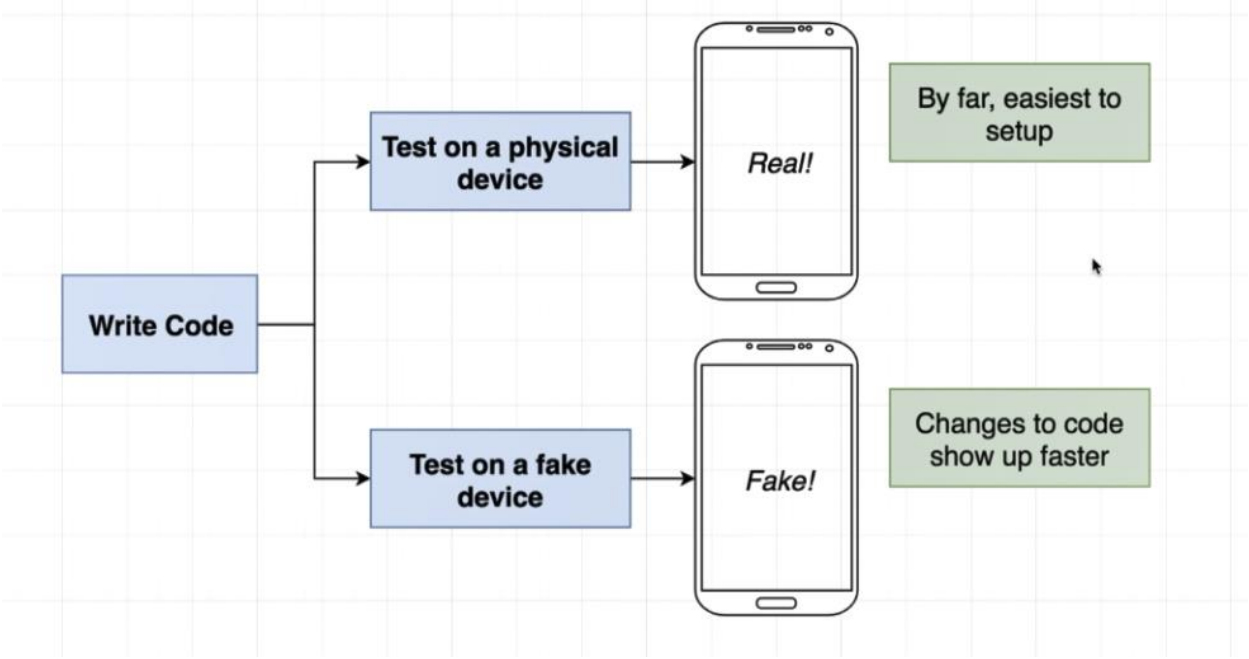
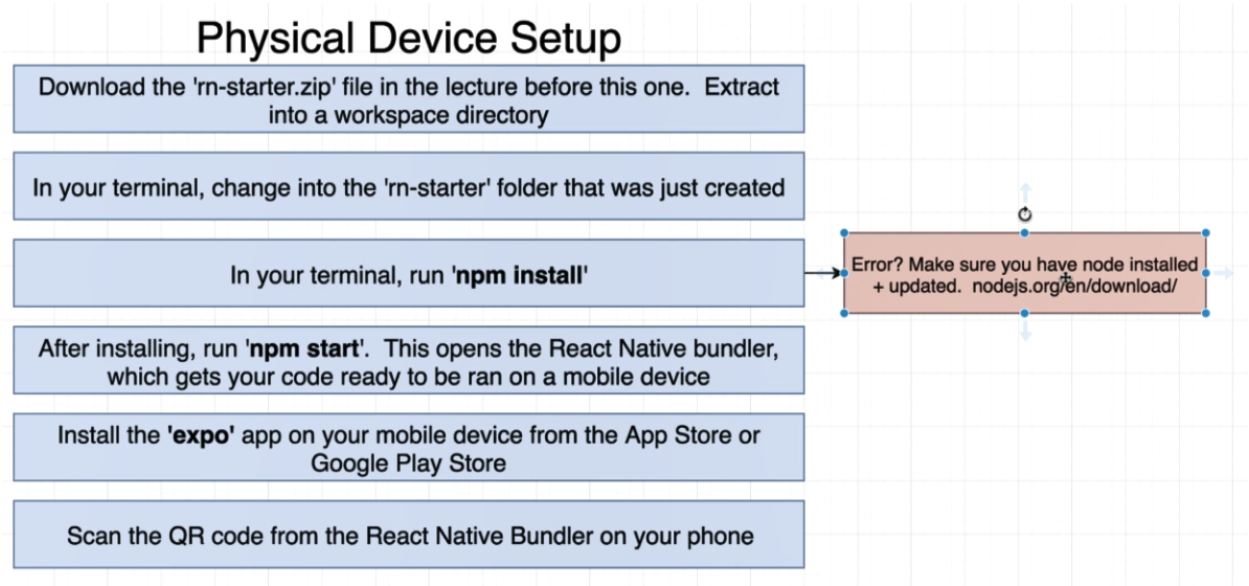
# React Native

