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

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box1}>BOX 1</Text>
      <Text style={styles.box2}>BOX 2</Text>
      <Text style={styles.box3}>BOX 3</Text>
      <Text style={styles.box4}>BOX 4</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 40,
    flex: 1,
    padding: 1,
    flexDirection: 'column',
    flexWrap: 'nowrap',
    // alignItems: 'center', (works with primary axis)
    justifyContent: 'center', // (works with secondary axis)
  },

  box1: {
    backgroundColor: 'red',
    height: 100,
    width: 100,
    //align-self: flex-start or flex-end;
  },
```

```
  box2: {
    backgroundColor: 'blue',
    height: 100,
    width: 100,
  },

  box3: {
    backgroundColor: 'green',
    height: 100,
    width: 100,
  },
  box4: {
    backgroundColor: 'orange',
    height: 100,
    width: 100,
  },
});

export default App;
```

Flexbox Justify content all options example

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

const JustifyContentBasics = () => {


const [label] = useState("justifyContent");
const [selectedValue, setSelectedValue] = useState("flex-start");


const [values] = useState([
  "flex-start",
  "flex-end",
  "center",
  "space-between",
  "space-around",
  "space-evenly",
]);

  return (

    <View style={{ padding: 10, flex: 1 }}>

    <Text style={styles.label}>{label}</Text>
```

```
    <View style={styles.row}>

      { values.map((value) => (
      <TouchableOpacity
      style={[styles.button, selectedValue === value && styles.selected ]}
      key={value}
      onPress={()=>setSelectedValue(value)}
      >
        <Text
        style={[styles.buttonLabel, selectedValue === value &&
styles.selectedLabel]}
        >{value}</Text>
     </TouchableOpacity>
      ))}


    </View>

    <View style={[styles.container, { [label]: selectedValue }]}>

      <View
        style={[styles.box, { backgroundColor: "powderblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "skyblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "steelblue" }]}
      />
    </View>

  </View>

  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 10,
    backgroundColor: "aliceblue",
  },
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
```

```
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",

  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default JustifyContentBasics;
```

Flexbox alignItems property

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

const AlignItemsLayout = () => {
  const [alignItems, setAlignItems] = useState("stretch");

  return (
```

```
    <PreviewLayout
      label="alignItems"
      selectedValue={alignItems}
      values={[
        "stretch",
        "flex-start",
        "flex-end",
        "center",
        "baseline",
      ]}
      setSelectedValue={setAlignItems}
    >
      <View
        style={[styles.box, { backgroundColor: "powderblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "skyblue" }]}
      />
      <View
        style={[
          styles.box,
          {
            backgroundColor: "steelblue",
            width: "auto",
            minWidth: 50,
          },
        ]}
      />
    </PreviewLayout>
  );
};

const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[
            styles.button,
```

```
            selectedValue === value && styles.selected,
          ]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value &&
                styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View
      style={[
        styles.container,
        { [label]: selectedValue },
      ]}
    >
      {children}
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
    minHeight: 200,
  },
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
```

```
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default AlignItemsLayout;
```

Flexbox alignSelf property

you can apply this property to a single child to change its alignment within its parent. alignSelf overrides any option set by the parent with alignItems.

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

const AlignSelfBasics = () => {
  const [alignSelf, setAlignSelf] = useState("stretch");

  return (
    <PreviewLayout
      label="alignSelf"
      selectedValue={alignSelf}
      values={[
        "stretch",
        "flex-start",
        "flex-end",
        "center",
        "baseline"
      ]}
      setSelectedValue={setAlignSelf}
    >
```

```
      <View
        style={[styles.box,
            {
                alignSelf,
                width: "auto",
                minWidth: 50,
                backgroundColor: "powderblue"
            }]}
      />
      <View
        style={[styles.box, { backgroundColor: "skyblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "steelblue" }]}
      />
    </PreviewLayout>
  );
};

const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[styles.button, selectedValue === value && styles.selected]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value && styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View style={styles.container}>
      {children}
```

```
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
  },
  box: {
    width: 50,
    height: 50,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});
```

```
export default AlignSelfBasics;
```

Align Content

alignContent defines the distribution of lines along the cross-axis. This only has effect when items are wrapped to multiple lines using flexWrap.

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

const AlignContentLayout = () => {
  const [alignContent, setAlignContent] = useState("flex-start");

  return (
    <PreviewLayout
      label="alignContent"
      selectedValue={alignContent}
      values={[
        "flex-start",
        "flex-end",
        "stretch",
        "center",
        "space-between",
        "space-around",
      ]}
      setSelectedValue={setAlignContent}>
      <View
        style={[styles.box, { backgroundColor: "orangered" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "orange" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumseagreen" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "deepskyblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumturquoise" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumslateblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "purple" }]}
      />
    </PreviewLayout>
  );
```

```
};

const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[
            styles.button,
            selectedValue === value && styles.selected,
          ]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value &&
                styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View
      style={[
        styles.container,
        { [label]: selectedValue },
      ]}
    >
      {children}
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexWrap: "wrap",
```

