Mobile Application Development

(React Native Programming)

Dr. Osman Khalid

Updated on 18 Dec 2023

http://osman.pakproject.com

Table of Contents

JAVASCRIPT ADVANCED CONCEPTS

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>
```

Javascript spread syntax

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

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

console.log(myArray2)

var myObject1 = { name: 'Devin', hairColor: 'brown' }

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

console.log(myObject2)

// to update value of name, we will write

myObject2 = { ...myObject2, name: 'Malik' }

console.log(myObject2)
```

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>
```

Defining variables in a separate file

Create a file named variables.js and write the following code:

```
export default country = "Pakistan";
```

Now we can use this value in another javascript file like:

myfile.js

```
import country  from './variables';

console.log(country);
```

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);
```

INTRODUCTION TO REACT NATIVE

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;
```

Product.js can also be written like this:

NOTE: Since we are using here export default so in the App.js this is how we will include it:

```
import Product from './Product';
```

If we remove default keyword, then we will export file like: export {Product} and then in App.js we will include it like this:

```
import {Product} from './Product';
```

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

export default function Product() {
    return(
    <View>
        <Text>This is product component</Text>
    </View>
    );
}
```

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',
  },
});
```

ANIMATIONS IN REACT NATIVE

https://reactnative.dev/docs/animations

CORE COMPONENTS

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;
```

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',
    },
});
```

Send 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;
```

Note: In above example, when we write: `color="powderblue" {...powderblue}`

this makes it a single object, and in the receiving function component this single object is destructured into multiple attributes.

Example:

obj1 = { id: 10, name: 'Ali'}

obj2 = { age: 45 }

obj3 = {...obj1, ...obj2} // both objects are merged to a single object

Destructuring:

const {id, name, age} = obj3;

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;
```

USING STYLES IN REACT NATIVE

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


export default function App() {
  const [status] = useState( true );
  const [number] = useState(300);
  return (
    <Text style={ number < 200 && status==true ? [styles.box1,
styles.paragraph] : styles.box2}> Hello world </Text>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  paragraph: {
```

```
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },

  box1: {
    backgroundColor: 'blue',
    width: 100,
    height: 100,
  },



    box2: {
    backgroundColor: 'green',
    width: 100,
    height: 100,
  }
});
```

Dynamic class assignment example:

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


export default function App() {

  const [num1] = useState(30);
  const [num2] = useState(20);
  const [num3] = useState(10);


  var newClass = null;

  if(num1 > num2 && num1 > num3)
  {
    newClass = styles.box1;
  }
  else if(num2 > num3 && num2 > num1)
  {
   newClass = styles.box2;
  }
  else if(num3 > num1 && num3 > num2)
  {
```

```
  newClass = styles.box3;
  }

  return (
<Text style = {newClass}>HELLO WORLD</Text>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },

  box1: {
    backgroundColor: 'red',
    width: 100,
    height: 100,
  },
  box2: {
    backgroundColor: 'blue',
    width: 100,
    height: 100,
  },
  box3: {
    backgroundColor: 'green',
    width: 100,
    height: 100,
  },

});
```

Another example

```
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 LAYOUT

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’

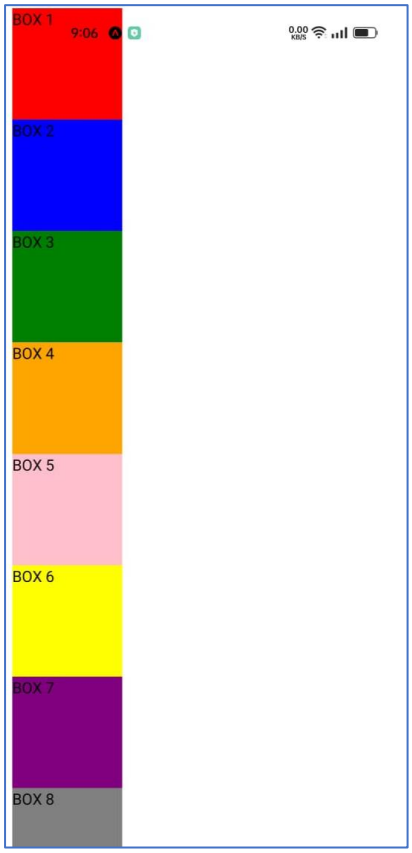
(Run this example on your mobile or emulator to see correct output)

```
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,
```

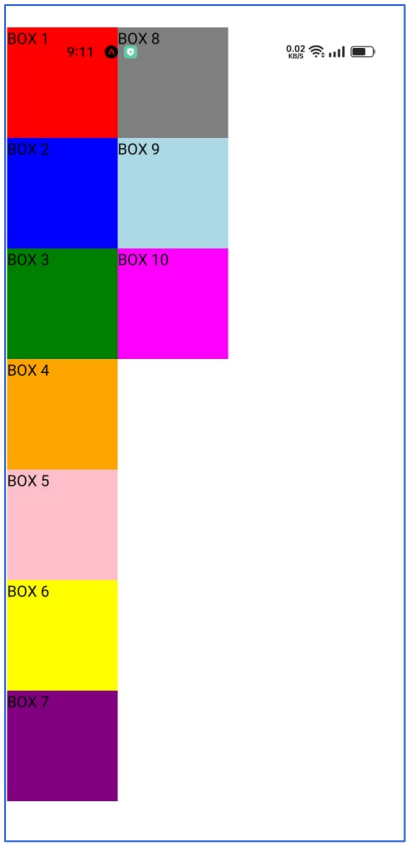
```
  },
});

export default App;
```

Output:



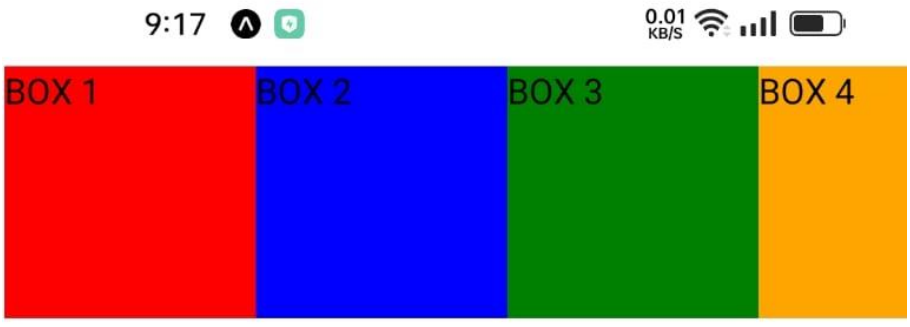
The following will be the output with

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



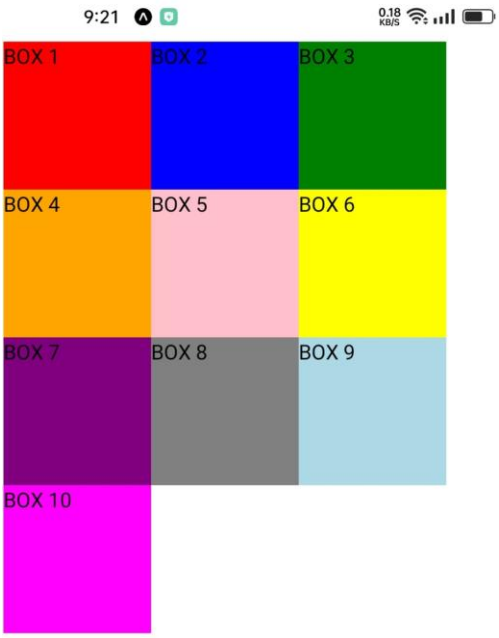
The following will be the output with:

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

export default function App() {

  const [count1, setCount1] = useState(0);
  const [count2, setCount2] = useState(0);

  useEffect( () => {
    console.warn("COUNT1: " + count1 + " COUNT2: " + count2)},
  [count1]

);

/* 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.
*/


  return (
    <View>
      <Button
      title="Click 1"
      onPress = {() => setCount1(count1+1)}
       />
      <Button
      title="Click 2"
      onPress = {() => setCount2(count2+1)}
       />

    </View>
  );
}
```

NAVIGATING BETWEEN MULTIPLE SCREENS

Simple example of react-native navigation

For installation, visit this site: https://reactnavigation.org/docs/getting-started/

```
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>
```

Optional Chaining operator react native example

```
import { NavigationContainer} from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";

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

const Stack = createNativeStackNavigator();

function HomeScreen({navigation, route})
{
  /*
  const adventurer = {
    name: 'Alice',
    cat: {
      name: 'Dinah',
    },
  };

  console.warn(adventurer.dog?.name);
  */

  return(
    <View>
    <Text>This is home screen</Text>
    <Text>{route.params?.city == undefined ? "" : "City: " +
route.params?.city}</Text>
    <Button
    title="Go to profile screen"
    onPress={() => navigation.navigate('Profile', {name: 'Ali', age: 10})}
    />
    </View>
  )
}

function ProfileScreen({navigation, route})
{
  const {name, age} = route.params;
  return(
    <View>
    <Text>This is profile screen</Text>
    <Text>{name} {age}</Text>
```

```
    <Button
    title = "Send data"
    onPress={ () => navigation.navigate('Home', {city: 'Peshawar'})}
    />
    </View>
  )
}

const App = () => {
  return(
    <NavigationContainer>
      <Stack.Navigator>

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

        <Stack.Screen
         name = "Profile"
         component={ProfileScreen}
         options={ {title: 'Profile'}}
         />

      </Stack.Navigator>

    </NavigationContainer>
  )

}

export default App;
```

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.