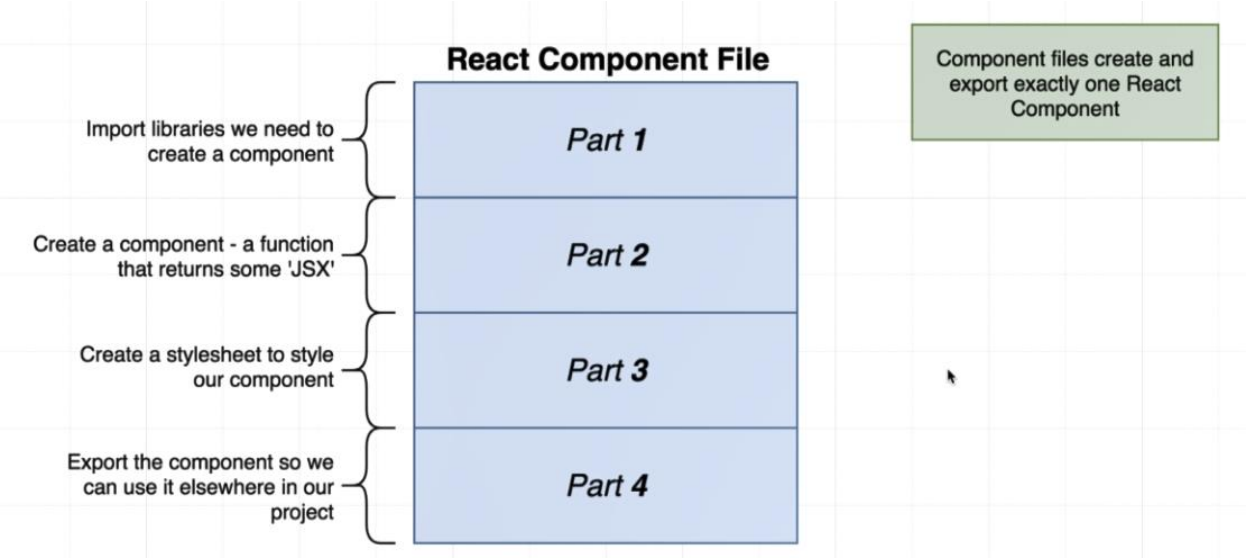
## 1. Getting Started



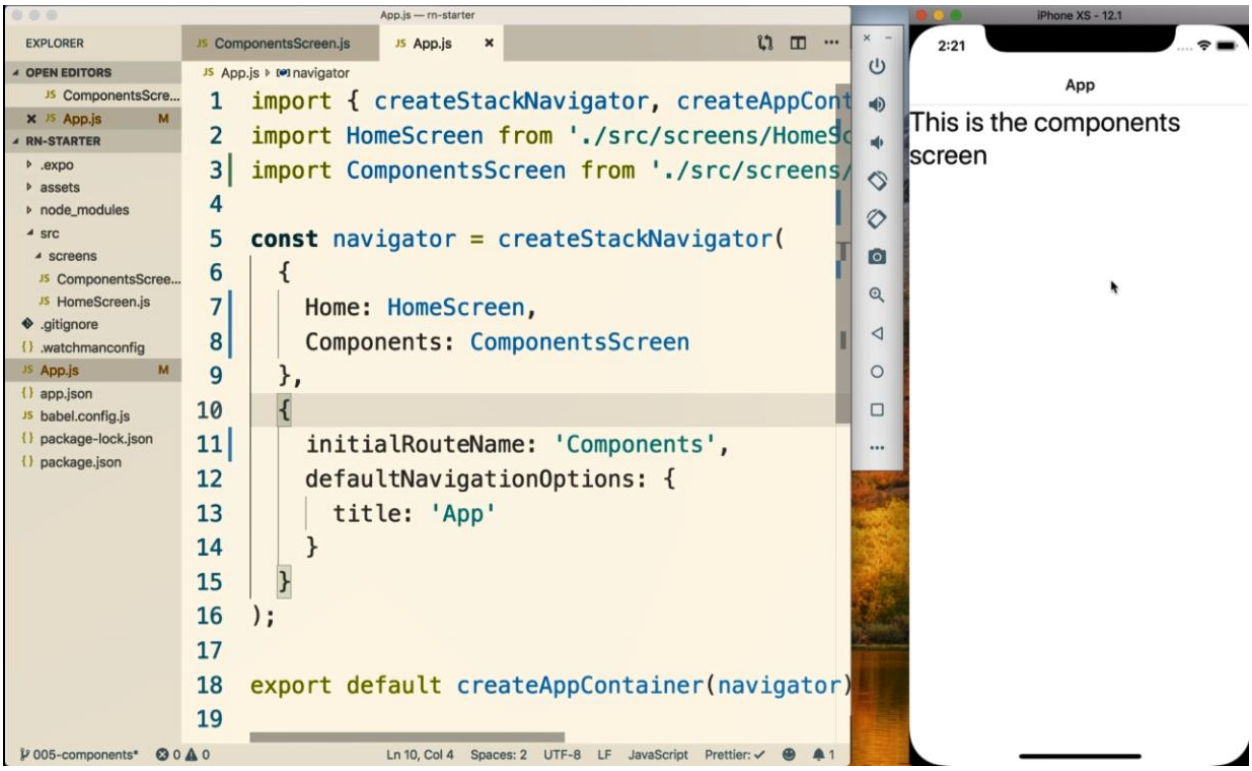



# 2. Working with Content

## 1. Overview of React Components




2. Showing a Custom Component



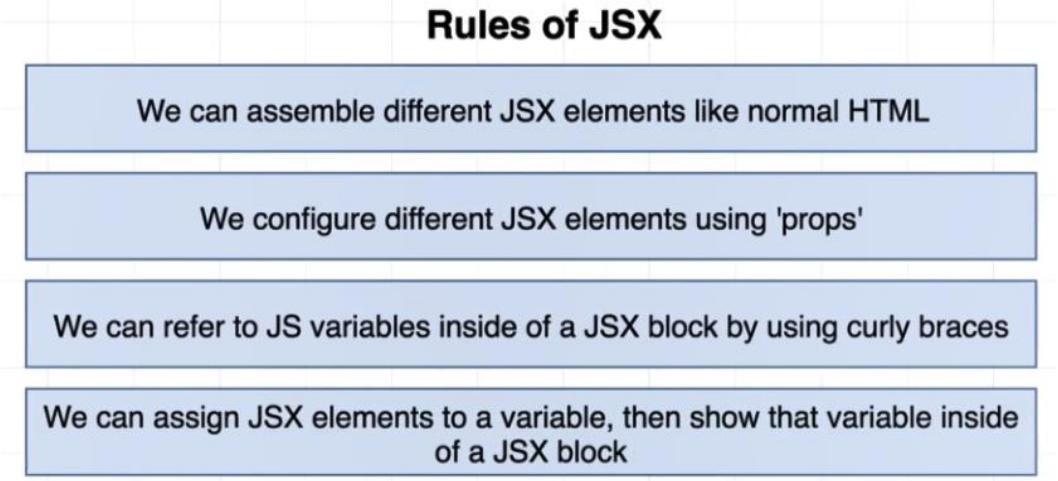

3. Common Questions and Answers

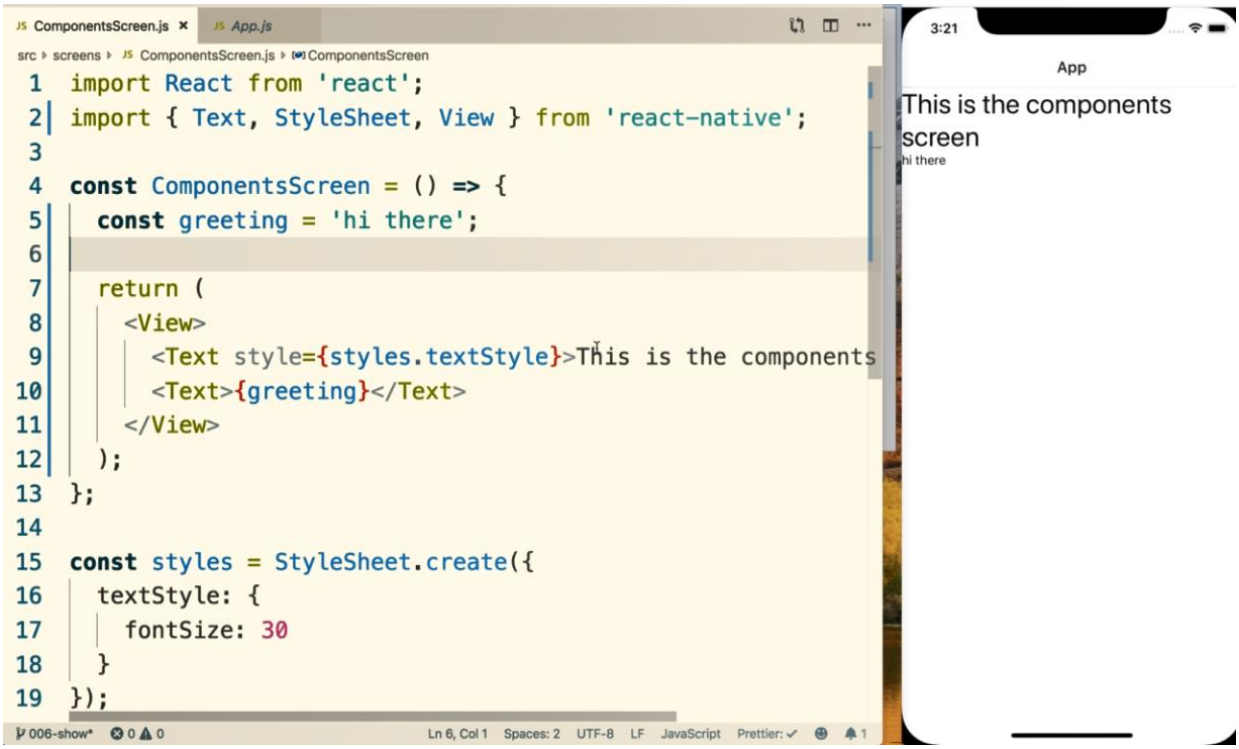
| Questions | Answers |
| --- | --- |
| What's that 'Text' thing? | A 'primitive' React element. Used to show some basic content on the screen. |
| What's that HTML looking stuff? | JSX! It is a 'dialect' of Javascript that tells React what content we want to show |
| What's that 'appNavigator' in the 'App.js' file? | Its a tool from a library called 'React Navigation' that is used to show different screens to the user |
| How's that 'styles' thing work? | The 'StyleSheet.create()' function *validates* a set of styling rules that we pass into it. We can use it or pass styling directly into an element |

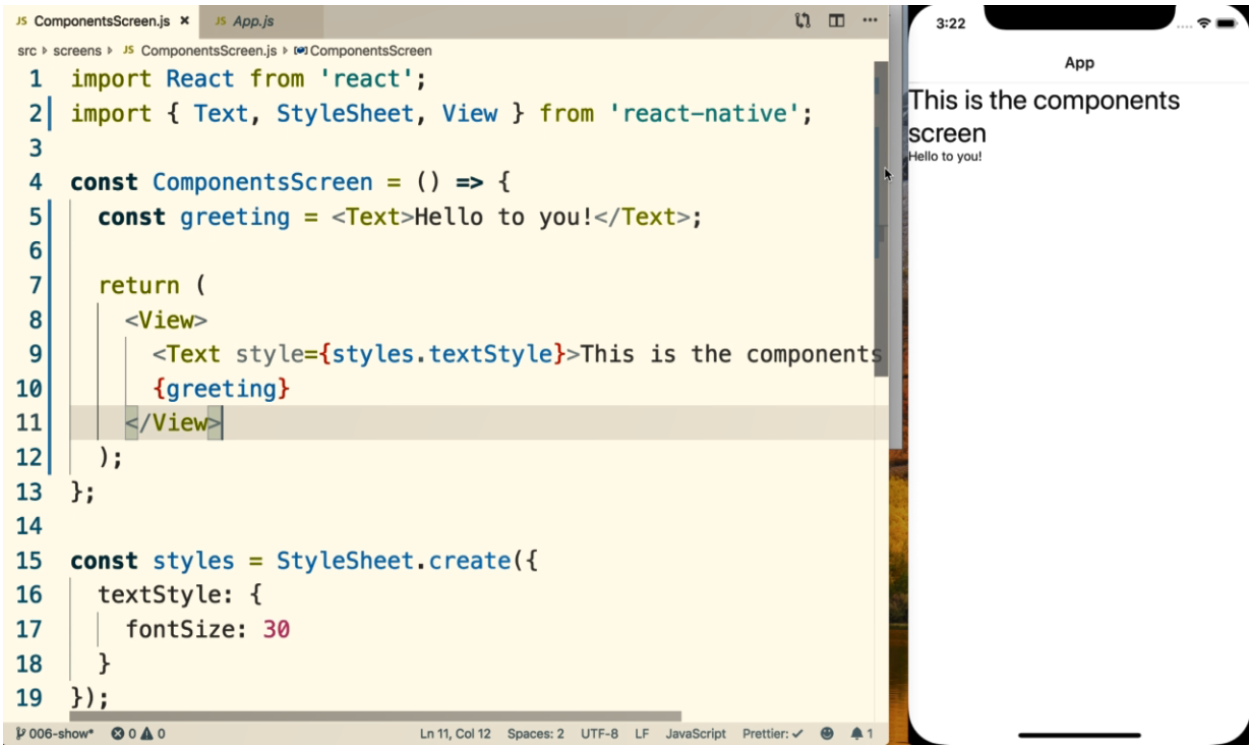
Use this link to show the JSX format: https://babeljs.io/repl

```
const ComponentsScreen = () => {
    return React.createElement(Text, {
        style: styles.textStyle
      }, "This is the Components Screen. Hello World");
};
```