```
    marginTop: 8,
    backgroundColor: "aliceblue",
    maxHeight: 400,
  },
  box: {
    width: 50,
    height: 80,
  },
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    alignSelf: "flex-start",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default AlignContentLayout;
```

Flex Wrap

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

```
const FlexWrapLayout = () => {
  const [flexWrap, setFlexWrap] = useState("wrap");

  return (
    <PreviewLayout
      label="flexWrap"
      selectedValue={flexWrap}
      values={["wrap", "nowrap"]}
      setSelectedValue={setFlexWrap}>
      <View
        style={[styles.box, { backgroundColor: "orangered" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "orange" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumseagreen" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "deepskyblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumturquoise" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "mediumslateblue" }]}
      />
      <View
        style={[styles.box, { backgroundColor: "purple" }]}
      />

<View
        style={[styles.box, { backgroundColor: "red" }]}
      />

<View
        style={[styles.box, { backgroundColor: "green" }]}
      />


    </PreviewLayout>
  );
};

const PreviewLayout = ({
  label,
  children,
  values,
  selectedValue,
```

```
  setSelectedValue,
}) => (
  <View style={{ padding: 10, flex: 1 }}>
    <Text style={styles.label}>{label}</Text>
    <View style={styles.row}>
      {values.map((value) => (
        <TouchableOpacity
          key={value}
          onPress={() => setSelectedValue(value)}
          style={[
            styles.button,
            selectedValue === value && styles.selected,
          ]}
        >
          <Text
            style={[
              styles.buttonLabel,
              selectedValue === value &&
                styles.selectedLabel,
            ]}
          >
            {value}
          </Text>
        </TouchableOpacity>
      ))}
    </View>
    <View
      style={[
        styles.container,
        { [label]: selectedValue },
      ]}
    >
      {children}
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 8,
    backgroundColor: "aliceblue",
    maxHeight: 400,
  },
  box: {
    width: 50,
    height: 80,
  },
```

```
  row: {
    flexDirection: "row",
    flexWrap: "wrap",
  },
  button: {
    paddingHorizontal: 8,
    paddingVertical: 6,
    borderRadius: 4,
    backgroundColor: "oldlace",
    marginHorizontal: "1%",
    marginBottom: 6,
    minWidth: "48%",
    textAlign: "center",
  },
  selected: {
    backgroundColor: "coral",
    borderWidth: 0,
  },
  buttonLabel: {
    fontSize: 12,
    fontWeight: "500",
    color: "coral",
  },
  selectedLabel: {
    color: "white",
  },
  label: {
    textAlign: "center",
    marginBottom: 10,
    fontSize: 24,
  },
});

export default FlexWrapLayout;
```

flexGrow, flexShrink, and flexBasis properties

flexBasis sets the initial width of the layout.

When we define flexGrow, the layout size will grow as the screen size increases

When we define flexShrink, the layout size will shrink as the screen size decreases

To see impact of other properties, uncomment in the below example, and test your application in online snack web interface to see the impact of changing screen size.

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

```
const App = () => {

  return (

      <View style={styles.content} >
        <View
          style={[
            styles.box,
            {
          //  flexGrow: 1,
          //  flexShrink: 1,
            flexBasis: 100, // set the base width of an element

              backgroundColor: "red",
            },
          ]}
        />
        <View
          style={[
            styles.box,
            {
          // flexGrow: 1,
          //  flexShrink: 1,
            flexBasis: 100,

              backgroundColor: "blue",
            },
          ]}
        />
        <View
          style={[
            styles.box,
            {
          // flexGrow: 1,
          //  flexShrink: 1,
            flexBasis: 100,


              backgroundColor: "green",
            },
          ]}
        />
      </View>

  );
};
```

```
const styles = StyleSheet.create({

  content: {
    flexDirection: "row",
  },

  box: {
    height: 50,
    width: 50,
  },

});

export default App;
```

Relative Layout in Flex

The relative layout is the default layout. Each new layout item is assigned position in relation to the existing layout item. For example, views will be placed in top down order. However, we to add an offset in the position of next item, we can use top, left, right, bottom properties.

In this example, the lower view is given an offset of 10 pixels in relation to the existing view.

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

const PositionLayout = () => {


  return (
<View style={{backgroundColor: 'lightblue'}} >
      <View
        style={[ styles.box,  {backgroundColor: "red",
          },
        ]}
      >
      </View>

      <View
        style={[styles.box, {top: 10, backgroundColor: "green",},
        ]}
      >
      </View>


 </View>
```

```
  );
};


const styles = StyleSheet.create({

  box: {
    width: 50,
    height: 50,
  },

});

export default PositionLayout;
```

Absolute Layout in Flex

When positioned absolutely, an element doesn't take part in the normal layout flow. It is instead laid out independent of its siblings. The position is determined based on the top, right, bottom, and left values.