```jsx
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()

Example 1 of setParams:

```
import { NavigationContainer} from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";

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

const Stack = createNativeStackNavigator();

function HomeScreen({navigation})
```

```
{

  return(
    <View>
    <Text>This is home screen</Text>

    <Button
    title="Go to profile screen"
    onPress={() => navigation.navigate('Profile', {name: 'Ali', age: 10,
title: 'Ali profile'})}
    />
    </View>
  )
}

function ProfileScreen({navigation, route})
{

  return(
    <View>
    <Text>This is profile screen</Text>
    <Text>{route.params.name} {route.params.age}</Text>
    <Button
    onPress={() => navigation.setParams({name: "Javed", age: 45}) }
    title = "Update the parameters"
    />

    </View>
  )
}

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

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

        <Stack.Screen
         name = "Profile"
         component={ProfileScreen}
         // To make the title dynamic
         options={ ({route}) => ({title: route.params.title})}
         />
```

```
      </Stack.Navigator>

    </NavigationContainer>
  )

}

export default App;
```

Example 2 of setParams

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

```
import React, {useState} 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 screen"
      onPress={()=>navigation.navigate('Profile',
      {
        title:'Profile',
        friends: ['Ali', 'Jawad', 'Faisal'],
        person: 'Faisal',
        })}
      />
    </View>
  );
}

function ProfileScreen({navigation, route}) {

  return(
    <View>
    <Text>This is profile screen</Text>
    <Text>{route.params.friends[0]}</Text>
    <Text>{route.params.friends[1]}</Text>
    <Text>{route.params.friends[2]}</Text>

    <Button
    onPress={() => {
```

```
    if( route.params.person === 'Ali')
    {
    navigation.setParams({friends:['A1', 'A2', 'A3'], title: "Ali's friend
list"});
    }
    else if( route.params.person === 'Jawad')
    {
    navigation.setParams({friends:['J1', 'J2', 'J3'], title: "Jawad's friend
list"});
    }
     else if( route.params.person === 'Faisal')
    {
    navigation.setParams({friends:['F1', 'F2', 'F3'], title: "Faisal's friend
list"});
    }


    }
    }

    title="Click here to change parameters"
    />
    </View>
  )
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>

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

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

GLOBAL STORAGE WITH CONTEXT API

Simple example of using Context API

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

const Context1 = React.createContext(null);


function HomeScreen({navigation}) {
  const context = React.useContext(Context1);


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

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

```
function ProfileScreen() {
  const context = React.useContext(Context1);


  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile Screen</Text>
      <Text>{context.name} {context.age}</Text>
      <Button title="Change global variable values"
      onPress = {() => context.setData('Shahid', 50)
        }
      />

    </View>
  );
}

const Stack = createNativeStackNavigator();



function App() {

const [personInfo, setPersonInfo] = useState({name: 'Ali', age:30});


const context1Setters = {setName, setAge, setData}

function setName(name) {
  const updatedpersonInfo = { ...personInfo, name};
  setPersonInfo( updatedpersonInfo );
}

function setAge(age) {
  const updatedpersonInfo = { ...personInfo, age};
  setPersonInfo( updatedpersonInfo );
}

function setData(name, age) {
  const updatedpersonInfo = { ...personInfo, name, age};
  setPersonInfo( updatedpersonInfo );
}



  return (
```

```
<Context1.Provider value={{...personInfo, ...context1Setters}}>
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
</Context1.Provider>
  );
}

export default App;
```

Context API app with multiple files

Create following files:

App.js

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import HomeScreen from './HomeScreen';
import ProfileScreen from './ProfileScreen';

export const Context1 = React.createContext(null);

const Stack = createNativeStackNavigator();


function App() {

const [personInfo, setPersonInfo] = useState({name: 'Ali', age:30});


const personInfoSetters = {setName, setAge, setData}

function setName(name) {
  const updatedpersonInfo = { ...personInfo, name};
  setPersonInfo( updatedpersonInfo );
}

function setAge(age) {
  const updatedpersonInfo = { ...personInfo, age};
  setPersonInfo( updatedpersonInfo );
}

function setData(name, age) {
  const updatedpersonInfo = { ...personInfo, name, age};
```

```
  setPersonInfo( updatedpersonInfo );
}



  return (

<Context1.Provider value={{...personInfo, ...personInfoSetters}}>
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
</Context1.Provider>
  );
}

export default App;
```

HomeScreen.js

```
import React from 'react';
import {Text, View, Button} from 'react-native';
import {Context1} from './App';

export default function HomeScreen({navigation}) {

  const context = React.useContext(Context1);


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

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

ProfileScreen.js

```
import React from 'react';
import {Text, View, Button} from 'react-native';
import {Context1} from './App';


export default function ProfileScreen() {
  const context = React.useContext(Context1);


  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Profile Screen</Text>
      <Text>{context.name} {context.age}</Text>
      <Button title="Change global variable values"
      onPress = {() => context.setData('Shahid', 50)
        }
      />

    </View>
  );
}
```

GLOBAL STORAGE USING REDUX TOOLKIT

Creating a global storage for react navigation using redux toolkit api

Run the following command to install redux toolkit api:

```
npm install @reduxjs/toolkit
```

```
npm install react-redux
```

To install the react navigation, use the following commands

```
npm install @react-navigation/native
```

```
npm install react-native-screens react-native-safe-area-context
```

```
npm install @react-navigation/native-stack
```

Simple Counter Example with Redux Toolkit

Create the following files:

store.js

```
import { configureStore } from '@reduxjs/toolkit'
import counterReducer from './counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
})
```

counterSlice.js

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    value: 0
}

export const counterSlice = createSlice({
    name: 'counter',
    initialState,

    reducers: {

        increment: (state) => {
            state.value += 1
        },

        decrement: (state) => {
            state.value -= 1
        },

        incrementByAmount: (state, action) => {
            state.value += action.payload
        }

    }
})

export const {increment, decrement, incrementByAmount} = counterSlice.actions;

export default counterSlice.reducer;
```

HomeScreen.js

```
import {View, Text, Button} from 'react-native';
import { useSelector, useDispatch } from 'react-redux';
import { increment } from './counterSlice';
```

```
export default function HomeScreen({navigation}) {

const count = useSelector( (state) => state.counter.value)
const dispatch = useDispatch();

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Text>Current value of count: {count}</Text>
      <Button
      title = "Update counter"
      onPress={() => dispatch( increment() )}
      />

      <Button
      title="go to display screen"
      onPress = {() => navigation.navigate('Display')}
      />
    </View>
  );
}
```

DisplayCounter.js

```
import {View, Text, Button} from 'react-native';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement } from './counterSlice';

export default function DisplayScreen() {

 const count = useSelector( (state) => state.counter.value)
 const dispatch = useDispatch();


  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Display Screen</Text>
      <Text>Count: {count}</Text>

      <Button title="Update counter"
      onPress={() => dispatch( increment() )}
      />



          </View>
  );
```

```
}
```

App.js

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

import {store} from './store';
import { Provider } from 'react-redux';
import HomeScreen from './HomeScreen'
import DisplayCounter from './DisplayCounter'

import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';


const Stack = createNativeStackNavigator();

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

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

export default App;
```

Example of globally storing person attributes with Redux Toolkit

Create the following files:

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import personReducer from "./personSlice";

export const store = configureStore({
    reducer: {
        counter: counterReducer,
        person: personReducer,
    },
})
```

personSlice.js

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    email: 'ali@gmail.com',
    name: 'Ali'
}

export const personSlice = createSlice({
    name: 'person',
    initialState,

    reducers: {

        updateEmail: (state, action) => {
            return {...state, email: action.payload}
        },

        updateName: (state, action) => {
            return {...state, name: action.payload}
        },


    }
})

export const {updateEmail, updateName} = personSlice.actions;

export default personSlice.reducer;
```

HomeScreen.js

```
import {View, Text, Button} from 'react-native';
import { useSelector, useDispatch } from 'react-redux';
import { updateEmail, updateName } from './personSlice';


export default function HomeScreen({navigation}) {

const person = useSelector( (state) => state.person)
const dispatch = useDispatch();

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Text>{person.email} {person.name}</Text>

      <Button
      title = "Update Email"
      onPress={() => dispatch( updateEmail('jamal@abc.com') )}
      />

<Button
      title = "Update Name"
      onPress={() => dispatch( updateName('Jamal Khan') )}
      />

      <Button
      title="go to display screen"
      onPress = {() => navigation.navigate('Display')}
      />
    </View>
  );
}
```

DisplayPerson.js

```
import {View, Text, Button} from 'react-native';
import { useSelector } from 'react-redux';



export default function DisplayPerson({navigation}) {



 const person = useSelector( (state) => state.person);
```

```
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Display Screen</Text>
      <Text>Email: {person.email}</Text>
      <Text>Name: {person.name}</Text>
    </View>
  );
}
```

App.tsx

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

import {store} from './store';
import { Provider } from 'react-redux';
import HomeScreen from './HomeScreen';
import DisplayPerson from './DisplayPerson'

import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';



const Stack = createNativeStackNavigator();

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

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

export default App;
```

The below two files show an example of how we can store a class object in global storage and retrieve and update its values.