4. Rules of JSX

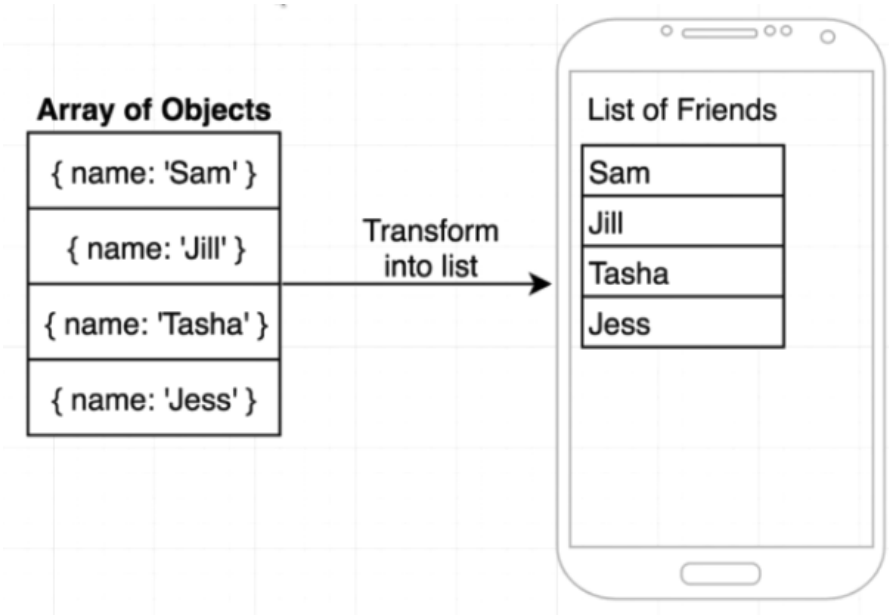





## 3. List Building - With Style

1. The FlatList Element



# FlatList Element

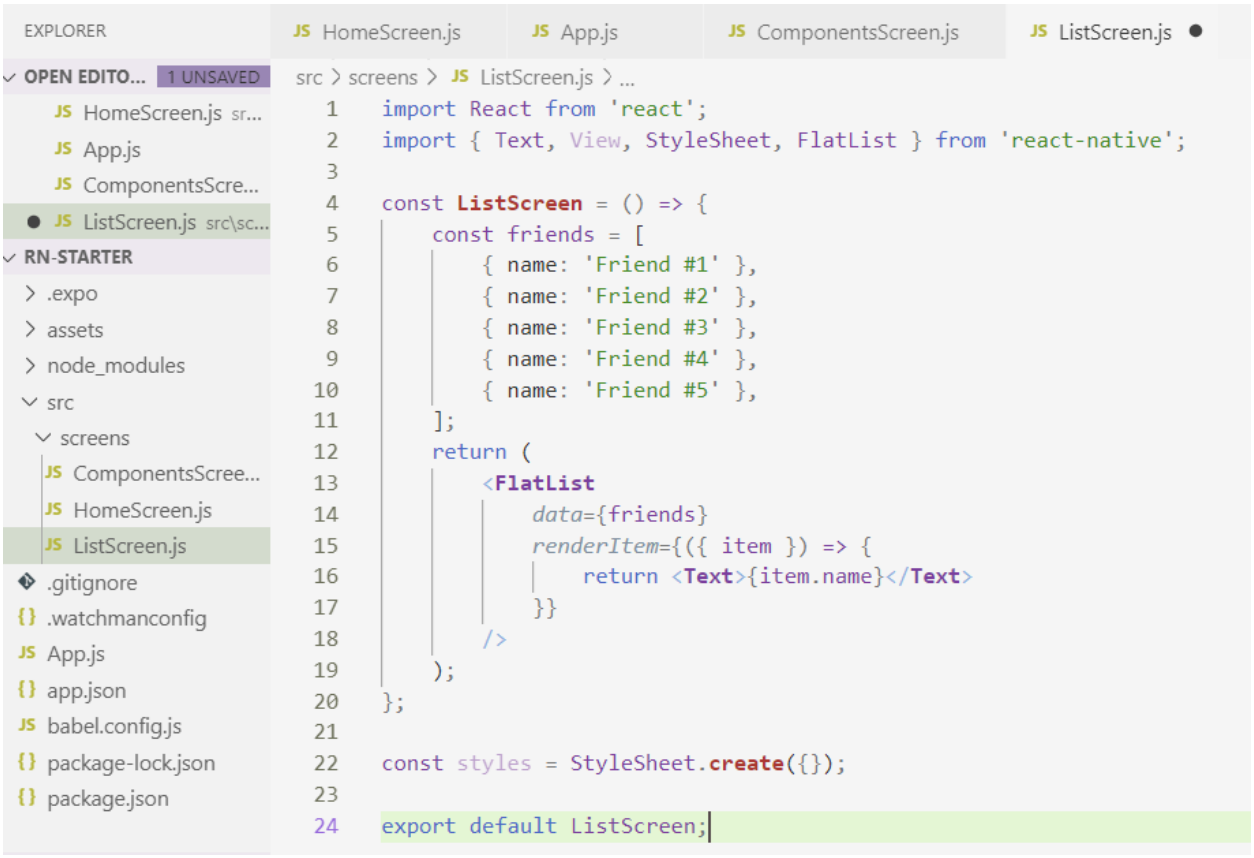
Turns an array into a list of elements

We are required to pass in a 'prop' of 'data' - the array of data that we are going to create a bunch of elements out of

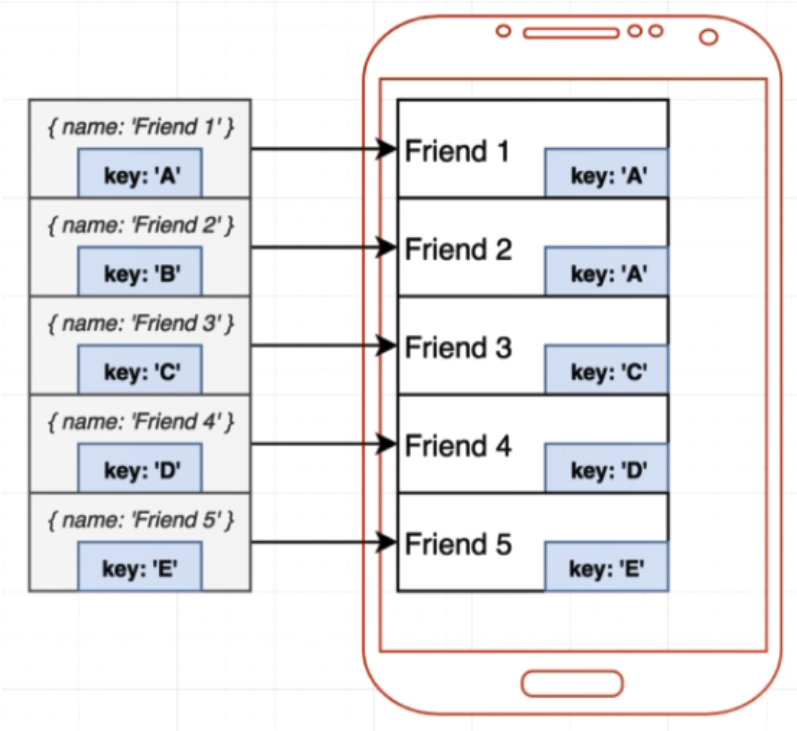
Also required to pass in a 'renderItem' prop - function that will turn each individual item into an element

If you are coming from React on the web, you might be used to 'mapping' an array of data to build a list - FlatList is better with RN

2. Rendering a FlatList



3. Solving the Key Issue




src > screens > JS ListScreen.js > ...

```
1   import React from 'react';
2   import { Text, View, StyleSheet, FlatList } from 'react-native';
3
4   const ListScreen = () => {
5       const friends = [
6           { name: 'Friend #1' },
7           { name: 'Friend #2' },
8           { name: 'Friend #3' },
9           { name: 'Friend #4' },
10          { name: 'Friend #5' },
11      ];
12      return (
13          <FlatList
14              keyExtractor = {friend => friend.name}
15              data={friends}
16              renderItem={({ item }) => {
17                  return <Text>{item.name}</Text>
18              }}
19          />
20      );
21  };
22
23  const styles = StyleSheet.create({});
24
25  export default ListScreen;
```

{ name: 'Friend #1' , key: '1' } we can use unique key for each items.

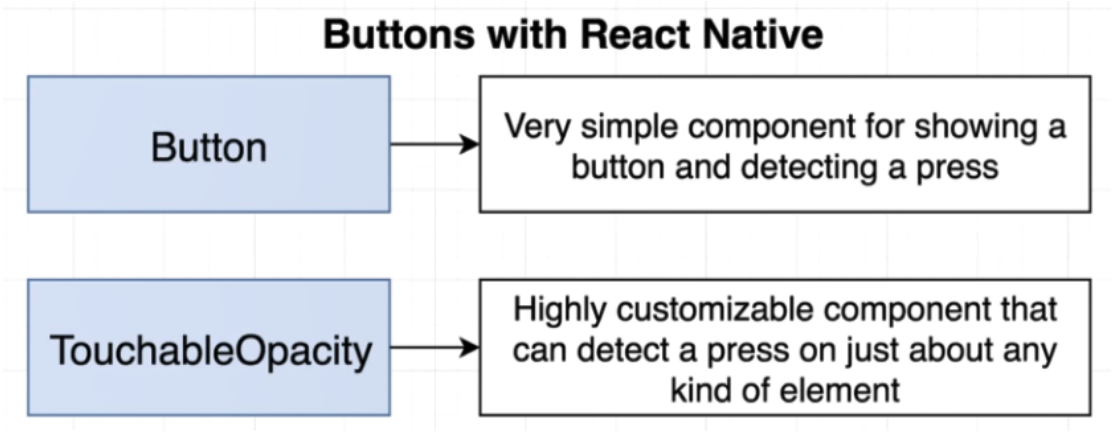
4. A Few Props Around FlatList

src > screens > JS ListScreen.js > ...

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

const ListScreen = () => {
    const friends = [
        { name: 'Friend #1', age: 24 },
        { name: 'Friend #2', age: 21 },
        { name: 'Friend #3', age: 29 },
        { name: 'Friend #4', age: 21 },
        { name: 'Friend #5', age: 27 },
        { name: 'Friend #6', age: 23 },
        { name: 'Friend #7', age: 23 },
        { name: 'Friend #8', age: 28 },
        { name: 'Friend #9', age: 20 },
    ];
    return (
        <FlatList
            //horizontal
            keyExtractor = {friend => friend.name}
            data={friends}
            renderItem={({ item }) => {
                return (
                    <Text style={ styles.textStyle}>
                        Name: {item.name} - Age: {item.age}
                    </Text>
                );
            }}
        />
    );
};

const styles = StyleSheet.create({
    textStyle: {
        marginHorizontal: 20,
        marginVertical: 20
    }
});

export default ListScreen;
```

ShowsHorizontalScrollIndicator?

# 4. Navigating Users Between Screens




1. Button in Action

App.js HomeScreen.js

src > screens > HomeScreen.js > ...