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

const PositionLayout = () => {

  return (
<View style={{backgroundColor: 'lightblue'}} >
      <View
        style={[ styles.box,  {backgroundColor: "red", position: "absolute" },
        ]}
      >
      </View>

      <View
        style={[styles.box, {backgroundColor: "green", top:60, left: 60,
position: "absolute"},
        ]}
      >
      </View>


 </View>

  );
};
```

```
const styles = StyleSheet.create({

  box: {
    width: 50,
    height: 50,
  },

});

export default PositionLayout;
```

Creating a grid using Flex layout

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

const Square = ({ text }) => (
  <View style={styles.square}>
    <Text style={styles.text}>{text}</Text>
  </View>
);

export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.row}>
        <Square text="A" />
        <Square text="B" />
        <Square text="C" />
      </View>
      <View style={styles.row}>
        <Square text="D" />
        <Square text="E" />
        <Square text="F" />
      </View>
      <View style={styles.row}>
        <Square text="G" />
        <Square text="H" />
        <Square text="I" />
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
```

```
    backgroundColor: "#7CA1B4",
    alignItems: "center",
    justifyContent: "center",
  },
  row: {
    flexDirection: "row",
  },
  square: {
    borderColor: "#fff",
    borderWidth: 1,
    width: 100,
    height: 100,
    justifyContent: "center",
    alignItems: "center",
  },
  text: {
    color: "#fff",
    fontSize: 18,
    fontWeight: "bold",
  },
});
```

useEffect example

```
import React, { useState, useEffect } from 'react';
import { Button, View, Text } from 'react-native';

const App = () =>{
  const [count, setCount] = useState(0);
  const [myvariable, setmyVariable] = useState(count);

/* the square bracket parameter is optional. If removed, the useeffect
will always render and there won't be any skipping.

if we want the rendering dependent on change of certain variables' values,
then we can add those variables in square brackets.
*/
  useEffect(() => {
    console.warn("MSG: ", count);

  }, [myvariable]);

  return (
    <View>

      <Button
      title="Click 1"
      onPress={() => setCount(count+1)}>
```

```
        Click me
      </Button>


      <Button
      title="Click 2"
      onPress={() => setmyVariable("hello")}>
        Click me
      </Button>
    </View>
  );
}

export default App;
```

Simple example of react-native navigation

For installation, visit this site: https://reactnative.dev/docs/navigation

```
import * as React from 'react';
import {View, Text, Button} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>

      <Stack.Navigator>

        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />

        <Stack.Screen
        name="Profile"
        component={ProfileScreen}
        />

      </Stack.Navigator>

    </NavigationContainer>
```

```
  );
};


const HomeScreen = ( {navigation} ) => {
  return (

    <Button
      title="GO to profile page"
      onPress={() => navigation.navigate('Profile')
      }
    />

  );
};

const ProfileScreen = ({ navigation }) => {
  return (
  <Text> This is profile page </Text>
  );
}

export default App;
```

Passing values from home screen to new screen

```
import * as React from 'react';
import {View, Text, Button} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>

      <Stack.Navigator>

        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />

        <Stack.Screen
        name="Profile"
        component={ProfileScreen}
```

```
        />

      </Stack.Navigator>

    </NavigationContainer>
  );
};


const HomeScreen = ( {navigation} ) => {
  return (

    <Button
      title="GO to profile page"
      onPress={() => navigation.navigate('Profile', {name: "Akhyar", age: 9})
      }
    />

  );
};

const ProfileScreen = ({ navigation, route }) => {
  return (
    <>
  <Text> This is profile page </Text>
  <Text>Name: {route.params.name}, Age: {route.params.age}</Text>

  </>
  );
}

export default App;
```

Javascript Optional Chaining example

The **optional chaining** operator (`?.`) accesses an object's property or calls a function. If the object is `undefined` or `null`, it returns `undefined` instead of throwing an error.

Syntax

```
obj.val?.prop
obj.val?.[expr]
obj.func?.(args)
```

For example, consider an object `obj` which has a nested structure. Without optional chaining, looking up a deeply-nested subproperty requires validating the references in between, such as:

```
const nestedProp = obj.first && obj.first.second;
```

The value of `obj.first` is confirmed to be non-`null` (and non-`undefined`) before then accessing the value of `obj.first.second`. This prevents the error that would occur if you accessed `obj.first.second` directly without testing `obj.first`.

With the optional chaining operator (`?.`), however, you don't have to explicitly test and short-circuit based on the state of `obj.first` before trying to access `obj.first.second`:

```
const nestedProp = obj.first?.second;
```

```
<html>
    <head>
        <title></title>
    </head>
<body>
    <script>
      const adventurer = {
  name: 'Alice',
  cat: {
    name: 'Dinah'
  }
};

const dogName = adventurer.dog?.name;
console.log(dogName);
// expected output: undefined

// if we write like below, this will give error
//const dogName = adventurer.dog.name;
//console.log(dogName);

console.log(adventurer.someNonExistentMethod?.());
// expected output: undefined

    </script>


</body>
</html>
```

Optional chaining operator use in if else statement

```
<html>
    <head>
        <title></title>
    </head>
<body>
    <script>
        const person = {
            name: 'Ali',
            address: {city: 'Karachi',
            area: {town: 'abc'}},

        }

 //console.log(person.add.city);

 if(person.add?.city)
 {
    console.log("this is if part");
 }
 else
 {
    console.log("This is else part");
 }




    </script>


</body>
</html>
```

Passing parameters to previous screen

A modal displays content that temporarily blocks interactions with the main view.