HomeScreen.js

```
import {View, Text, Button} from 'react-native';
import { useSelector, useDispatch } from 'react-redux';
import { updateEmail, updateName, updateCity } from './personSlice';

export default function HomeScreen({navigation}) {

const person = useSelector( (state) => state.person)
const dispatch = useDispatch();

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Text>{person.email} {person.name}</Text>

      <Button
      title = "Update Email"
      onPress={() => dispatch( updateEmail('jamal@abc.com') )}
      />

<Button
      title = "Update Name"
      onPress={() => dispatch( updateName('Jamal Khan') )}
      />

<Button
      title = "Update City"
      onPress={() => dispatch( updateCity(  {cityName: 'Lahore', cityLocation:
'Clifton'}) )}
      />

      <Button
      title="go to display screen"
      onPress = {() => navigation.navigate('Display')}
      />
    </View>
  );
}
```

personSlice.js

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    email: 'ali@gmail.com',
```

```
    name: 'Ali'
}

export const personSlice = createSlice({
    name: 'person',
    initialState,

    reducers: {

        updateEmail: (state, action) => {
            return {...state, email: action.payload}
        },

        updateName: (state, action) => {
            return {...state, name: action.payload}
        },

        updateCity: (state, action) => {
            return {...state, city: action.payload}
        }


    }
})

export const {updateEmail, updateName, updateCity} = personSlice.actions;

export default personSlice.reducer;
```

DisplayPerson.js

```
import {View, Text, Button} from 'react-native';
import { useSelector } from 'react-redux';



export default function DisplayPerson({navigation}) {

 const person = useSelector( (state) => state.person);

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Display Screen</Text>
      <Text>Email: {person.email}</Text>
      <Text>Name: {person.name}</Text>
      <Text>City: {person.city?.cityName} Area:
{person.city?.cityLocation}</Text>
```

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

ASYNCHRONOUS STORAGE

React Native Asynchronous Storage

To install the library, use the following command:

`npm install @react-native-async-storage/async-storage`

```
import React from 'react';
import { View, Text, Button } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

class Person
{
  constructor(name, age)
  {
    this.name = name;
    this.age = age;
  }
}

const storeStringData = async (value) => {
  try {
    await AsyncStorage.setItem('@storage_Key', value)
  } catch (e) {
    // saving error
    console.log(e);
  }
}


const getStringData = async () => {
  try {
    const value = await AsyncStorage.getItem('@storage_Key')

    if(value !== null) {
      // value previously stored
      console.log(value);
    }
```

```
  } catch(e) {
    // error reading value
    console.log(e);
  }
}

const storeObjectData = async (value) => {
  try {
    const jsonValue = JSON.stringify(value)
    await AsyncStorage.setItem('@storage_object_Key', jsonValue)
  } catch (e) {
    // saving error
    console.log(e);
  }
}


const getObjectData = async () => {
  try {
    var jsonValue = await AsyncStorage.getItem('@storage_object_Key')
    jsonValue = jsonValue != null ? JSON.parse(jsonValue) : null
    console.log(jsonValue.name, " ", jsonValue.age);
  } catch(e) {
    // error reading value
    console.log(e);
  }
}
```

```
async function storeArrayData()
{
  try{
    await AsyncStorage.setItem('@array_key',  JSON.stringify([10,30,40,50]));
  }
  catch(e)
  {
   console.warn(e);
  }
}

async function getArrayData()
{
  try{
  const myarray = await AsyncStorage.getItem('@array_key');

  var aa = JSON.parse(myarray);

  if(aa != null)
  {
    alert( aa[0] + " " + aa[1])
```

```
  }
  }
  catch(e)
  {
   alert(e);
  }
}
```

```
const App = () => {

  return (
 <View>
<Button
title="Store string"
onPress={() => storeStringData("Hello")}
/>

<Button
title="Store object"
onPress={() => storeObjectData({cityname: "Lahore", cityLocation: "Punjab"})}
/>

<Button
title="Store Class object"
onPress={() => storeObjectData(new Person("Ali", 45))}
/>

<Button
title="Get string"
onPress={() => getStringData()}
/>

<Button
title="Get object"
onPress={() => getObjectData()}
/>
```

```
<Button
      title="Store array data"
      onPress={() => storeArrayData()}
      />
      <Button
      title="Get array data"
      onPress={() => getArrayData()}
      />
```

```
    </View>
  );
};

export default App;
```

For more methods, please visit:

https://react-native-async-storage.github.io/async-storage/docs/api/

CONNECTING WITH SQLITE DATABASE

Connecting with SQLite Database with expo

To install library:

npx expo install expo-sqlite

```
import { useEffect, useState } from 'react';
import { View,  Button, StyleSheet, Text } from 'react-native';
import * as SQLite from 'expo-sqlite';


const App = () => {
  const [users, setUsers]= useState([]);

const db = SQLite.openDatabase('example.db');

useEffect( ()=>{
  db.transaction( (tx)=>{
    tx.executeSql("create table if not exists users (id integer primary key
autoincrement, name text, age integer);"
    )

    db.transaction( (tx) => {tx.executeSql("select * from users",null,
    (txObj, resultSet) => setUsers(resultSet.rows._array),
    (txObj, error) => console.log(error)
    )})


  })},[])
```

```
  const addUser = () => {
    var name = "javed";
  var age = "51";

    db.transaction( (tx) => {

      tx.executeSql(
         "INSERT INTO users (name, age) VALUES (?,?)",
         [name, age],
         (txObj, resultSet) => {console.log("success")},
         (txObj, error) => console.log(error)
     );
  })
  }



  const getData = () => {
    try {

        db.transaction((tx) => {
            tx.executeSql(
                "SELECT id, name, age FROM Users",
                [],
                (tx, results) => {
                    var len = results.rows.length;
                    if (len > 0) {
                      for(let i=0; i<len; i++) {

                        var Id = results.rows.item(i).id;
                        var userName = results.rows.item(i).name;
                        var userAge = results.rows.item(i).age;

                        console.log("ID:" + Id + " NAME: " + userName + " AGE:
" + userAge);
                      }
                    }
                }
            )
        })
    } catch (error) {
        console.log(error);
    }
  }



  const updateData = async () => {
```

```
    var name="jawad";
    var id = 3;
       try {

           db.transaction((tx) => {
               tx.executeSql(
                   "UPDATE Users SET name=? where id=?",
                   [name, id],
                   () => { console.warn('Success!', 'Your data has been
updated.') },
                   error => { console.warn(error) }
               )
           })
       } catch (error) {
           console.log(error);
       }

 }

 const removeData = async () => {
   try {
       var id=2;
       db.transaction((tx) => {
           tx.executeSql(
               "DELETE FROM users where id=?",
               [id],
               console.warn("Record deleted"),
               error => { console.warn(error) }
           )
       })
   } catch (error) {
       console.log(error);
   }
 }


  return (

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

<Button
title="Set Data"
onPress = {() => addUser()}
/>

<Button
```

```
title="Get Data"
onPress = {() => getData()}
/>

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

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


    </View>

  )
  }



  const styles = StyleSheet.create({
    container: {
      flex: 1,
      backgroundColor: '#fff',
      alignItems: 'center',
      justifyContent: 'center',
    },
    row: {
      flexDirection: 'row',
      alignItems: 'center',
      alignSelf: 'stretch',
      justifyContent: 'space-between'
    },
  });


export default App;
```

WORKING WITH FETCH API TO STORE DATA IN MYSQL USING PHP

Connecting React Nave with PHP / MySQL

Insall Xampp server

Create the following database in phpMyAdmin

Database name: **rndb**

```
CREATE TABLE `users` (
  `userid` int(11) NOT NULL,
  `uname` varchar(50) NOT NULL,
  `age` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`userid`, `uname`, `age`) VALUES
(1, 'Shahid Najam', 44),
(2, 'Noman Javed', 32),
(3, 'Faisal Ali', 22),
(4, 'Jawad Khan', 45),
(5, 'Shah Khan', 0),