```
1   import React from 'react';
2   import { Text, StyleSheet, Button, View } from 'react-native';
3
4   const HomeScreen = () => {
5     return (
6       <View>
7         <Text style={styles.text}>Hello World!</Text>
8         <Button
9           title="Go to Components Demo"
10          onPress={() => console.log('Button Pressed')}
11        />
12      </View>
13    );;
14  };
15
16  const styles = StyleSheet.create({
17    text: {
18      fontSize: 30,
19      padding: 20
20    }
21  });
22
23  export default HomeScreen;
24
```

2. Touchable Opacity in Action

src > screens > JS HomeScreen.js > styles

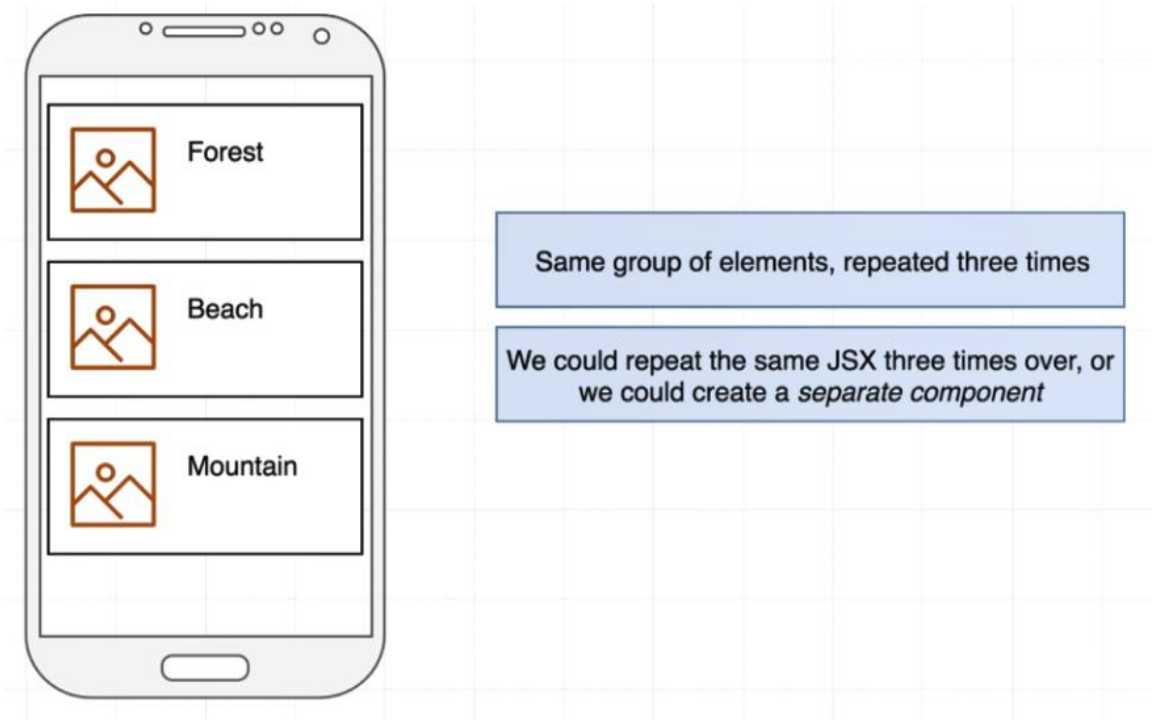
```
1   import React from 'react';
2   import { Text, StyleSheet, Button, View, TouchableOpacity } from 'react-native';
3
4   const HomeScreen = () => {
5     return (
6       <View style={styles.viewStyle}>
7         <Text style={styles.text}>Hello World!</Text>
8         <Button
9           title="Go to Components Demo"
10          onPress={() => console.log('Button Pressed')}
11        />
12        <TouchableOpacity onPress={() => console.log('List Pressed')}>
13          <Text>Go to List Demo</Text>
14        </TouchableOpacity>
15      </View>
16    );;
17  };
18
19  const styles = StyleSheet.create({
20    text: {
21      fontSize: 30,
22      padding: 20
23    },
24    viewStyle:{
25      marginHorizontal: 15,
26      marginVertical: 15
27    }
28  });
29
30  export default HomeScreen;
31
```

3. Navigating with React Navigation

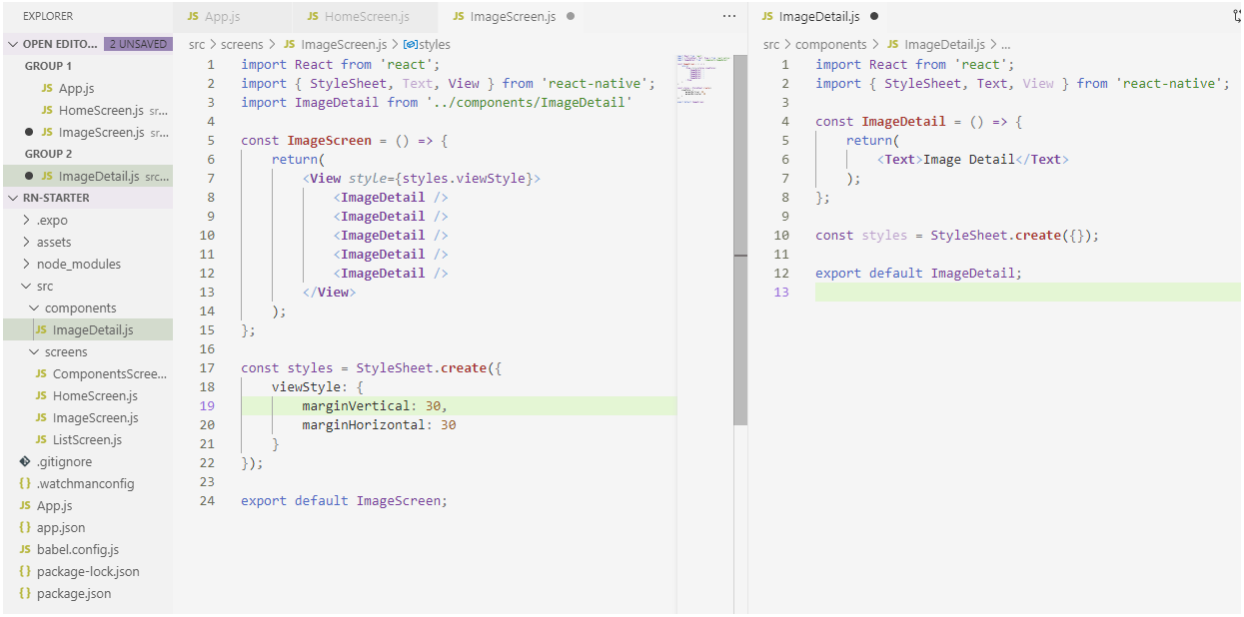
```
const HomeScreen = (props) => {
  //console.log(props.navigation.navigate);
  return (
    <View style={styles.viewStyle}>
      <Text style={styles.text}>Hello World!</Text>
      <Button
        title="Go to Components Demo"
        onPress={() => props.navigation.navigate('Components')}
      />
      <TouchableOpacity onPress={() => props.navigation.navigate('List')}>
        <Text>Go to List Demo</Text>
      </TouchableOpacity>
    </View>
  );
};
```

```jsx
const HomeScreen = ({ navigation }) => {
  return (
    <View style={styles.viewStyle}>
      <Text style={styles.text}>Hello World!</Text>
      <Button
        title="Go to Components Demo"
        onPress={() => navigation.navigate('Components')}
      />
      <Button
        title="Go to List Demo"
        onPress={() => navigation.navigate('List')}
      />

    </View>
  );
};
```

## 5. Building Reusable Components




Create ImageScreen.js and add button for HomeScreen to Navigate to the Image Screen.




1. Communicating from Parent to Child

2. Passing Images as Props

ImageScreen.js

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
import ImageDetail from '../components/ImageDetail'

const ImageScreen = () => {
    return(
        <View style={styles.viewStyle}>
            <ImageDetail
                title="Forest"
                imageSource={require('../../assets/forest.jpg')}
            />
            <ImageDetail
                title="Beach"
                imageSource={require('../../assets/beach.jpg')}
            />
            <ImageDetail
                title="Mountain"
                imageSource={require('../../assets/mountain.jpg')}
            />
        </View>
    );
};

const styles = StyleSheet.create({
    viewStyle: {
        marginVertical: 30,
        marginHorizontal: 30
    }
});

export default ImageScreen;
```

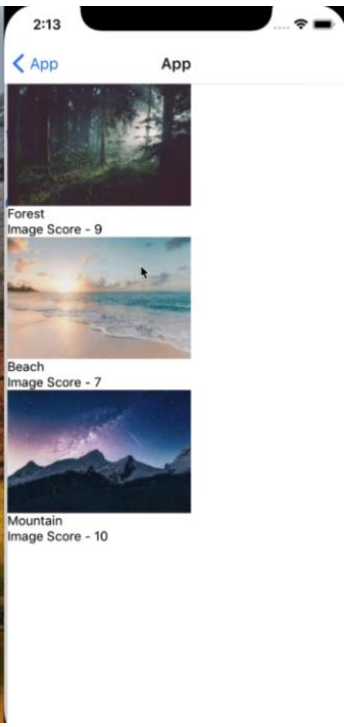
ImageDetail.js

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

const ImageDetail = (props) => {
    console.log(props);
    return(
        <View>
            <Image source={props.imageSource} />
            <Text>{props.title}</Text>
        </View>
    );
};

const styles = StyleSheet.create({});

export default ImageDetail;
```

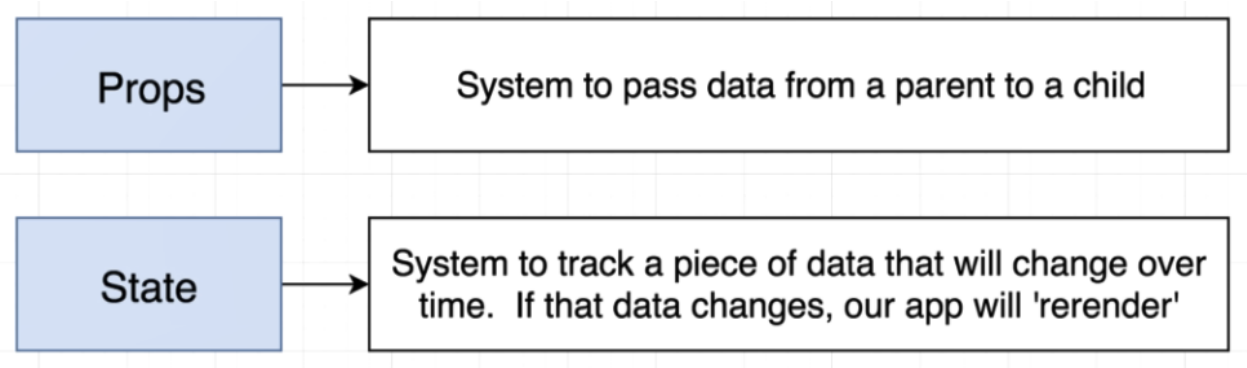
ListScreen.js | HomeScreen.js | ImageDetail.js | ImageScreen.js

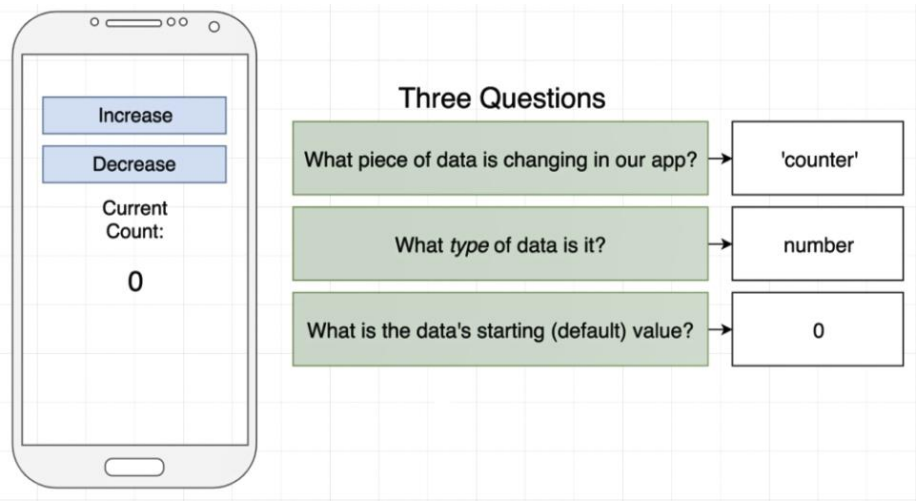
src > components > ImageDetail.js > ImageDetail

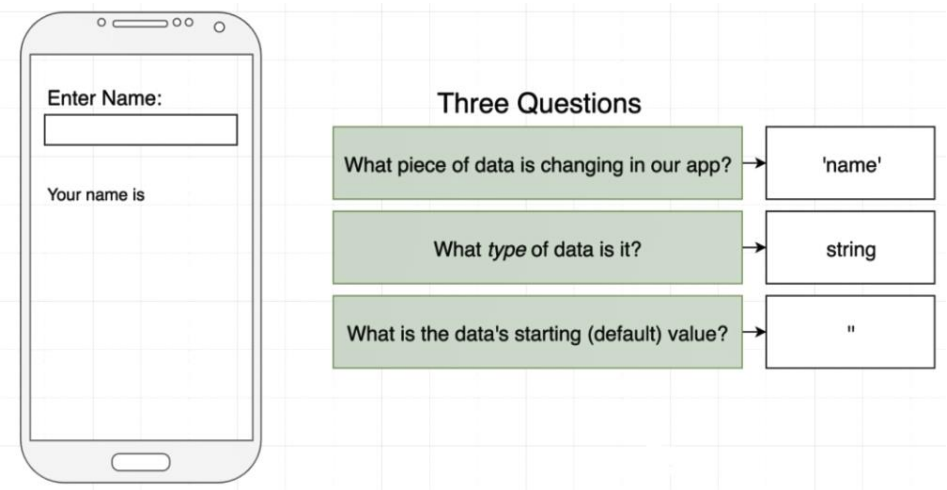
```
import React from 'react';
import { View, Text, StyleSheet, Image } from 'react-nati