A modal is like a popup — it's not part of your primary navigation flow — it usually has a different transition, a different way to dismiss it, and is intended to focus on one particular piece of content or interaction.

```
import * as React from 'react';
import { Text, TextInput, View, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation, route }) {
  React.useEffect(() => {
    if (route.params?.newvalue) {
      // Post updated, do something with `route.params.post`
      // For example, send the post to the server
    }
  }, [route.params?.newvalue]);

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Button
        title="Create post"
        onPress={() => navigation.navigate('CreatePost')}
      />
      <Text style={{ margin: 10 }}>Post: {route.params?.newvalue}</Text>
    </View>
  );
}

function CreatePostScreen({ navigation, route }) {
  const [postText, setPostText] = React.useState('');

  return (
    <>
      <TextInput
        multiline
        placeholder="What's on your mind?"
        style={{ height: 200, padding: 10, backgroundColor: 'white' }}
        value={postText}
        onChangeText={setPostText}
      />
      <Button
        title="Done"
        onPress={() => {
          // Pass and merge params back to home screen
          navigation.navigate({
            name: 'Home',
            params: { newvalue: postText },
            merge: true,
          });
        }}
      />
    </>
```

```
  );
}

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator mode="modal">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="CreatePost" component={CreatePostScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Going Back in react navigation

Sometimes you'll want to be able to programmatically trigger this behavior, and for that you can use navigation.goBack();.

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.push('Details')}
      />
```

```
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')}
/>
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Going back multiple screens

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
```

```
        onPress={() => navigation.push('Details')}
      />
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')}
/>
      <Button title="Go back" onPress={() => navigation.goBack()} />
      <Button
        title="Go back to first screen in stack"
        onPress={() => navigation.popToTop()}
      />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Updating params using navigation.setParams()

initialRouteName - Sets the default screen of the stack. Must match one of the keys in route configs.

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>This is the home screen of the app</Text>
      <Button
        onPress={() =>
          navigation.navigate('Profile', {
```

```
            friends: ['Brent', 'Satya', 'Michaś'],
            title: "Brent's Profile",
          })
        }
        title="Go to Brent's profile"
      />
    </View>
  );
}

function ProfileScreen({ navigation, route }) {

  //console.warn("TITLE: ",route.params.title, "CITY: ", route.params.city);

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile Screen</Text>
      <Text>Friends: </Text>
      <Text>{route.params.friends[0]}</Text>
      <Text>{route.params.friends[1]}</Text>
      <Text>{route.params.friends[2]}</Text>
      <Button
        onPress={() =>
          navigation.setParams({
            friends:
              route.params.friends[0] === 'Brent'
                ? ['Wojciech', 'Szymon', 'Jakub']
                : ['Brent', 'Satya', 'Michaś'],
            title:
              route.params.title === "Brent's Profile"
                ? "Lucy's Profile"
                : "Brent's Profile",

          })
        }
        title="Swap title and friends"
      />
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
```

```
        <Stack.Screen
        name="Home"
        component={HomeScreen}>
        </Stack.Screen>

        <Stack.Screen
          name="Profile"
          component={ProfileScreen}
          //if we want to update the title
          options={ ({route})=>({title: route.params.title})}
          >
        </Stack.Navigator>
      </NavigationContainer>
  );
}

export default App;
```

Navigation setOptions() method

Using setOptions() method to change title of a screen.

```
import * as React from 'react';
import {View, Text, Button} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>

      <Stack.Navigator>

        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />

        <Stack.Screen
```

```
        name="Profile"
        component={ProfileScreen}
        />

      </Stack.Navigator>

    </NavigationContainer>
  );
};

const HomeScreen = ( {navigation} ) => {
  return (

    <Button
      title="Go to profile page"
      onPress={() => navigation.navigate('Profile')
      }
    />

  );
};

const ProfileScreen = ({ navigation }) => {
  return (
    <View>
  <Text> This is profile page </Text>
  <Button
  title="Click Here"
  onPress={()=>navigation.setOptions({title: "New title"})}
  />
  </View>
  );
}

export default App;
```

Using setOptions in combination of useEffect

```
import * as React from 'react';
import { Button, View, Text, TextInput } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation: { navigate } }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>This is the home screen of the app</Text>
```

```
      <Button
        onPress={() => navigate('Profile', { title: "Brent's profile" })}
        title="Go to Profile"
      />
    </View>
  );
}

function ProfileScreen({ navigation, route }) {
  const [value, onChangeText] = React.useState(route.params.title);

  React.useEffect(() => {
    navigation.setOptions({
      title: value === '' ? 'No title' : value,
    });
  }, [navigation, value]);

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <TextInput
        style={{ height: 40, borderColor: 'gray', borderWidth: 1 }}
        onChangeText={(text) => onChangeText(text)}
        value={value}
      />
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home"
        component={HomeScreen} />

        <Stack.Screen
          name="Profile"
          component={ProfileScreen}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Initial params

You can also pass some initial params to a screen. If you didn't specify any params when navigating to this screen, the initial params will be used. They are also shallow merged with any params that you pass. Initial params can be specified with an `initialParams` prop:

```
<Stack.Screen
  name="Details"
  component={DetailsScreen}
  initialParams={{ itemId: 42 }}
/>
```

Example:

```
import * as React from 'react';
import {View, Text, Button} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>

      <Stack.Navigator>

        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />

        <Stack.Screen
        name="Profile"
        component={ProfileScreen}
        initialParams={{itemId: 42}}
        />

      </Stack.Navigator>

    </NavigationContainer>
  );
```

```
};

const HomeScreen = ( {navigation} ) => {
  return (

    <Button
      title="GO to profile page"
      onPress={() => navigation.navigate('Profile')}
    />

  );
};

const ProfileScreen = ({ navigation, route }) => {
  return (
    <>
  <Text> This is profile page </Text>
  <Text>{route.params.itemId}</Text>

  </>
  );
}

export default App;
```

Setting title of screens manually

We use options prop.

```
import * as React from 'react';

import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Profile"
        onPress={() =>
          navigation.navigate('Profile')
        }
      />
    </View>
```

```
  );
}

function ProfileScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile screen</Text>
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Home Screen' }}
        />
        <Stack.Screen
          name="Profile"
          component={ProfileScreen}
          options={{title: 'Profile Screen'}}

        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Change title of screen dynamically

```
import * as React from 'react';
import {View, Text, Button} from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
```

```
    <NavigationContainer>

      <Stack.Navigator>

        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />

        <Stack.Screen
        name="Profile"
        options={ ({route}) => ( {title: route.params.title} )}

        component={ProfileScreen}
        />

      </Stack.Navigator>

    </NavigationContainer>
  );
};

const HomeScreen = ( {navigation} ) => {
  return (

    <Button
      title="GO to profile page"
      onPress={() => navigation.navigate('Profile', {title: "This is new
title"})
      }
    />

  );
};

const ProfileScreen = ({ navigation, route }) => {
  return (
    <>

  <Text> This is profile page </Text>

  </>
  );
}

export default App;
```

Setting the title of screens dynamically.

Here we defined a function in options property of Profile screen. This function is a setting the value of “name” in the function component of HomeScreen

```
import * as React from 'react';
import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Profile"
        onPress={() =>
          navigation.navigate('Profile', { name: 'This is new title' })
        }
      />
    </View>
  );
}

function ProfileScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile screen</Text>
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Home Screen' }}
        />
        <Stack.Screen
          name="Profile"
          component={ProfileScreen}
        // The below line will set title of profile screen dynamically
          options={({ route }) => ({ title: route.params.name })}
```

```
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Changing header style

```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{
            title: 'My home',
            headerStyle: {
              backgroundColor: '#f4511e',
            },
            headerTintColor: '#fff',
          }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Sharing same header styles across multiple screens

We can instead move the configuration up to the native stack navigator under the prop screenOptions.

```
import * as React from 'react';
import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen({navigation}) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
      title="Click Here"
      onPress={() => navigation.navigate('Profile')}
      />
    </View>
  );
}

function ProfileScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator
        screenOptions={{
          headerStyle: {
            backgroundColor: '#f4511e',
          },
          headerTintColor: '#fff',
        }}
      >
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'My home' }}
        />
```

```
        <Stack.Screen
        name="Profile"
        component={ProfileScreen}
        options={{title: "My profile"}}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Replacing the title with a custom component

headerTitle is a property that is specific to stack navigators, the headerTitle defaults to a Text component that displays the title.

```
import * as React from 'react';
import { View, Text, Image } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

function LogoTitle() {
  return (
    <Image
      style={{ width: 50, height: 50 }}
      source={require('./images/react-native-logo.png')}
    />
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
```

```
          component={HomeScreen}
          options={{ headerTitle: (props) => <LogoTitle {...props} /> }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Adding a button to the header

```
import * as React from 'react';
import { View, Text, Button, Image } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

function LogoTitle() {
  return (
    <Image
      style={{ width: 50, height: 50 }}
      source={require('./images/react-native-logo.png')}
    />
  );
}

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{
            headerTitle: (props) => <LogoTitle {...props} />,
            headerRight: () => (
              <Button
```

```
                onPress={() => alert('This is a button!')}
                title="Info"
                color="#00cc00"
              />
            ),
          }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Header interaction with its screen component

```
import * as React from 'react';
import { Button, Text, Image } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function LogoTitle() {
  return (
    <Image
      style={{ width: 50, height: 50 }}
      source={require('./images/react-native-logo.png')}
    />
  );
}

function HomeScreen({ navigation }) {
  const [count, setCount] = React.useState(0);

  React.useEffect(() => {
    // Use `setOptions` to update the button that we previously specified
    // Now the button includes an `onPress` handler to update the count
    navigation.setOptions({
      headerRight: () => (
        <Button onPress={() => setCount((c) => c + 1)} title="Update count" />
      ),
    });
  }, [navigation, setCount]);

  return <Text>Count: {count}</Text>;
}
```

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={({ navigation, route }) => ({
            headerTitle: (props) => <LogoTitle {...props} />,
            // Add a placeholder button without the `onPress` to avoid flicker
            headerRight: () => (
              <Button title="Update count" />
            ),
          })}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Tab navigation example

(More examples: https://reactnavigation.org/docs/tab-based-navigation)

We are simply navigating from home screen to details screen without changing the tabs.