--
-- Indexes for dumped tables
--

--
-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`userid`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `userid` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=15;
COMMIT;
```

Install postman app to test the PHP backend whether it is running fine

Create a CLI project

Install library from following URL:

https://www.npmjs.com/package/react-native-document-picker

using command:

`npm i react-native-document-picker`

Use the following code in react native:

App.js

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

// specify the IP of your PC and name of php app folder as below:
const phpAppAddress = "http://192.168.30.77/rnphpmysql/";

const insertData = async () =>
  {
    console.log(phpAppAddress + 'setdata.php');

    try {
    const response = await fetch(phpAppAddress + 'setdata.php', {
      method: 'POST',
      headers: {
      Accept: 'application/json',
              'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Shahid Khan',
    age: '44'})
  }
  )
  const json = await response.json();
  console.warn(json['Message']);
} catch (error) {
  console.error(error);
} finally {

}
}

const updateData = async () =>
  {
    try {
    const response = await fetch(phpAppAddress + 'updatedata.php', {
      method: 'POST',
      headers: {
      Accept: 'application/json',
```

```
            'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    id: '1',
    name: 'Shahid Najam',
    age: '44'})
  }
  )
  const json = await response.json();
  console.warn(json['Message']);
} catch (error) {
  console.error(error);
} finally {

}
}

const deleteData = async () =>
  {
    try {
    const response = await fetch(phpAppAddress + 'deletedata.php', {
      method: 'POST',
      headers: {
      Accept: 'application/json',
              'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    id: '13',
   })
  }
  )
  const json = await response.json();
  console.warn(json['Message']);
} catch (error) {
  console.error(error);
} finally {

}
}

const getMovies = async () => {
  try {
   const response = await fetch('https://reactnative.dev/movies.json');
   const json = await response.json();
   console.log(json.movies);
 } catch (error) {
   console.error(error);
 } finally {
```

```
 }
}

const getSingleResponse = async () => {
  try {
   const response = await fetch(phpAppAddress + 'getresponse.php');
   const json = await response.json();
   console.warn(json['Message']);
 } catch (error) {
   console.error(error);
 } finally {

 }
}


const App = () => {

  const [records, setRecords] = useState();

  const getAllRecords = async () => {
    try {
     const response = await fetch(phpAppAddress + 'getalldata.php');
     const json = await response.json();
     //console.log(json);
     setRecords(json);

   } catch (error) {
     console.error(error);
   } finally {

   }
  }

//////// Get filtered data
   const getFilteredData = async () => {

      try{
      const response = await fetch( phpAppAddress + 'selectfiltereddata.php',

      {
    method: 'POST',
    headers: {Accept: 'application/json', 'Content-Type':'application/json'},

    body: JSON.stringify( {age: '133'} )
      }
    )
```

```
      const json = await response.json();
      setRecords(json['Record']);
      console.warn(json['Message']);
      }
      catch(error){
      console.error(error);
      }
      }


////////////// DOCUMENT PICKER CODE ////////////////

const selectFile = async () => {

  // Opening Document Picker to select one file
  //////// file selection code begins here //////

  try {

    const res = await DocumentPicker.pick({
      // Provide which type of file you want user to pick
      type: [DocumentPicker.types.allFiles],

      // to select multiple files, use this property
      //allowMultiSelection: true

      // There can me more options as well
      // DocumentPicker.types.allFiles
      // DocumentPicker.types.images
      // DocumentPicker.types.plainText
      // DocumentPicker.types.audio
      // DocumentPicker.types.pdf
    });
    // Printing the log realted to the file
    //console.log('res : ' + JSON.stringify(res));

  //// ==== file selection code ends here ////

/// file uploading code begins here ///

const formData = new FormData();

    formData.append('file_attachment', {
                  uri:res[0].uri,
                  type:res[0].type,
                  name:res[0].name
            })
```

```
    let resp = await fetch(phpAppAddress + 'upload.php',
    {
      method: 'post',
      body: formData,
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })

  let response = await resp.json();
  if (response.status == 1) {
    console.warn('Upload Successful');
  }

//==== file uploading code ends here ====

  }

  catch (err) {

    // Handling any exception (If any)

    if (DocumentPicker.isCancel(err)) {
      // If user canceled the document selection
      console.warn('Canceled');
    } else {
      // For Unknown Error
      alert('Unknown Error: ' + JSON.stringify(err));
      throw err;
    }
  }
};

////////////////////////////////////////////////

  return (
    <View style={{marginTop:100}}>

    <Button
    title='Insert Data'
    onPress={() => insertData()}
    />

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

```
<Button
    title='Delete Data'
    onPress={() => deleteData()}
    />

<Button
title = "Upload file"
onPress={selectFile}
/>

<Button
    title='Get All Records'
    onPress={() => getAllRecords()}
    />

<Button title="select filtered data"
    onPress={() => getFilteredData()}
     />


<Button
    title='Get Single Response'
    onPress={() => getSingleResponse()}
    />

<Button
    title='Get Movies'
    onPress={() => getMovies()}
    />

<FlatList
         data={records}

        renderItem = {

            ({item}) =>
            <Text>{item.userid}  {item.uname}  {item.age}</Text>

    }
    />

    </View>
  );
};

export default App;
```

Create a new app in php, name the app folder as: **rnphpmysql**

Create a folder in it named: **rnphpmysql/uploads**

Add the following files in the app

opendb.php

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";  // set this field "" (empty quotes) if you have not set any password in mysql
$dbname = "rndb";
$e="";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// echo "Connection successful";
}
catch(PDOException $e)
    {
    echo $e->getMessage();
    }

?>
```

menu.php

```php
<table>
    <tr>
        <td>
            <a href="index.php">Home</a>
        </td>

        <td>
            <a href="insertrecord.php">Insert</a>
        </td>



        </tr>
```

```
    </table>
```

index.php

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">

<title>Untitled Document</title>

</head>

<body>
    <?php
    include("menu.php");
    include("opendb.php");

    $query = "select * from users";
    $stmt = $conn -> prepare($query);
    $stmt -> execute()
    ?>
    <br>
    <table border="1">

          <?php
          while($row = $stmt->fetch())
            {
              ?>
        <tr>
          <td><?php echo $row[0]; ?></td>
          <td><?php echo $row[1]; ?></td>
          <td><?php echo $row[2]; ?></td>
        </tr>
<?php

            }
          ?>


    </table>

    <?php
        $conn = NULL;
    ?>

</body>
</html>
```

insertrecord.php

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

</head>

<body>
    <?php
    include("menu.php");
    include("opendb.php");

    if(isset($_POST['btnsave']))
    {
        $name = $_POST['txtname'];
        $age = $_POST['txtage'];

    try {
            $query = "insert into users(uname, age) values(:uname, :age)";
            $stmt = $conn -> prepare($query);
            $stmt -> bindParam(':uname', $name);
            $stmt -> bindParam(':age', $age);
            $stmt -> execute();
    }
    catch(PDOException $e)
    {
            echo $e -> getMessage();
    }

    }
    ?>

    <form id="frm" name="frm" method="post" action="">
    <table border="1">

        <tr>
          <td>Name</td>
          <td>
          <input type="text" name="txtname" id="txtname"></td>
        </tr>
        <tr>
          <td>Age</td>
          <td>
```

```
          <input id="txtage" name="txtage" type="text" >
        </td>
        </tr>
        <tr>
          <td><input name="btnsave" type="submit" id="btnsave" title="Save"
value="Save"></td>
          <td> </td>
        </tr>

    </table>

    </form>

    <?php
        $conn = NULL;
    ?>

</body>
</html>
```

setdata.php

```
<?php
include("opendb.php");

        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);
        $name = $DecodedData['name'];
        $age = $DecodedData['age'];

        $Message = "Record inserted";


        try {
                        $query = "insert into users(uname, age) values(:uname,
:age)";
                        $stmt = $conn -> prepare($query);
                        $stmt -> bindParam(':uname', $name);
                        $stmt -> bindParam(':age', $age);
                        $stmt -> execute();
        }
        catch(PDOException $e)
        {
        $Message = $e -> getMessage();
        }

    $conn = NULL;
```

```
    $Response = array("Message" => $Message);
    echo json_encode($Response);

    ?>
```

updatedata.php

```
<?php
include("opendb.php");

        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);

        $id = $DecodedData['id'];
        $name = $DecodedData['name'];
        $age = $DecodedData['age'];

        $Message = "Record updated";

    try {
            $query = "update users set uname = :uname, age = :age where userid
= :userid";

            $stmt = $conn -> prepare($query);


            $stmt -> bindParam(':uname', $name);
            $stmt -> bindParam(':age', $age);
            $stmt -> bindParam(':userid', $id);
            $stmt -> execute();
    }
    catch(PDOException $e)
    {
        $Message = $e -> getMessage();
    }

    $conn = NULL;
    $Response = array("Message" => $Message);
    echo json_encode($Response);

    ?>
```

deletedata.php

```
<?php
include("opendb.php");
```

```
        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);

        $id = $DecodedData['id'];

        $Message = "Record delete";

        try {
                        $query = "delete from users where userid = :userid";

                        $stmt = $conn -> prepare($query);

            $stmt -> bindParam(':userid', $id);
            $stmt -> execute();
        }
        catch(PDOException $e)
        {
        $Message = $e -> getMessage();
        }

    $conn = NULL;
    $Response = array("Message" => $Message);
    echo json_encode($Response);

    ?>
```

getalldata.php

```
<?php
    include("opendb.php");

    $query = "select * from users";
    $stmt = $conn -> prepare($query);
    $stmt -> execute();

    $resultJSON = json_encode($stmt->fetchAll(PDO::FETCH_ASSOC));

    echo $resultJSON;
?>
```

selectfiltereddata.php

```
<?php
include("opendb.php");
```

```php
$EncodedData = file_get_contents('php://input');
$DecodedData = json_decode($EncodedData, true);
$age = $DecodedData['age'];


$Message = "No record found";

try{

$query = "select * from users where age > :age";
$stmt = $conn -> prepare($query);
$stmt -> bindParam(":age", $age);
$stmt -> execute();

$count = $stmt -> rowCount();

if( $count == 0)
  $Message = "No record found";
else
  $Message = "$count rows returned";

$resultJSON = $stmt -> fetchAll(PDO::FETCH_ASSOC);
}
catch(PDOException $e)
{
  $Message = $e -> getMessage();
}
finally{
    $conn = NULL;

    $Response = array("Record" => $resultJSON, "Message" => $Message);
echo json_encode($Response);

}



?>
```

getresponse.php

```php
<?php
    $Message = "This is a response";
    $Response = array("Message" => $Message);
    echo json_encode($Response);
    ?>
```

upload.php

```php
<?php
 if(!empty($_FILES['file_attachment']['name']))
 {

    $target_dir = "uploads/";
    if (!file_exists($target_dir))
    {
      mkdir($target_dir, 0777);
    }
    $target_file =
      $target_dir . basename($_FILES["file_attachment"]["name"]);
    $imageFileType =
      strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
    // Check if file already exists
    if (file_exists($target_file)) {
      echo json_encode(
         array(
           "status" => 0,
           "data" => array()
           ,"msg" => "Sorry, file already exists."
         )
      );
      die();
    }
    // Check file size
    if ($_FILES["file_attachment"]["size"] > 50000000) {
      echo json_encode(
         array(
           "status" => 0,
           "data" => array(),
           "msg" => "Sorry, your file is too large."
         )
       );
      die();
    }
    if (
      move_uploaded_file(
        $_FILES["file_attachment"]["tmp_name"], $target_file
      )
    ) {
      echo json_encode(
        array(
          "status" => 1,
          "data" => array(),
          "msg" => "The file " .
                   basename( $_FILES["file_attachment"]["name"]) .
                   " has been uploaded."));
    } else {
```

```
      echo json_encode(
        array(
          "status" => 0,
          "data" => array(),
          "msg" => "Sorry, there was an error uploading your file."
        )
      );
    }



}
?>
```