const ImageDetail = ({ imageSource, title, score }) => {
  return (
    <View>
      <Image source={imageSource} />
      <Text>{title}</Text>
      <Text>Image Score - {score}</Text>
    </View>
  );
};

const styles = StyleSheet.create({});

export default ImageDetail;
```

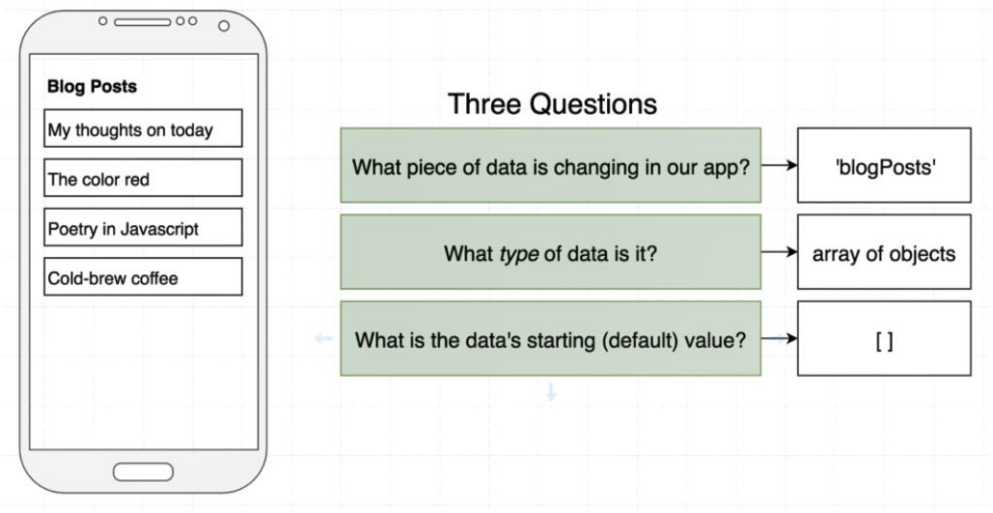


# 6. State Management in React Components






Create a CounterScreen.js File in screens folder.

JS ImageScreen.js | JS CounterScreen.js × | JS ListScreen.js | JS HomeScreen.js

src > screens > JS CounterScreen.js > styles

```
1   import React from 'react';
2   import { Text, View, StyleSheet, Button } from 'react-native';
3
4   const CounterScreen = () => {
5
6       let counter = 0;
7
8       return (
9           <View style={styles.viewStyle}>
10              <Button
11                  title="Increase"
12                  onPress={() => {
13                      counter++;
14                      console.log(counter);
15                  }} />
16              <Button
17                  title="dicrease"
18                  onPress={() => {
19                      counter--;
20                      console.log(counter);
21                  }} />
22              <Text>Current Count: {counter}</Text>
23          </View>
24      );
25  };
26
27  const styles = StyleSheet.create({
28      viewStyle: {
29          padding: 15
30      }
31  });
32
33  export default CounterScreen;
```

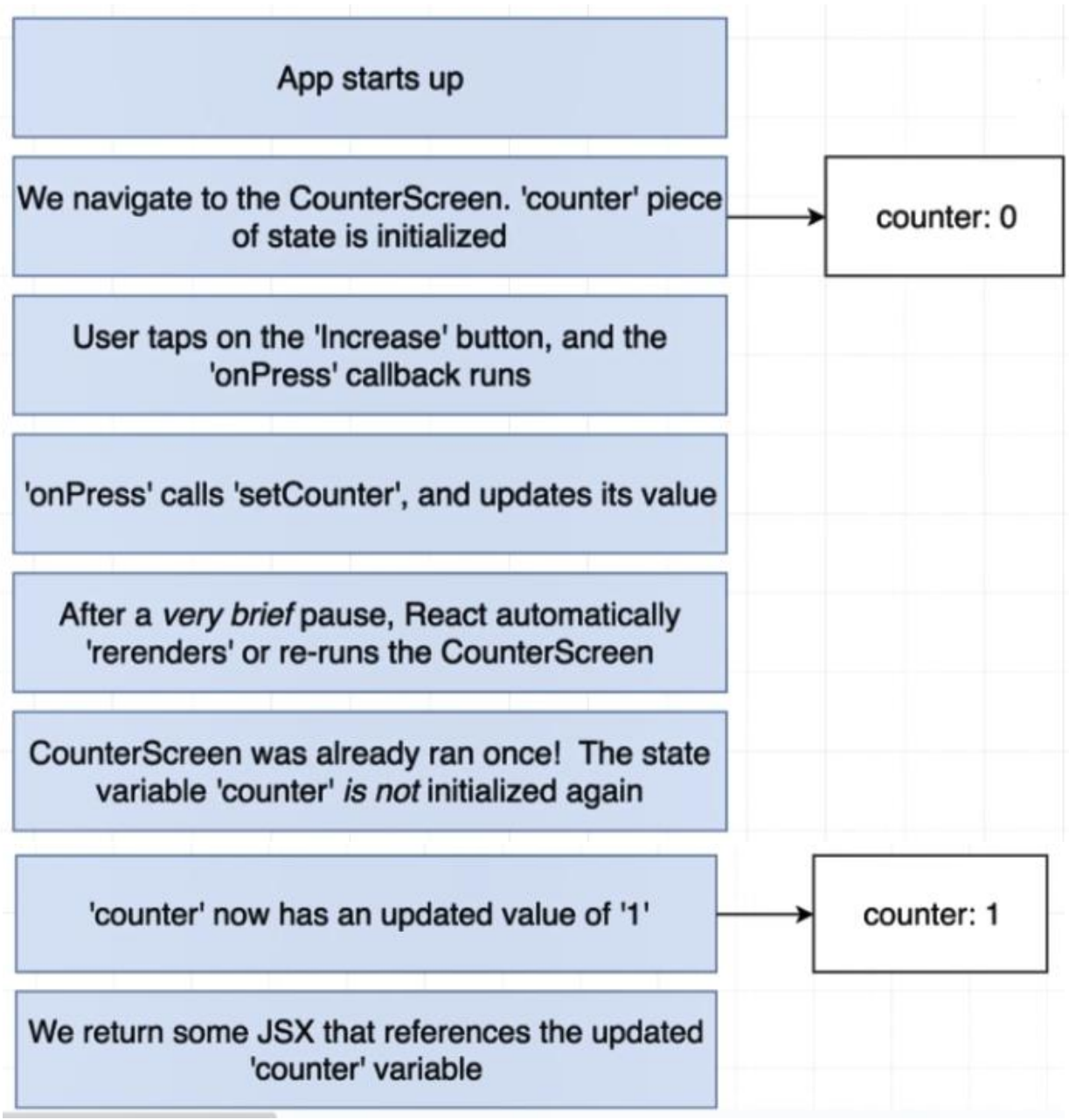
When press button, counter is up or down in console log. But not showing in App.

1. State in Action

We have to import useState from react Library. After it's worked well.

src > screens > JS CounterScreen.js > ...

```
1   import React, { useState } from 'react';
2   import { Text, View, StyleSheet, Button } from 'react-native';
3
4   const CounterScreen = () => {
5       const [counter, setCounter] = useState(0);
6       return (
7           <View style={styles.viewStyle}>
8               <Button title="Increase" onPress={() => {
9                   setCounter(counter + 1);
10              }} />
11              <Button title="dicrease" onPress={() => {
12                  setCounter(counter - 1);
13              }} />
14              <Text>Current Count: {counter}</Text>
15          </View>
16      );
17  };
18
19  const styles = StyleSheet.create({
20      viewStyle: {
21          padding: 15
22      }
23  });
24
25  export default CounterScreen;
```


App starts up
We navigate to the CounterScreen. 'counter' piece of state is initialized
counter: 0
User taps on the 'Increase' button, and the 'onPress' callback runs
'onPress' calls 'setCounter', and updates its value
After a *very brief* pause, React automatically 'rerenders' or re-runs the CounterScreen
CounterScreen was already ran once! The state variable 'counter' *is not* initialized again
'counter' now has an updated value of '1'
counter: 1
We return some JSX that references the updated 'counter' variable

## Few Notes on State

We are using function-based state in a functional component. React also has class-based components that have access to state

We **never** directly modify a state variable. React doesn't detect this change! Only use the 'setter' function

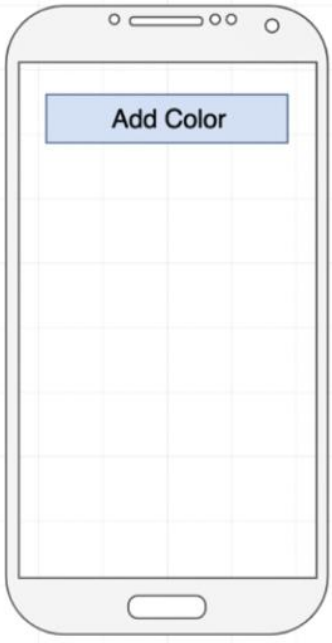
We can name the state variable anything we wish

We can track any kind of data that changes over time - a number, string, array of objects, etc

When a component is rerendered, *all of its children get rerendered too*

A state variable can be passed to a child component! At that point, the state variable is now being used as props.

2. Generating Random Colors




### Three Questions

| | |
|---|---|
| What piece of data is changing in our app? | colors |
| What *type* of data is it? | array of strings |
| What is the data's starting (default) value? | [] |

ColorScreen.js App.js

src > screens > ColorScreen.js > ...

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

const ColorScreen = () => {
    return (
        <View style={ styles.viewStyle }>
            <Text>ColorScreen</Text>
            <Button title="Add a Color"/>
            {/* <View style={{ height: 100, width: 100, backgroundColor: 'rgb(0, 255, 0)' }}/> */}
            <View style={{ height: 100, width: 100, backgroundColor: randomRgb() }}/>
        </View>
    );
};

const randomRgb = () => {
    const red = Math.floor(Math.random() * 256);
    const green = Math.floor(Math.random() * 256);
    const blue = Math.floor(Math.random() * 256);

    return `rgb(${red}, ${green}, ${blue})`;
    {/* The grave accent ( ` ) is a diacritical mark used in many written languages. */}
};


const styles = StyleSheet.create({
    viewStyle: {
        padding: 15
    }
});

export default ColorScreen;
```