```
import * as React from 'react';
import { Button, Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

function DetailsScreen() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Details!</Text>
    </View>
  );
}

function HomeScreen({ navigation }) {
```

```
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Home screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function SettingsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Settings screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

const HomeStack = createNativeStackNavigator();

function HomeStackScreen() {
  return (
    <HomeStack.Navigator>
      <HomeStack.Screen name="Home" component={HomeScreen} />
      <HomeStack.Screen name="Details" component={DetailsScreen} />
    </HomeStack.Navigator>
  );
}

const SettingsStack = createNativeStackNavigator();

function SettingsStackScreen() {
  return (
    <SettingsStack.Navigator>
      <SettingsStack.Screen name="Settings" component={SettingsScreen} />
      <SettingsStack.Screen name="Details" component={DetailsScreen} />
    </SettingsStack.Navigator>
  );
}

const Tab = createBottomTabNavigator();

export default function App() {
```

```
  return (
    <NavigationContainer>
      <Tab.Navigator screenOptions={{ headerShown: false }}>
        <Tab.Screen name="Home" component={HomeStackScreen} />
        <Tab.Screen name="Settings" component={SettingsStackScreen} />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Drawer Navigation Example

(More examples: https://reactnavigation.org/docs/drawer-based-navigation)

```
import * as React from 'react';
import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import {
  createDrawerNavigator,
  DrawerContentScrollView,
  DrawerItemList,
  DrawerItem,
} from '@react-navigation/drawer';

function Feed({ navigation }) {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Feed Screen</Text>
      <Button title="Open drawer" onPress={() => navigation.openDrawer()} />
      <Button title="Toggle drawer" onPress={() => navigation.toggleDrawer()}
/>
    </View>
  );
}

function Notifications() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Notifications Screen</Text>
    </View>
  );
}

function CustomDrawerContent(props) {
```

```
  return (
    <DrawerContentScrollView {...props}>
      <DrawerItemList {...props} />
      <DrawerItem
        label="Close drawer"
        onPress={() => props.navigation.closeDrawer()}
      />
      <DrawerItem
        label="Toggle drawer"
        onPress={() => props.navigation.toggleDrawer()}
      />
    </DrawerContentScrollView>
  );
}

const Drawer = createDrawerNavigator();

function MyDrawer() {
  return (
    <Drawer.Navigator
      useLegacyImplementation
      drawerContent={(props) => <CustomDrawerContent {...props} />}
    >
      <Drawer.Screen name="Feed" component={Feed} />
      <Drawer.Screen name="Notifications" component={Notifications} />
    </Drawer.Navigator>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <MyDrawer />
    </NavigationContainer>
  );
}
```

Connecting with firestore realtime database

Follow this document for firestore database connectivity:

https://firebase.google.com/docs/firestore

For this you need to create an “expo” project

Create an online firebase firestore database.

Copy the configuration code (to be pasted in firestoreconfig.js)

Install firebase in project using: npm `install firebase`

Give command: npm run android

firebaseconfig.js

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import {getFirestore} from  "firebase/firestore";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDKSJfAZLNWA328ArcCdQXwtRMiIaRm7dA",
  authDomain: "dbproj-f3054.firebaseapp.com",
  projectId: "dbproj-f3054",
  storageBucket: "dbproj-f3054.appspot.com",
  messagingSenderId: "352306058995",
  appId: "1:352306058995:web:962d3872b952ef33478986"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);

export {db};
```

App.js

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc } from 'firebase/firestore';

import {db} from './firestoreconfig.js';

// To create or overwrite a single document, use the following language-
specific set() methods:

async function writeData() {
  await setDoc(doc(db, "users", "kamal@gmail.com"), {
    name: "Kamal Ahmed",
    age: "22"
```

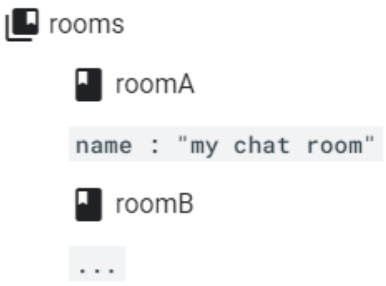
```
  });

}

/*
If the document does not exist, it will be created. If the document does
exist, its contents will be overwritten with the newly provided data, unless
you specify that the data should be merged into the existing document, as
follows:
*/
function mergeData()
{
  const userRef = doc(db, 'users', 'ali@gmail.com');
setDoc(userRef, { age: 46 }, { merge: true });
}


async function readData() {

  const docRef = doc(db, "users", "ali@gmail.com");
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    console.log("Document data:", docSnap.data());
  } else {
    // doc.data() will be undefined in this case
    console.log("No such document!");
  }


}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   <Button
   title='Read Data'
   onPress={ () => readData()}
    />
```

```
<Button
   title='Merge Data'
   onPress={ () => mergeData()}
    />

   </View>
   );
}

export default App;
```

Example Storing the Hierarchical Data

To understand how hierarchical data structures work in Cloud Firestore, consider an example chat app with messages and chat rooms.