CONNECTING WITH NOSQL FIRESTORE DATABASE

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:

rooms

roomA

`name : "my chat room"`

roomB

`...`

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()),
    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.
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;
```

Storing and reading a subcollection

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

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

import {db} from './firestoreconnection';

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

async function writeData() {


const docRef = doc(db, "students", "15");

  await setDoc(
    docRef, {
      name: "Ali",
    Address: {
    country: "Pakistan",
    Location: {st: 45, Lane: 11}
  }
});


  // Add a document to the "subjects" subcollection
  const subjectsCollectionRef = collection(docRef, 'subjects');

  await addDoc(subjectsCollectionRef, {
    subjectid: 's1',
    subjectname: 'English',
  });


  await addDoc(subjectsCollectionRef, {
    subjectid: 's2',
    subjectname: 'Urdu',
  });


}
```

```
const readData = async() => {


    const docRef = doc(db, "students", "15");

      const result = await getDoc(docRef);


     console.log(
      result.data().name, " ",
      result.data().Address.country, " ",
      result.data().Address.Location.Lane);

      const subjectsCollectionRef = collection(docRef, 'subjects');

     const subjectsSnapshot = await getDocs(subjectsCollectionRef);
      subjectsSnapshot.forEach(doc => console.log(doc.data()));




    }



const App = () => {
  return (

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

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

Transactions and batched writes

Cloud Firestore supports atomic operations for reading and writing data. In a set of atomic operations, either all of the operations succeed, or none of them are applied. There are two types of atomic operations in Cloud Firestore:

Transactions: a transaction is a set of read and write operations on one or more documents.

Batched Writes: a batched write is a set of write operations on one or more documents.

Each transaction or batch of writes can write to a maximum of 500 documents. For additional limits related to writes, see Quotas and Limits.

Updating data with transactions

Using the Cloud Firestore client libraries, you can group multiple operations into a single transaction. Transactions are useful when you want to update a field's value based on its current value, or the value of some other field.

A transaction consists of any number of get() operations followed by any number of write operations such as set(), update(), or delete(). In the case of a concurrent edit, Cloud Firestore runs the entire transaction again. For example, if a transaction reads documents and another client modifies any of those documents, Cloud Firestore retries the transaction. This feature ensures that the transaction runs on up-to-date and consistent data.

Transactions never partially apply writes. All writes execute at the end of a successful transaction.

When using transactions, note that:

Read operations must come before write operations.

A function calling a transaction (transaction function) might run more than once if a concurrent edit affects a document that the transaction reads.

Transaction functions should not directly modify application state.

Transactions will fail when the client is offline.

The following example shows how to create and run a transaction:

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

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

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


async function updateData() {
  // a transaction to increase population of a city

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

   try {
  await runTransaction(db, async (transaction) => {
    const sfDoc = await transaction.get(sfDocRef);
    if (!sfDoc.exists()) {
      throw "Document does not exist!";
    }

    const newPopulation = sfDoc.data().population + 1;
    transaction.update(sfDocRef, { population: newPopulation });
  });
  console.log("Transaction successfully committed!");
} catch (e) {
  console.log("Transaction failed: ", e);
}


}

const App = () => {
  return (

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

   <Button
```

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


   </View>
   );
}

export default App;
```

Passing information out of transactions

Do not modify application state inside of your transaction functions. Doing so will introduce concurrency issues, because transaction functions can run multiple times and are not guaranteed to run on the UI thread. Instead, pass information you need out of your transaction functions. The following example builds on the previous example to show how to pass information out of a transaction:

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

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

import {db} from './firestoreconnection';

async function updateData(setPopulation) {
  // a transaction to increase population of a city

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

    try {
      const newPopulation = await runTransaction(db, async (transaction) => {
        const sfDoc = await transaction.get(sfDocRef);
        if (!sfDoc.exists()) {
          throw "Document does not exist!";
        }

        const newPop = sfDoc.data().population + 1;
        if (newPop <= 1000000) {
          transaction.update(sfDocRef, { population: newPop });
          return newPop;
        } else {
          return Promise.reject("Sorry! Population is too big");
        }
      });
```

```
      console.log("Population increased to ", newPopulation);
      setPopulation(newPopulation);
    } catch (e) {
      // This will be a "population is too big" error.
      console.error(e);
    }

}

const App = () => {
  const [population, setPopulation] = useState(0);
  return (

<View
style = {{marginTop: 50}}
>
<Text>{population}</Text>

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

   </View>
   );
}

export default App;
```

Transaction failure

A transaction can fail for the following reasons:

The transaction contains read operations after write operations. Read operations must always come before any write operations.

The transaction read a document that was modified outside of the transaction. In this case, the transaction automatically runs again. The transaction is retried a finite number of times.

The transaction exceeded the maximum request size of 10 MiB.

Transaction size depends on the sizes of documents and index entries modified by the transaction. For a delete operation, this includes the size of the target document and the sizes of the index entries deleted in response to the operation.

A failed transaction returns an error and does not write anything to the database. You do not need to roll back the transaction; Cloud Firestore does this automatically.

Batched writes

If you do not need to read any documents in your operation set, you can execute multiple write operations as a single batch that contains any combination of set(), update(), or delete() operations. A batch of writes completes atomically and can write to multiple documents. The following example shows how to build and commit a write batch:

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

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

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


async function updateData() {

// Get a new write batch
const batch = writeBatch(db);

// Set the value of 'NYC'
const nycRef = doc(db, "cities", "NYC");
batch.set(nycRef, {name: "New York City"});

// Update the population of 'SF'
const sfRef = doc(db, "cities", "SF");
batch.update(sfRef, {"population": 1000000});

// Delete the city 'LA'
const laRef = doc(db, "cities", "LA");
batch.delete(laRef);

// Commit the batch
await batch.commit();

}
```

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

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

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


   </View>
   );
}

export default App;
```

A batched write can contain up to 500 operations. Each operation in the batch counts separately towards your Cloud Firestore usage.

Like transactions, batched writes are atomic. Unlike transactions, batched writes do not need to ensure that read documents remain un-modified which leads to fewer failure cases. They are not subject to retries or to failures from too many retries. Batched writes execute even when the user's device is offline.

Delete Documents

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

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

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


async function deleteData() {



await deleteDoc(doc(db, "cities", "ATD1"));
```

```
}

const App = () => {
  return (

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

   <Button
   title='Delete Data'
   onPress={ () => deleteData()}
    />


   </View>
   );
}

export default App;
```

Warning: Deleting a document does not delete its subcollections!

When you delete a document, Cloud Firestore does not automatically delete the documents within its subcollections. You can still access the subcollection documents by reference. For example, you can access the document at path /mycoll/mydoc/mysubcoll/mysubdoc even if you delete the ancestor document at /mycoll/mydoc.

Non-existent ancestor documents appear in the console, but they do not appear in query results and snapshots.

If you want to delete a document and all the documents within its subcollections, you must do so manually. For more information, see Delete Collections.

Delete fields

To delete specific fields from a document, use the FieldValue.delete() method when you update a document:

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

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

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


async function deleteData() {



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

  // Remove the 'capital' field from the document
  await updateDoc(cityRef, {
      country: deleteField()
  });

}

const App = () => {
  return (

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

   <Button
   title='Delete Data'
   onPress={ () => deleteData()}
    />


   </View>
   );
}

export default App;
```

Get data with Cloud Firestore

Example data

To get started, write some data about cities so we can look at different ways to read it back:

```
import React from 'react';
```

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

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

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

async function writeData() {
  const citiesRef = collection(db, "cities");

await setDoc(doc(citiesRef, "SF"), {
    name: "San Francisco", state: "CA", country: "USA",
    capital: false, population: 860000,
    regions: ["west_coast", "norcal"] });
await setDoc(doc(citiesRef, "LA"), {
    name: "Los Angeles", state: "CA", country: "USA",
    capital: false, population: 3900000,
    regions: ["west_coast", "socal"] });
await setDoc(doc(citiesRef, "DC"), {
    name: "Washington, D.C.", state: null, country: "USA",
    capital: true, population: 680000,
    regions: ["east_coast"] });
await setDoc(doc(citiesRef, "TOK"), {
    name: "Tokyo", state: null, country: "Japan",
    capital: true, population: 9000000,
    regions: ["kanto", "honshu"] });
await setDoc(doc(citiesRef, "BJ"), {
    name: "Beijing", state: null, country: "China",
    capital: true, population: 21500000,
    regions: ["jingjinji", "hebei"] });
}

async function readData() {

}




const App = () => {
  return (

<View
style = {{marginTop: 50}}
>
   <Button
   title='Read Data'
   onPress={ () => readData()}
    />
```

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


   </View>
   );
}

export default App;
```