src > screens > ColorScreen.js > styles

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

const ColorScreen = () => {
    const [colors, setColors] = useState([]);
    console.log(colors);
    return (
        <View style={ styles.viewStyle }>
            <Text>ColorScreen</Text>
            <Button title="Add a Color"
                onPress={() => {
                    setColors([...colors, randomRgb()]);
                }}
            />
            {/* <View style={{ height: 100, width: 100, backgroundColor: 'rgb(0, 255, 0)' }}/> */}
            <View style={{ height: 100, width: 100, backgroundColor: randomRgb() }}/>
        </View>
    );
};
```

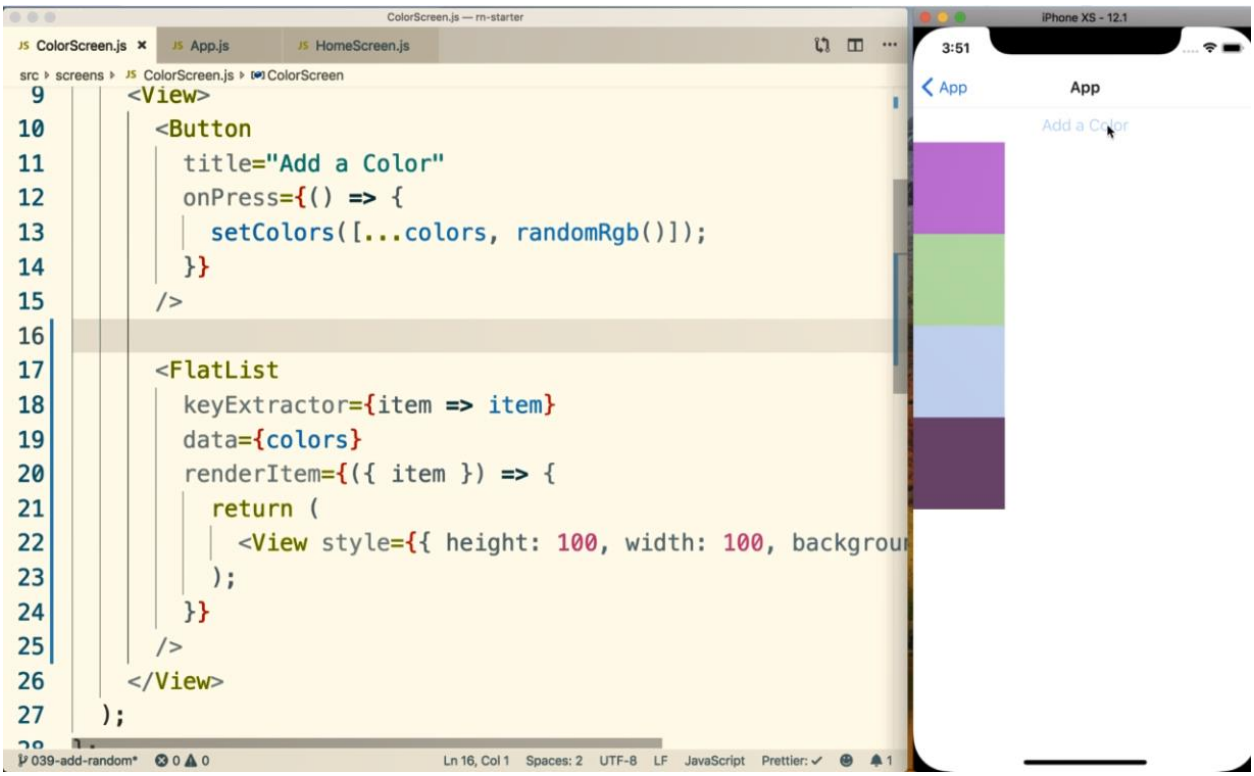
Press “Add a Color” button. Then change the color on view area.

3. Showing Colors with a FlatList

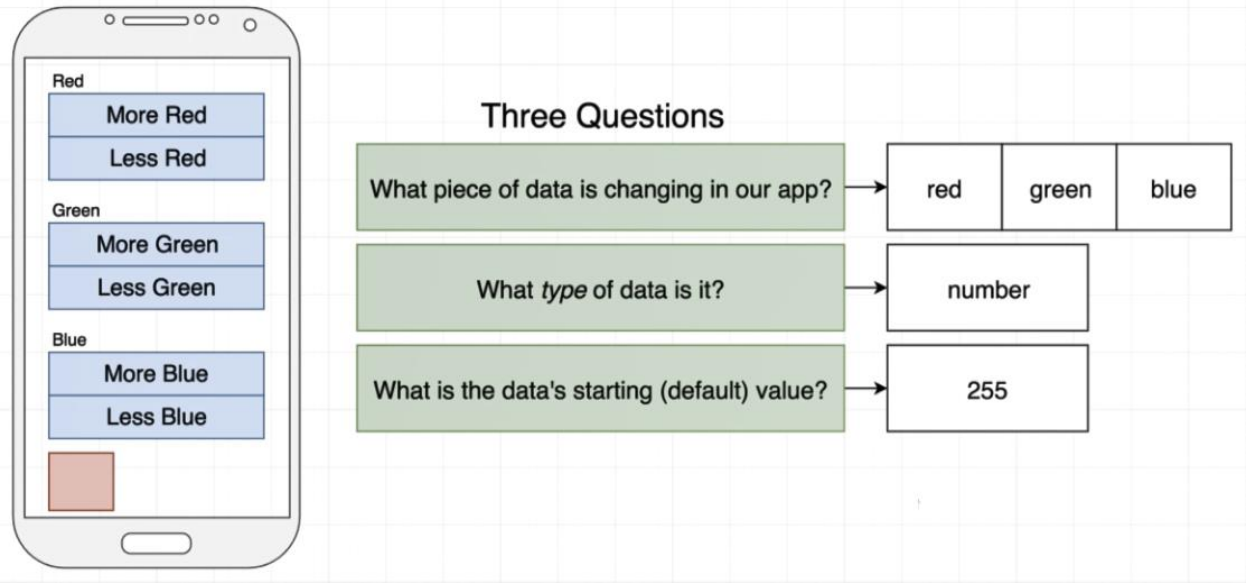
src > screens > JS ColorScreen.js > randomRgb

```
1   import React, { useState } from 'react';
2   import { View, Text, StyleSheet,  Button, FlatList } from 'react-native';
3
4   const ColorScreen = () => {
5       const [colors, setColors] = useState([]);
6       return (
7           <View style={ styles.viewStyle }>
8               <Text>ColorScreen</Text>
9               <Button title="Add a Color"
10                  onPress={() => {
11                      setColors([...colors, randomRgb()]);
12                  }}
13              />
14              {/* <View style={{ height: 100, width: 100, backgroundColor: 'rgb(0, 255, 0)' }}/> */}
15              <FlatList
16                  keyExtractor={item => item}
17                  data={colors}
18                  renderItem={({ item }) => {
19                      return <View style={{ height: 100, width: 100, backgroundColor: item }}/>;
20                  }}
21              />
22          </View>
23      );
24  };
```

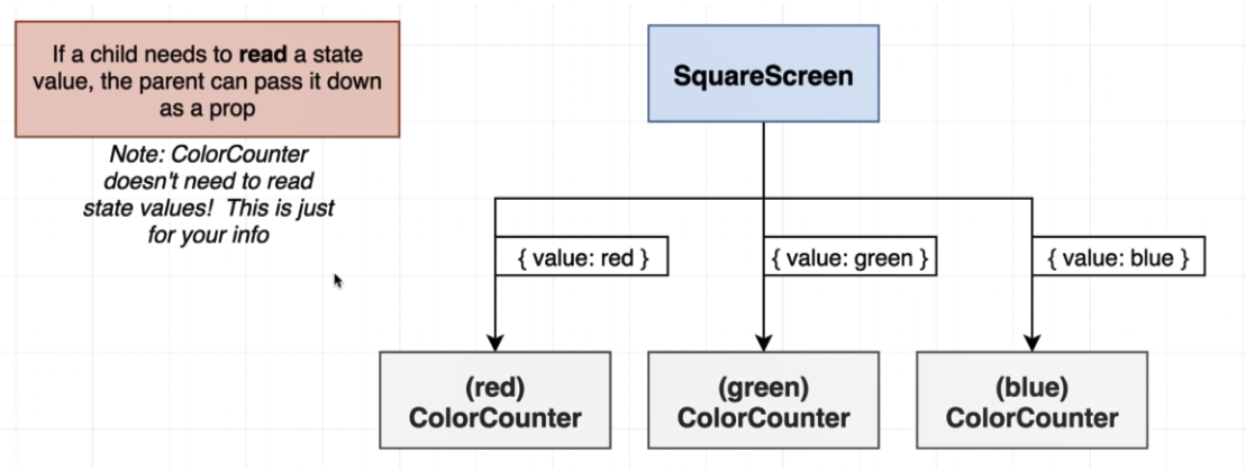
We can see the colors in FlatList.

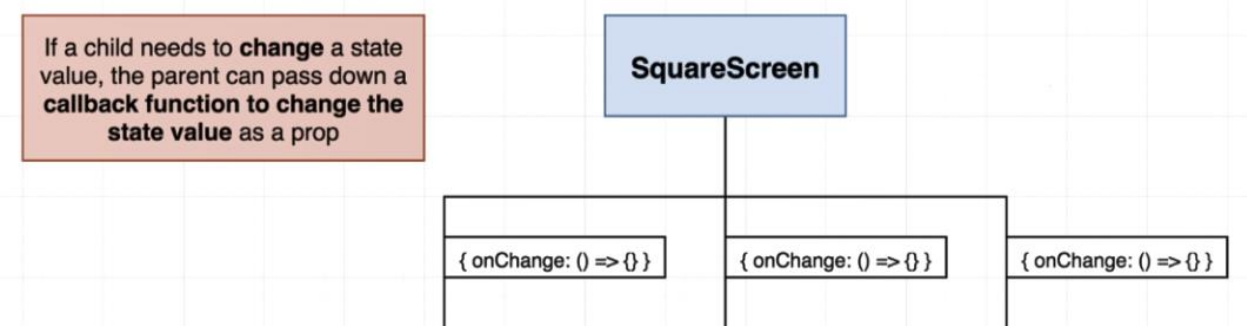


4. Reusable Color Adjusters




Create SquareScreen.js in screens directory and ColorCounter.js in components directory.



ColorCounter.js

SquareScreen.js ● | ColorCounter.js × | HomeScreen.js

src > components > ColorCounter.js > ...