You can create a collection called `rooms` to store different chat rooms:



Now that you have chat rooms, decide how to store your messages. You might not want to store them in the chat room's document. Documents in Cloud Firestore should be lightweight, and a chat room could contain a large number of messages. However, you can create additional collections within your chat room's document, as subcollections.

Subcollections

The best way to store messages in this scenario is by using subcollections. A subcollection is a collection associated with a specific document.

> **Note:** You can query across subcollections with the same collection ID by using Collection Group Queries.

You can create a subcollection called `messages` for every room document in your `rooms` collection:

rooms

roomA

`name : "my chat room"`

messages

message1

`from : "alex"`
`msg : "Hello World!"`

message2

`...`

roomB

`...`

```
import React from 'react';
import { Button, View } from 'react-native';

import { doc, getDoc } from 'firebase/firestore';

import {db} from './firestoreconfig.js';



async function readData() {

  const docRef = doc(db, "rooms", "roomA", "messages", "message1");

  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    console.log("Document data:", docSnap.data());
  } else {
    // doc.data() will be undefined in this case
    console.log("No such document!");
```

```
  }


}



const App = () => {
  return (

<View
style = {{marginTop: 50}}
>


   <Button
   title='Read Data'
   onPress={ () => readData()}
    />



   </View>
   );
}

export default App;
```

Data types in Firestore

Cloud Firestore lets you write a variety of data types inside a document, including strings, booleans, numbers, dates, null, and nested arrays and objects. Cloud Firestore always stores numbers as doubles, regardless of what type of number you use in your code.

```
import React from 'react';
import { Button, View } from 'react-native';

import { doc, setDoc, Timestamp } from "firebase/firestore";

import {db} from './firestoreconfig.js';



async function writeData() {
```

```
  const docData = {
    stringExample: "Hello world!",
    booleanExample: true,
    numberExample: 3.14159265,
    dateExample: Timestamp.fromDate(new Date("December 10, 1815")),
    arrayExample: [5, true, "hello"],
    nullExample: null,
    objectExample: {
        a: 5,
        b: {
            nested: "foo"
        }
    }
};
await setDoc(doc(db, "data", "one"), docData);

}



const App = () => {
  return (

<View
style = {{marginTop: 50}}
>


   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   </View>
   );
}

export default App;
```

Custom objects

Using Map or Dictionary objects to represent your documents is often not very convenient, so Cloud Firestore supports writing documents with custom classes. Cloud Firestore converts the objects to supported data types.

Using custom classes, you could rewrite the initial example as shown:

```
import React from 'react';
import { Button, View } from 'react-native';

import { doc, setDoc } from "firebase/firestore";

import {db} from './firestoreconfig.js';


class City {
  constructor (name, state, country ) {
      this.name = name;
      this.state = state;
      this.country = country;
  }
  toString() {
      return this.name + ', ' + this.state + ', ' + this.country;
  }
}

// Firestore data converter
const cityConverter = {
  toFirestore: (city) => {
      return {
          name: city.name,
          state: city.state,
          country: city.country
          };
  },
  fromFirestore: (snapshot, options) => {
      const data = snapshot.data(options);
      return new City(data.name, data.state, data.country);
  }
};



async function writeData() {


  const ref = doc(db, "cities", "LA").withConverter(cityConverter);
  await setDoc(ref, new City("Los Angeles", "CA", "USA"));

}



const App = () => {
  return (
```

```
<View
style = {{marginTop: 50}}
>


   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   </View>
   );
}

export default App;
```

Add a document with explicitly specified ID

When you use set() to create a document, you must specify an ID for the document to create. For example:

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc } from 'firebase/firestore';

import {db} from './firestoreconfig.js';

// To create or overwrite a single document, use the following language-
specific set() methods:

async function writeData() {
    await setDoc(doc(db, "cities", "new-city-id"), {country: "Pakistan", name:
"Sargodha", state: "Punjab"});

}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
```

```
   title='Write Data'
   onPress={ () => writeData()}
    />

   <Button
   title='Read Data'

    />

<Button
   title='Merge Data'

    />

   </View>
   );
}

export default App;
```

Add document with auto-generated ID

But sometimes there isn't a meaningful ID for the document, and it's more convenient to let Cloud Firestore auto-generate an ID for you. You can do this by calling the following language-specific add() methods:

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, addDoc } from
'firebase/firestore';

import {db} from './firestoreconfig.js';



async function writeData() {

    // Add a new document with a generated id.
const docRef = await addDoc(collection(db, "cities"), {
    name: "Tokyo",
    country: "Japan",
    state: "North"
```

```
  });
  console.log("Document written with ID: ", docRef.id);


}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   <Button
   title='Read Data'

    />

<Button
   title='Merge Data'

    />

   </View>
   );
}

export default App;
```

Create a document reference with auto-generated ID

In some cases, it can be useful to create a document reference with an auto-generated ID, then use the reference later. For this use case, you can call doc():

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, addDoc } from
'firebase/firestore';
```

```
import {db} from './firestoreconfig.js';



async function writeData() {

    // Add a new document with a generated id
const newCityRef = doc(collection(db, "cities"));

// later...
await setDoc(newCityRef, {country: "Germany", name: "Berg", state: "South"});