Get a document

The following example shows how to retrieve the contents of a single document using get():

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

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

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


async function readData() {

const docRef = doc(db, "cities", "SF");
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;
```

Showing document attributes in Text box

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

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

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




const App = () => {
   const [country, setCountry] = useState('');

   async function readData()
   {
      const docRef = doc(db, "cities", "SF");
      const docSnap = await getDoc(docRef);

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

```
  return (

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

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

<Text>Data shown in a textbox: {country}</Text>

    </View>
   );
}

export default App;
```

Custom objects

The previous example retrieved the contents of the document as a map, but in some languages it's often more convenient to use a custom object type. In Add Data, you defined a City class that you used to define each city. You can turn your document back into a City object:

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

import {doc, getDoc, 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);
   }
};



const App = () => {


   async function readData()
   {
      const ref = doc(db, "cities", "LA").withConverter(cityConverter);
      const docSnap = await getDoc(ref);
      if (docSnap.exists()) {
      // Convert to City object
      const city = docSnap.data();
      // Use a City instance method
      console.log(city.toString());
      } else {
      console.log("No such document!");
      }
   }

   async function writeData()
   {

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

    }




  return (
```

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

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

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



    </View>
   );
}

export default App;
```

Get multiple documents from a collection

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

import {query, where, collection, getDocs } from 'firebase/firestore';

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


const App = () => {


   async function readData()
   {
      const q = query(collection(db, "cities"), where("capital", "==", true));

      const querySnapshot = await getDocs(q);
      querySnapshot.forEach((doc) => {
      // doc.data() is never undefined for query doc snapshots
      console.log(doc.id, " => ", doc.data());
```

```
});
      }

  return (

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

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

    </View>
   );
}

export default App;
```

Displaying a collection with multiple records in a flat list

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

import {query, where,  collection, getDocs } from 'firebase/firestore';

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

const App = () => {
   const [cities, setCities] = useState([]);

   async function readData()
   {
      setCities([]);
     const mycities = [];
     const q = query(collection(db, "cities"), where("capital", "==", true) );
     const querySnapshot = await getDocs(q);
     querySnapshot.forEach( (city) => {mycities.push({key: city.id, name:
city.data().name} )})

     setCities(mycities);
```

```
   }
  return (

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

         <FlatList
         data={cities}

        renderItem = {

            ({item}) =>
            <Text>{item.key} {item.name}</Text>

    }
    />

    </View>
   );
}

export default App;
```

Get all documents in a collection

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

import {query, where, collection, getDocs } from 'firebase/firestore';

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


const App = () => {


   async function readData()
   {
      const querySnapshot = await getDocs(collection(db, "cities"));
```

```
      querySnapshot.forEach((doc) => {
      // doc.data() is never undefined for query doc snapshots
      console.log(doc.id, " => ", doc.data());
      });
   }

  return (

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

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

    </View>
   );
}

export default App;
```

Get realtime updates with Cloud Firestore

You can listen to a document with the onSnapshot() method. An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document. Then, each time the contents change, another call updates the document snapshot.

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

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

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


const App = () => {


   async function readData()
   {
      const unsub = onSnapshot(doc(db, "cities", "SF"), (doc) => {
         console.log("Current data: ", doc.data());
     });
   }
```

```
  return (

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

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

    </View>
   );
}

export default App;
```

Unsubscribe from real-time updates

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

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

import { db } from './firestoreconnection';

var unsubscribe = () => {}

const App = () => {
  const [city, setCity] = useState(null);

  const readData = () => {
    const unsub = onSnapshot(doc(db, 'cities', 'SF'), (item) => {
      setCity(item.data());
      //console.log('Current data: ', doc.data());
    });

    // Store the unsubscribe function in the state

  unsubscribe = unsub;
  };
```

```
  const unsubscribeFromData = () => {
    unsubscribe();
  };

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

      <Text>{city?.name}</Text>
      <Text>{city?.population}</Text>
      <Text>{city?.country}</Text>
      <Button title="Read Data" onPress={() => readData()} />
      <Button title="Unsubscribe" onPress={() => unsubscribeFromData()} />
    </View>
  );
};

export default App;
```

Listen to multiple documents in a collection

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

import { collection, query, where, onSnapshot } from 'firebase/firestore';

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


const App = () => {


   async function readData()
   {
      const q = query(collection(db, "cities"), where("state", "==", "CA"));
const unsubscribe = onSnapshot(q, (querySnapshot) => {
  const cities = [];
  querySnapshot.forEach((doc) => {
      cities.push(doc.data().name);
  });
  console.log("Current cities in CA: ", cities.join(", "));
});
   }
```

```
  return (

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

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

    </View>
   );
}

export default App;
```

Displaying a flatlist showing real-time updates of documents in a collection

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

import {query, where,  collection, getDocs, onSnapshot } from
'firebase/firestore';

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


const App = () => {

   const [loading, setLoading] = useState(true); // Set loading to true on
component mount
   const [users, setUsers] = useState([]); // Initial empty array of users

   useEffect(() => {

      const q = query(collection(db, "users"));
const subscriber = onSnapshot(q, (querySnapshot) => {

          // see next step
```

```
       const users = [];

      querySnapshot.forEach(
         documentSnapshot => { users.push({ ...documentSnapshot.data(), key:
documentSnapshot.id, });
      });

      setUsers(users);
      setLoading(false);


        });

      // Unsubscribe from events when no longer in use
      return () => subscriber();
    }, []);

    if (loading) {
      return <ActivityIndicator />;
    }




  return (

<View
style = {{marginTop: 50}}
>
<FlatList
         data={users}

        renderItem = {

            ({item}) =>
            <Text>{item.key}    {item.name}</Text>

    }
    />



    </View>
   );
}

export default App;
```

Perform simple and compound queries in Cloud Firestore

Simple queries

The following query returns all cities with state CA:

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

import {  collection, query, where, getDocs  } from 'firebase/firestore';

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


const App = () => {


   async function readData()
   {
      const citiesRef = collection(db, "cities");

// Create a query against the collection.
const q = query(citiesRef, where("state", "==", "CA"));

const querySnapshot = await getDocs(q);
querySnapshot.forEach((doc) => {
// doc.data() is never undefined for query doc snapshots
console.log(doc.id, " => ", doc.data());
});


   }

  return (

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

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

    </View>
   );
```

```
}

export default App;
```

Query operators

The where() method takes three parameters: a field to filter on, a comparison operator, and a value. Cloud Firestore supports the following comparison operators:

< less than

<= less than or equal to

== equal to

> greater than

>= greater than or equal to

!= not equal to

array-contains

array-contains-any

in

not-in

Example

```
const stateQuery = query(citiesRef, where("state", "==", "CA"));
const populationQuery = query(citiesRef, where("population", "<", 100000));
const nameQuery = query(citiesRef, where("name", ">=", "San Francisco"));
```

Not equal (!=)

Use the not equal (!=) operator to return documents where the given field exists and does not match the comparison value. For example:

```
const notCapitalQuery = query(citiesRef, where("capital", "!=", false));
```

This query returns every city document where the capital field exists with a value other than false or null. This includes city documents where the capital field value equals true or any non-boolean value besides null.

This query does not return city documents where the capital field does not exist. **Not-equal (`!=`) and `not-in` queries exclude documents where the given field does not exist**.

A field exists when it's set to any value, including an empty string (""), null, and NaN (not a number). Note that null field values do not match != clauses, because x != null evaluates to undefined.

Warning: A **`!=`** query clause might match many documents in a collection. To control the number of results returned, use a limit clause or paginate your query.

Limitations

Note the following limitations for != queries:

- Only documents where the given field exists can match the query.
- You can't combine not-in and != in a compound query.
- In a compound query, range (<, <=, >, >=) and not equals (!=, not-in) comparisons must all filter on the same field.

Array membership

You can use the array-contains operator to filter based on array values. For example:

```
import { query, where } from "firebase/firestore";
const q = query(citiesRef, where("regions", "array-contains", "west_coast"));
```

This query returns every city document where the regions field is an array that contains west_coast. If the array has multiple instances of the value you query on, the document is included in the results only once.

You can use at most one array-contains clause per query. You can't combine array-contains with array-contains-any.

in, not-in, and array-contains-any

Use the in operator to combine up to 10 equality (==) clauses on the same field with a logical OR. An in query returns documents where the given field matches any of the comparison values. For example:

```
import { query, where } from "firebase/firestore";

const q = query(citiesRef, where('country', 'in', ['USA', 'Japan']));
```

This query returns every city document where the country field is set to USA or Japan. From the example data, this includes the SF, LA, DC, and TOK documents.

not-in

Use the not-in operator to combine up to 10 not equal (!=) clauses on the same field with a logical AND. A not-in query returns documents where the given field exists, is not null, and does not match any of the comparison values. For example:

```
import { query, where } from "firebase/firestore";

const q = query(citiesRef, where('country', 'not-in', ['USA', 'Japan']))
```

This query returns every city document where the country field exists and is not set to USA, Japan, or null. From the example data, this includes the London and Hong Kong documents.

`not-in` queries exclude documents where the given field does not exist. A field exists when it's set to any value, including an empty string (""), null, and NaN (not a number). Note that x != null evaluates to undefined. A not-in query with null as one of the comparison values does not match any documents.

Warning: A **`not-in`** query clause might match many documents in a collection. To control the number of results returned, use a limit clause or paginate your query.

array-contains-any

Use the array-contains-any operator to combine up to 10 array-contains clauses on the same field with a logical OR. An array-contains-any query returns documents where the given field is an array that contains one or more of the comparison values:

```
import { query, where } from "firebase/firestore";

const q = query(citiesRef,
  where('regions', 'array-contains-any', ['west_coast', 'east_coast']));
```

This query returns every city document where the regions field is an array that contains west_coast or east_coast. From the example data, this includes the SF, LA, and DC documents.

Results from array-contains-any are de-duped. Even if a document's array field matches more than one of the comparison values, the result set includes that document only once.

array-contains-any always filters by the array data type. For example, the query above would not return a city document where instead of an array, the regions field is the string west_coast.

You can use an array value as a comparison value for in, but unlike array-contains-any, the clause matches for an exact match of array length, order, and values. For example:

```
import { query, where } from "firebase/firestore";

const q = query(citiesRef, where('regions', 'in', [['west_coast',
'east_coast']]));
```

This query returns every city document where the regions field is an array that contains exactly one element of either west_coast or east_coast. From the example data, only the DC document qualifies with its regions field of ["east_coast"].
The SF document, however, does not match because its regions field is ["west_coast", "norcal"].

Limitations

Note the following limitations for in, not-in, and array-contains-any:

- in, not-in, and array-contains-any support up to 10 comparison values.
- You can use at most one array-contains clause per query. You can't combine array-contains with array-contains-any.
- You can use at most one in, not-in, or array-contains-any clause per query. You can't combine these operators in the same query.
- You can't combine not-in with not equals !=.
- You can't order your query by a field included in an equality (==) or in clause.

Compound queries

You can chain multiple equality operators (== or array-contains) methods to create more specific queries (logical AND). However, you must create a composite index to combine equality operators with the inequality operators, <, <=, >, and !=.

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

import {query, where, collection, getDocs } from 'firebase/firestore';

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

const App = () => {


   async function readData()
```

```
   {
      const q = query(collection(db, "cities"), where("state", "==", "CA"),
where("name", "==", "Los Angeles"));

      const querySnapshot = await getDocs(q);
      querySnapshot.forEach((doc) => {
      // doc.data() is never undefined for query doc snapshots
      console.log(doc.id, " => ", doc.data());
});
      }

  return (

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

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

    </View>
   );
}

export default App;
```

You can perform range (<, <=, >, >=) or not equals (`!=`) comparisons only on a single field, and you can include at most one `array-contains` or `array-contains-any` clause in a compound query:

```
import { query, where } from "firebase/firestore";

const q1 = query(citiesRef, where("state", ">=", "CA"), where("state", "<=",
"IN"));
const q2 = query(citiesRef, where("state", "==", "CA"), where("population",
">", 1000000));
```

Collection group queries

A collection group consists of all collections with the same ID. By default, queries retrieve results from a single collection in your database. Use a collection group query to retrieve documents from a collection group instead of from a single collection.

For example, you can create a landmarks collection group by adding a landmarks subcollection to each city:

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

import {collection, addDoc } from 'firebase/firestore';

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

const App = () => {


   async function writeData()
   {
      const citiesRef = collection(db, 'cities');

      await Promise.all([
          addDoc(collection(citiesRef, 'SF', 'landmarks'), {
              name: 'Golden Gate Bridge',
              type: 'bridge'
          }),
          addDoc(collection(citiesRef, 'SF', 'landmarks'), {
              name: 'Legion of Honor',
              type: 'museum'
          }),
          addDoc(collection(citiesRef, 'LA', 'landmarks'), {
              name: 'Griffith Park',
              type: 'park'
          }),
          addDoc(collection(citiesRef, 'LA', 'landmarks'), {
              name: 'The Getty',
              type: 'museum'
          }),
          addDoc(collection(citiesRef, 'DC', 'landmarks'), {
              name: 'Lincoln Memorial',
              type: 'memorial'
          }),
          addDoc(collection(citiesRef, 'DC', 'landmarks'), {
              name: 'National Air and Space Museum',
              type: 'museum'
          }),
          addDoc(collection(citiesRef, 'TOK', 'landmarks'), {
              name: 'Ueno Park',
              type: 'park'
```

```
          }),
          addDoc(collection(citiesRef, 'TOK', 'landmarks'), {
              name: 'National Museum of Nature and Science',
              type: 'museum'
          }),
          addDoc(collection(citiesRef, 'BJ', 'landmarks'), {
              name: 'Jingshan Park',
              type: 'park'
          }),
          addDoc(collection(citiesRef, 'BJ', 'landmarks'), {
              name: 'Beijing Ancient Observatory',
              type: 'museum'
          })
      ]);
      }

  return (

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

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

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

    </View>
   );
}

export default App;
```

We can use the simple and compound query described earlier to query a single city's landmarks subcollection, but you might also want to retrieve results from every city's landmarks subcollection at once.

The landmarks collection group consists of all collections with the ID landmarks, and you can query it using a collection group query. For example, this collection group query retrieves all museum landmarks across all cities:

Query limitations

The following list summarizes Cloud Firestore query limitations:

- Cloud Firestore provides limited support for logical OR queries. The in, and array-contains-any operators support a logical OR of up to 10 equality (==) or array-contains conditions on a single field. For other cases, create a separate query for each OR condition and merge the query results in your app.
- In a compound query, range (<, <=, >, >=) and not equals (!=, not-in) comparisons must all filter on the same field.
- You can use at most one array-contains clause per query. You can't combine array-contains with array-contains-any.
- You can use at most one in, not-in, or array-contains-any clause per query. You can't combine in , not-in, and array-contains-any in the same query.
- You can't order your query by a field included in an equality (==) or in clause.
- The sum of filters, sort orders, and parent document path (1 for a subcollection, 0 for a root collection) in a query cannot exceed 100.

Order and limit data

By default, a query retrieves all documents that satisfy the query in ascending order by document ID. You can specify the sort order for your data using orderBy(), and you can limit the number of documents retrieved using limit()

For example, you could query for the first 3 cities alphabetically with:

```
import { query, orderBy, limit } from "firebase/firestore";

const q = query(citiesRef, orderBy("name"), limit(3));
```

You could also sort in descending order to get the *last* 3 cities

```
import { query, orderBy, limit } from "firebase/firestore";

const q = query(citiesRef, orderBy("name", "desc"), limit(3));
```

You can also order by multiple fields. For example, if you wanted to order by state, and within each state order by population in descending order:

```
import { query, orderBy } from "firebase/firestore";

const q = query(citiesRef, orderBy("state"), orderBy("population", "desc"));
```

You can combine `where()` filters with `orderBy()` and `limit()`. In the following example, the queries define a population threshold, sort by population in ascending order, and return only the first few results that exceed the threshold:

```
import { query, where, orderBy, limit } from "firebase/firestore";

const q = query(citiesRef, where("population", ">", 100000),
orderBy("population"), limit(2));
```

However, if you have a filter with a range comparison (<, <=, >, >=), your first ordering must be on the same field, see the list of `orderBy()` limitations below.

Limitations

Note the following restrictions for orderBy() clauses:

- An orderBy() clause also filters for existence of the given fields. The result set will not include documents that do not contain the given fields.
- If you include a filter with a range comparison (<, <=, >, >=), your first ordering must be on the same field:

  Valid: Range filter and orderBy on the same field

```
import { query, where, orderBy } from "firebase/firestore";

const q = query(citiesRef, where("population", ">", 100000),
orderBy("population"));
```

Invalid: Range filter and first orderBy on different fields

```
import { query, where, orderBy } from "firebase/firestore";

const q = query(citiesRef, where("population", ">", 100000),
orderBy("country"));
```

Use the count() aggregation

The following count() aggregation returns the total number of cities in the cities collection.

```
const coll = collection(db, "cities");
const snapshot = await getCountFromServer(coll);
console.log('count: ', snapshot.data().count);
```

The `count()` aggregation takes into account any filters on the query and any `limit` clauses. For example, the following aggregation returns a count of the number of cities where `state` equals `CA`.

```
const coll = collection(db, "cities");
const query_ = query(coll, where('state', '==', 'CA'));
const snapshot = await getCountFromServer(query_);
console.log('count: ', snapshot.data().count);
```

Paginate data with query cursors

Add a simple cursor to a query

Use the startAt() or startAfter() methods to define the start point for a query. The startAt() method includes the start point, while the startAfter() method excludes it.

For example, if you use startAt(A) in a query, it returns the entire alphabet. If you use startAfter(A) instead, it returns B-Z.

```
import { query, orderBy, startAt } from "firebase/firestore";

const q = query(citiesRef, orderBy("population"), startAt(1000000));
```

Similarly, use the endAt() or endBefore() methods to define an end point for your query results.