```
1   import React, { useState } from 'react';
2   import { View, Button, Text, StyleSheet } from 'react-native';
3
4   const ColorCounter = ({ color, onIncrease, onDecrease }) => {
5       return (
6           <View>
7               <Text>{color}</Text>
8               <Button onPress={() => onIncrease()} title={`Increase ${color}`}   />
9               <Button onPress={() => onDecrease()} title={`Decrease ${color}`} />
10          </View>
11      );
12  };
13
14  const styles = StyleSheet.create({});
15
16  export default ColorCounter;
```

SquareScreen.js

SquareScreen.js ● | ColorCounter.js | HomeScreen.js

src > screens > SquareScreen.js > ...

```
1   import React, { useState } from 'react';
2   import { View, StyleSheet } from 'react-native';
3   import ColorCounter from '../components/ColorCounter';
4
5   const COLOR_INCREMENT = 15;
6
7   const SquareScreen = () => {
8       const [red, setRed] = useState(0);
9       const [green, setGreen] = useState(0);
10      const [blue, setBlue] = useState(0);
11      console.log(red);
12      return (
13          <View style={ styles.viewStyle }>
14              <ColorCounter
15                  color="Red"
16                  onIncrease={() => setRed(red + COLOR_INCREMENT)}
17                  onDecrease={() => setRed(red - COLOR_INCREMENT)}
18              />
19              <ColorCounter
20                  color="Green"
21                  onIncrease={() => setGreen(green + COLOR_INCREMENT)}
22                  onDecrease={() => setGreen(green - COLOR_INCREMENT)}
23              />
24              <ColorCounter
25                  color="Blue"
26                  onIncrease={() => setBlue(blue + COLOR_INCREMENT)}
27                  onDecrease={() => setBlue(blue - COLOR_INCREMENT)}
28              />
29              <View style={{ height: 100, width: 100, backgroundColor: `rgb(${red}, ${green}, ${blue})` }}/>
30          </View>
31      );
32  };
```

How to use if function in render part

```
return (
    <View style={ styles.viewStyle }>
        <ColorCounter
            color="Red"
            onIncrease={() => {
                if(red + COLOR_INCREMENT > 255){
                    return;
                }
                setRed(red + COLOR_INCREMENT)
            }}
            onDecrease={() => setRed(red - COLOR_INCREMENT)}
        />
```

How to use switch case in React Native

```
const setColor = (color, change) => {
    switch (color) {
        case 'red':
            if (red + change > 255 || red + change < 0) {
                return;
            } else {
                setRed(red + change);
            }
            return;
        case 'green':
            if (green + change > 255 || green + change < 0) {
                return;
            } else {
                setRed(green + change);
            }
            return;
        case 'blue':
            if (blue + change > 255 || blue + change < 0) {
                return;
            } else {
                setRed(blue + change);
            }
            return;
    }
};
```

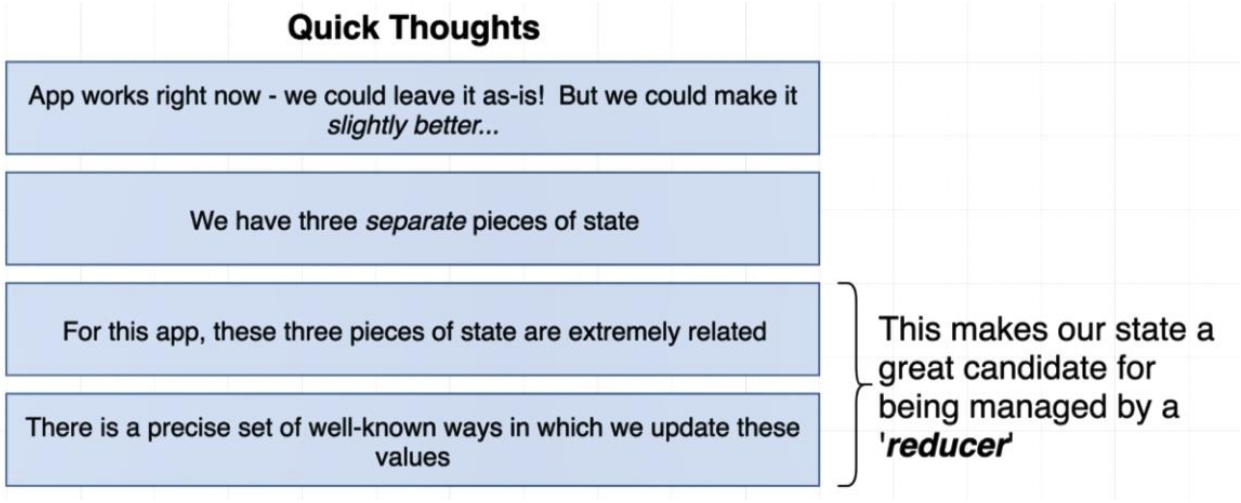
5. Validating State Changes

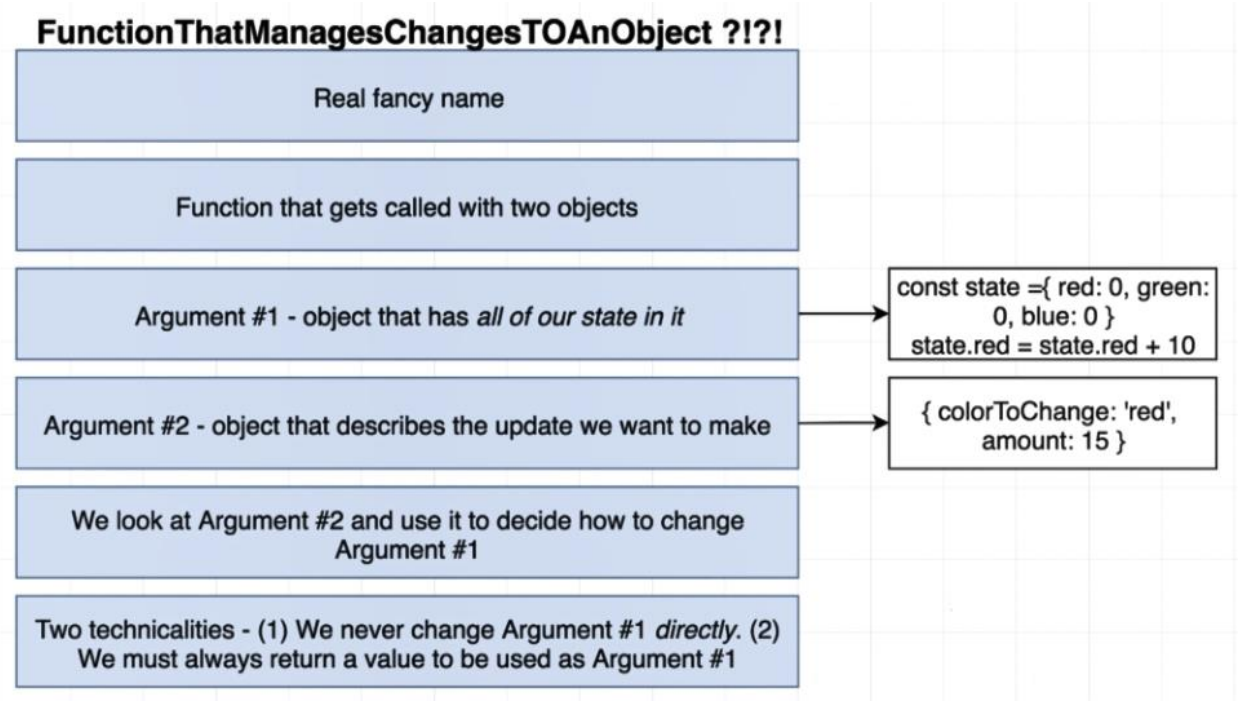
```
const COLOR_INCREMENT = 15;
const SquareScreen = () => {
    const [red, setRed] = useState(0);
    const [green, setGreen] = useState(0);
    const [blue, setBlue] = useState(0);
    //Validator
    const setColor = (color, change) => {
        switch (color) {
            case 'red':
                red + change > 255 || red + change < 0 ? null : setRed(red + change);
                return;
            case 'green':
                green + change > 255 || green + change < 0 ? null : setGreen(green + change);
                return;
            case 'blue':
                blue + change > 255 || blue + change < 0 ? null : setBlue(blue + change);
                return;
            default:
                return;
        }
    };
```

```
    return (
        <View style={styles.viewStyle}>
            <ColorCounter
                color="Red"
                onIncrease={() => setColor('red', COLOR_INCREMENT)}
                onDecrease={() => setColor('red', -1 * COLOR_INCREMENT)}
            />
            <ColorCounter
                color="Green"
                onIncrease={() => setColor('green', COLOR_INCREMENT)}
                onDecrease={() => setColor('green', -1 * COLOR_INCREMENT)}
            />
            <ColorCounter
                color="Blue"
                onIncrease={() => setColor('blue', COLOR_INCREMENT)}
                onDecrease={() => setColor('blue', -1 * COLOR_INCREMENT)}
            />
            <View style={{ height: 100, width: 100, backgroundColor: `rgb(${red}, ${green}, ${blue})` }} />
        </View>
    );
};
```

One line if statement :