}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   <Button
   title='Read Data'

    />

<Button
   title='Merge Data'

    />

   </View>
   );
}

export default App;
```

Behind the scenes, .add(...) and .doc().set(...) are completely equivalent, so you can use whichever is more convenient.

Update a document

To update some fields of a document without overwriting the entire document, use the following language-specific update() methods:

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, addDoc, updateDoc } from
'firebase/firestore';

import {db} from './firestoreconfig.js';



async function updateData() {

    const Ref = doc(db, "cities", "KHI");

// Set the "capital" field of the city 'DC'
await updateDoc(Ref, {state: "Punjab"});



}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Update Data'
   onPress={ () => updateData()}
    />


   </View>
   );
}

export default App;
```

Server Timestamp

You can set a field in your document to a server timestamp which tracks when the server receives the update.

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, addDoc, updateDoc,
serverTimestamp  } from 'firebase/firestore';

import {db} from './firestoreconfig.js';



async function writeData() {

    const docRef = doc(db, 'cities', 'KHI');

// Update the timestamp field with the value from the server
const updateTimestamp = await updateDoc(docRef, {
    timestamp: serverTimestamp()
});



}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Update Data'
   onPress={ () => writeData()}
    />


   </View>
   );
}

export default App;
```

When updating multiple timestamp fields inside of a transaction, each field receives the same server timestamp value.

Update fields in nested objects

If your document contains nested objects, you can use "dot notation" to reference nested fields within the document when you call update():

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, addDoc, updateDoc,
serverTimestamp  } from 'firebase/firestore';

import {db} from './firestoreconfig.js';



async function updateData() {

// Create an initial document to update.
const frankDocRef = doc(db, "users", "frank");
await setDoc(frankDocRef, {
    name: "Frank",
    favorites: { food: "Pizza", color: "Blue", subject: "recess" },
    age: 12
});

// To update age and favorite color:
await updateDoc(frankDocRef, {
    "age": 13,
    "favorites.color": "Red"
});


}


const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Update Data'
   onPress={ () => updateData()}
```

```
    />


   </View>
   );
}

export default App;
```

Dot notation allows you to update a single nested field without overwriting other nested field. If you update a nested field without dot notation, you will overwrite the entire map field, for example:

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, updateDoc } from
'firebase/firestore';

import {db} from './firestoreconfig.js';

// To create or overwrite a single document, use the following language-
specific set() methods:

async function writeData() {
    const DocRef = doc(db, "users", "tommy");
    await setDoc(DocRef, {
        name: "Frank",
        favorites: { food: "Pizza", color: "Blue", subject: "recess" },
        age: 12
    });

    // To update age and favorite color:
    await updateDoc(DocRef, {
        "age": 13,
        "favorites.color": "Red"
    });

}


async function updateData() {

    // Update the doc without using dot notation.
// Notice the map value for favorites.

const DocRef = doc(db, "users", "tommy");
```

```
await updateDoc(DocRef, {
    favorites: {
      food: "Ice Cream"
    }
  });

}

const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Write Data'
   onPress={ () => writeData()}
    />

   <Button
   title='Update Data'
   onPress={ () => updateData()}
    />


   </View>
   );
}

export default App;
```

Update elements of an array

```
import React from 'react';
import { Button, View } from 'react-native';

import {collection, getDocs, getDoc, doc, setDoc, updateDoc, arrayUnion,
arrayRemove } from 'firebase/firestore';

import {db} from './firestoreconfig.js';

// To create or overwrite a single document, use the following language-
specific set() methods:

async function removeData() {
    const docRef = doc(db, "cities", "KHI");
```

```
    // Atomically remove a region from the "regions" array field.
await updateDoc(docRef, {regions: arrayRemove("greater_virginia")});

}


async function updateData() {

const Ref = doc(db, "cities", "KHI");

// Atomically add a new region to the "regions" array field.
await updateDoc(Ref, {
    regions: arrayUnion("greater_virginia")
});


}

const App = () => {
  return (

<View
style = {{marginTop: 50}}
>


   <Button
   title='Update Data'
   onPress={ () => updateData()}
    />

<Button
   title='Remove Data'
   onPress={ () => removeData()}
    />

   </View>
   );
}

export default App;
```

Increment a numeric value

You can increment or decrement a numeric field value as shown in the following example. An increment operation increases or decreases the current value of a field by the given amount.

```
import React from 'react';
import { Button, View } from 'react-native';
```

```
import {collection, increment, getDocs, getDoc, doc, setDoc, updateDoc,
arrayUnion, arrayRemove } from 'firebase/firestore';

import {db} from './firestoreconfig.js';



async function updateData() {

    const washingtonRef = doc(db, "cities", "LA");

    //suppose we have a population field in cities for "DC" document

    // Atomically increment the population of the city by 50.
    await updateDoc(washingtonRef, {
        population: increment(50)
    });


}

const App = () => {
  return (

<View
style = {{marginTop: 50}}
>


   <Button
   title='Update Data'
   onPress={ () => updateData()}
    />

<Button
   title='Remove Data'
   onPress={ () => removeData()}
    />

   </View>
   );
}

export default App;
```