```
import { query, orderBy, endAt } from "firebase/firestore";

const q = query(citiesRef, orderBy("population"), endAt(1000000));
```

Use a document snapshot to define the query cursor

You can also pass a document snapshot to the cursor clause as the start or end point of the query cursor. The values in the document snapshot serve as the values in the query cursor.

For example, take a snapshot of a "San Francisco" document in your data set of cities and populations. Then, use that document snapshot as the start point for your population query cursor. Your query will return all the cities with a population larger than or equal to San Francisco's, as defined in the document snapshot.

```
import { collection, doc, getDoc, query, orderBy, startAt } from
"firebase/firestore";
const citiesRef = collection(db, "cities");

const docSnap = await getDoc(doc(citiesRef, "SF"));

// Get all cities with a population bigger than San Francisco
const biggerThanSf = query(citiesRef, orderBy("population"),
startAt(docSnap));
// ...
```

Paginate a query

Paginate queries by combining query cursors with the limit() method. For example, use the last document in a batch as the start of a cursor for the next batch.

```
import { collection, query, orderBy, startAfter, limit, getDocs } from
"firebase/firestore";

// Query the first page of docs
const first = query(collection(db, "cities"), orderBy("population"),
limit(25));
const documentSnapshots = await getDocs(first);

// Get the last visible document
const lastVisible = documentSnapshots.docs[documentSnapshots.docs.length-1];
console.log("last", lastVisible);

// Construct a new query starting at this document,
// get the next 25 cities.
const next = query(collection(db, "cities"),
    orderBy("population"),
    startAfter(lastVisible),
    limit(25));
```

Set cursor based on multiple fields

When using a cursor based on a field value (not a DocumentSnapshot), you can make the cursor position more precise by adding additional fields. This is particularly useful if your data set includes multiple documents that all have the same value for your cursor field, making the cursor's position ambiguous. You can add additional field values to your cursor to further specify the start or end point and reduce ambiguity.

For example, in a data set containing all the cities named "Springfield" in the United States, there would be multiple start points for a query set to start at "Springfield":

Cities	
Name	State
Springfield	Massachusetts
Springfield	Missouri
Springfield	Wisconsin

To start at a specific Springfield, you could add the state as a secondary condition in your cursor clause.

```
// Will return all Springfields
import { collection, query, orderBy, startAt } from "firebase/firestore";
const q1 = query(collection(db, "cities"),
   orderBy("name"),
   orderBy("state"),
   startAt("Springfield"));

// Will return "Springfield, Missouri" and "Springfield, Wisconsin"
const q2 = query(collection(db, "cities"),
   orderBy("name"),
   orderBy("state"),
   startAt("Springfield", "Missouri"));
```

Access data offline

Cloud Firestore supports offline data persistence. This feature caches a copy of the Cloud Firestore data that your app is actively using, so your app can access the data when the device is offline. You can write, read, listen to, and query the cached data. When the device comes back online, Cloud Firestore synchronizes any local changes made by your app to the Cloud Firestore backend.

CAMERA HANDLING

Camera handling in react native

For expo apps, use (recommended):

https://docs.expo.dev/versions/latest/sdk/camera/

For CLI based apps use:

https://react-native-vision-camera.com/docs/guides

Install the library using following command.

```
npm i react-native-vision-camera
```

and need to install this library:

https://github.com/react-native-cameraroll/react-native-cameraroll

Open your project's `AndroidManifest.xml` (located in android -> app -> src -> main) and add the following lines inside the `<manifest>` tag:

```
<uses-permission android:name="android.permission.CAMERA" />

<!-- optionally, if you want to record audio: -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />

<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
  <uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="32" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

App.js

```
import React,{useEffect, useRef, useState} from 'react';
import { Text, View, StyleSheet, Linking, TouchableOpacity, Alert} from
'react-native';
import {Camera, useCameraDevice, useCameraFormat, useCameraPermission,
useMicrophonePermission } from 'react-native-vision-camera';
import { CameraRoll } from '@react-native-camera-roll/camera-roll';

const App = () => {

const [cameraType, setCameraType] = useState('back');
const [recordingStatus, setRecordingStatus] = useState(false);

const device = useCameraDevice(cameraType);
```

```
const format = useCameraFormat(device, [

{ videoResolution: { width: 1280, height: 720 } },
  { fps: 30 },
  { pixelFormat: 'native' }

])


const {hasPermission, requestPermission} = useCameraPermission();
const {hasPermission: microphonePermission, requestPermission:
requestMicrophonePermission} = useMicrophonePermission();


const camera = useRef<Camera>(null)



useEffect( ()=> {

  if(!hasPermission) {
   requestPermission()
  }

  if(!microphonePermission) {
    requestMicrophonePermission()
  }
}, [hasPermission, microphonePermission])



if( !hasPermission || !microphonePermission)
{
return <Text>Please manually set camera and voice recording and restart the
app</Text>

}


console.log("Camera: " + hasPermission + " " + "Microphone: " +
microphonePermission);

if(!device) {
  return <Text>Camera device not found</Text>
}

if(!camera) {
  return <Text>Camera is null</Text>
}
```

```
async function PhotoFunction() {
  const file = await camera.current?.takePhoto();
  await CameraRoll.save(`file://${file.path}`, { type: 'photo' })
}

async function RecordFunction() {

   setRecordingStatus(true);

  if(!camera.current) {
    return;
  }

  camera.current?.startRecording({

    onRecordingFinished: async (video) => {
      const path = video.path;
      await CameraRoll.save(`file://${path}`, {type: 'video'})
    },


    onRecordingError: (error) => console.error(error)
  })

}

async function StopRecordFunction() {

  setRecordingStatus(false);

  await camera.current?.stopRecording()

}



return(

<View style={[styles.container, {
      // Try setting `flexDirection` to `"row"`.
      flexDirection: "column"
    }]}>

      <View style={{ flex: 0.9, backgroundColor: "darkorange" }}>
      <Camera
  style={StyleSheet.absoluteFill}
  device={device}
```

```
  ref = {camera}
  isActive={true}
  format={format}
  photo={true}
  video={true}
  audio={true}
/>
</View>

      <View style={{ flex: 0.1, backgroundColor: "green" }} >
      <View style={{flexDirection: "row"}}>

        <TouchableOpacity
          style={styles.button}
          onPress={() => setCameraType(cameraType==='back'? 'front' :
'back')}>
          <Text style={styles.buttonText}>Flip</Text>
        </TouchableOpacity>

        <TouchableOpacity
          style={styles.button}
          onPress={() => PhotoFunction()}>
          <Text style={styles.buttonText}>Photo</Text>
        </TouchableOpacity>

        <TouchableOpacity
          style={[styles.button, {backgroundColor:
recordingStatus==true?'red': '#DDDDDD'}]}
          onPress={() => {recordingStatus == false ? RecordFunction() :
StopRecordFunction()}}>
          <Text style={styles.buttonText}>{recordingStatus==true ?
"Recording":"Record"}</Text>
        </TouchableOpacity>
      </View>
      </View>
    </View>


    )
  }



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

```
    button: {
      alignItems: 'center',
      padding: 15,
      margin: 5,
    },
    buttonText: {
      fontSize: 20,
      fontWeight:'bold',
    }

  });



export default App;
```

GETTING GPS COORDINATES

Geo Location in React Native

Install the library:

```
expo install expo-location
```

App.js

```
import React, { useEffect, useState } from 'react';
import { View, Text } from 'react-native';
import * as Location from 'expo-location';

export default function App() {
  const [location, setLocation] = useState(null);
  const [errorMsg, setErrorMsg] = useState(null);

  useEffect(() => {
    // Request permission to access the device's location
    (async () => {
      let { status } = await Location.requestForegroundPermissionsAsync();
      if (status !== 'granted') {
        setErrorMsg('Permission to access location was denied');
        return;
      }

      // Start getting location updates every 5 seconds
      const intervalId = setInterval(getLocation, 5000);
```

```
      // Cleanup function to clear the interval when the component unmounts
      return () => clearInterval(intervalId);
    })();
  }, []);

  const getLocation = async () => {
    try {
      let location = await Location.getCurrentPositionAsync({});
      setLocation(location);
      console.log('Location:', location);
    } catch (error) {
      setErrorMsg('Error getting location: ' + error.message);
    }
  };

  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      {errorMsg ? (
        <Text>{errorMsg}</Text>
      ) : (
        <Text>
          Latitude: {location ? location.coords.latitude : 'Loading...'}
{'\n'}
          Longitude: {location ? location.coords.longitude : 'Loading...'}
        </Text>
      )}
    </View>
  );
}
```

Here we used Immediate Invoked Function Expression.

Example

```
(() => {
  console.log('Ok');
})();
```

Is the equivalent of IIFE

```
(function() {
  console.log('Ok');
})();
```

And the possible reason why this works in Node.js but not in Chrome, is because its parser interprets it as a self-executing function, as this

```
function() { console.log('hello'); }();
```

works fine in Node.js. This is a function expression, and Chrome, Firefox, and most browsers interpret it this way. You need to invoke it manually.
The most widely accepted way to tell the parser to expect a function expression is just to wrap it in parens, because in JavaScript, parens can't contain statements. At this point, when the parser encounters the function keyword, it knows to parse it as a function expression and not a function declaration.

Regarding the **parametrized version**, this will work.

```
((n) => {
  console.log('Ok');
})()
```