```
red + change > 255 || red + change < 0 ? null : setRed(red + change);
```


Quick Thoughts
App works right now - we could leave it as-is! But we could make it slightly better...
We have three separate pieces of state
For this app, these three pieces of state are extremely related
There is a precise set of well-known ways in which we update these values
This makes our state a great candidate for being managed by a 'reducer'


FunctionThatManagesChangesTOAnObject ?!?!
Real fancy name
Function that gets called with two objects
Argument #1 - object that has all of our state in it
const state ={ red: 0, green: 0, blue: 0 }
state.red = state.red + 10
Argument #2 - object that describes the update we want to make
{ colorToChange: 'red', amount: 15 }
We look at Argument #2 and use it to decide how to change Argument #1
Two technicalities - (1) We never change Argument #1 directly. (2) We must always return a value to be used as Argument #1

Note :

## 6. Creating a Reducer

JS SquareScreen.js × JS ColorCounter.js JS HomeScreen.js

src > screens > JS SquareScreen.js > SquareScreen

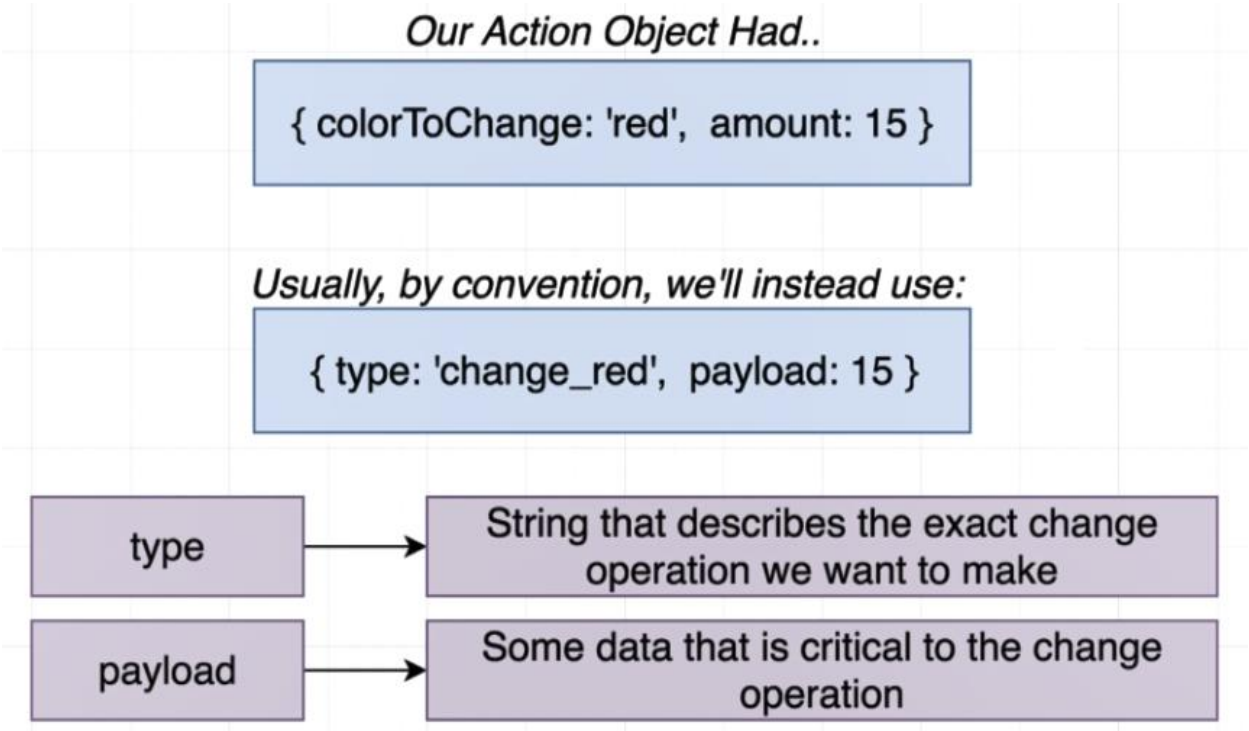
```
1  import React, { useReducer } from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import ColorCounter from '../components/ColorCounter';
4
5  const COLOR_INCREMENT = 15;
6
7  const reducer = (state, action) => {
8      //state === { red: number, green: number, blue: number }
9      //action === { colorToChange: 'red' || 'green' || 'blue', amount: 15 || -15 }
10
11     switch (action.colorToChange) {
12         case 'red':
13             return { ...state, red: state.red + action.amount };
14         case 'green':
15             return { ...state, green: state.green + action.amount };
16         case 'blue':
17             return { ...state, blue: state.blue + action.amount };
18         default:
19             return state;
20     }
21
22 };
```

```
24 const SquareScreen = () => {
25     const [state, dispatch] = useReducer(reducer, { red: 0, green: 0, blue: 0 });
26     const { red, green, blue } = state;
27     return (
28         <View style={styles.viewStyle}>
29             <ColorCounter
30                 color="Red"
31                 onIncrease={() => dispatch({ colorToChange: 'red', amount: COLOR_INCREMENT })}
32                 onDecrease={() => dispatch({ colorToChange: 'red', amount: -1 * COLOR_INCREMENT })}
33             />
34             <ColorCounter
35                 color="Green"
36                 onIncrease={() => dispatch({ colorToChange: 'green', amount: COLOR_INCREMENT })}
37                 onDecrease={() => dispatch({ colorToChange: 'green', amount: -1 * COLOR_INCREMENT })}
38             />
39             <ColorCounter
40                 color="Blue"
41                 onIncrease={() => dispatch({ colorToChange: 'blue', amount: COLOR_INCREMENT })}
42                 onDecrease={() => dispatch({ colorToChange: 'blue', amount: -1 * COLOR_INCREMENT })}
43             />
44             <View style={{ height: 100, width: 100, backgroundColor: `rgb(${state.red}, ${state.green}, ${state.blue})` }}
45         </View>
46     );
47 };
```

7. Restoring Validation

```
7   const reducer = (state, action) => {
8       //state === { red: number, green: number, blue: number }
9       //action === { colorToChange: 'red' || 'green' || 'blue', amount: 15 || -15 }
10
11      switch (action.colorToChange) {
12          case 'red':
13              return state.red + action.amount > 255 || state.red + action.amount < 0
14                  ? state
15                  : { ...state, red: state.red + action.amount };
16          case 'green':
17              return state.green + action.amount > 255 || state.green + action.amount < 0
18                  ? state
19                  : { ...state, green: state.green + action.amount };
20          case 'blue':
21              return state.blue + action.amount > 255 || state.blue + action.amount < 0
22                  ? state
23                  : { ...state, blue: state.blue + action.amount };
24          default:
25              return state;
26      }
27
28  };
29
```

8. Community Convention in Reducers



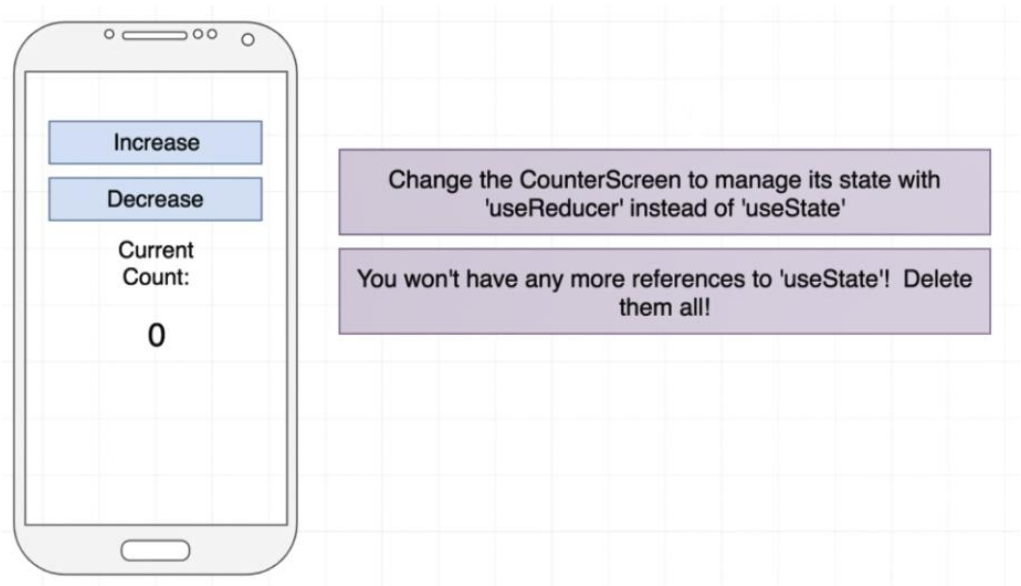
After change Community Convention type.

src > screens > JS SquareScreen.js > SquareScreen

```
import React, { useReducer } from 'react';
import { View, StyleSheet } from 'react-native';
import ColorCounter from '../components/ColorCounter';

const COLOR_INCREMENT = 15;

const reducer = (state, action) => {
    //state === { red: number, green: number, blue: number }
    //action === { type: 'change_red' || 'change_green' || 'change_blue', payload: 15 || -15 }

    switch (action.type) {
        case 'change_red':
            return state.red + action.payload > 255 || state.red + action.payload < 0
                ? state
                : { ...state, red: state.red + action.payload };
        case 'change_green':
            return state.green + action.payload > 255 || state.green + action.payload < 0
                ? state
                : { ...state, green: state.green + action.payload };
        case 'change_blue':
            return state.blue + action.payload > 255 || state.blue + action.payload < 0
                ? state
                : { ...state, blue: state.blue + action.payload };
        default:
            return state;
    }

};
```

```
const SquareScreen = () => {
    const [state, dispatch] = useReducer(reducer, { red: 0, green: 0, blue: 0 });
    const { red, green, blue } = state;
    return (
        <View style={styles.viewStyle}>
            <ColorCounter
                color="Red"
                onIncrease={() => dispatch({ type: 'change_red', payload: COLOR_INCREMENT })}
                onDecrease={() => dispatch({ type: 'change_red', payload: -1 * COLOR_INCREMENT })}
            />
            <ColorCounter
                color="Green"
                onIncrease={() => dispatch({ type: 'change_green', payload: COLOR_INCREMENT })}
                onDecrease={() => dispatch({ type: 'change_green', payload: -1 * COLOR_INCREMENT })}
            />
            <ColorCounter
                color="Blue"
                onIncrease={() => dispatch({ type: 'change_blue', payload: COLOR_INCREMENT })}
                onDecrease={() => dispatch({ type: 'change_blue', payload: -1 * COLOR_INCREMENT })}
            />
            <View style={{ height: 100, width: 100, backgroundColor: `rgb(${state.red}, ${state.green}, ${state.blue})` }}
        </View>
    );
};
```

9. Exercise Solution




src > screens > JS CounterScreen.js > ...

```
1   import React, { useReducer } from 'react';
2   import { Text, View, StyleSheet, Button } from 'react-native';
3
4   const reducer = (state, action) => {
5       // state === { count: number }
6       // action === { type: 'increment' || 'decrement', payload: 1 }
7       switch (action.type) {
8           case 'increment':
9               return { ...state, count: state.count + action.payload };
10          case 'decrement':
11              return { ...state, count: state.count - action.payload };
12          default:
13              return state;
14      }
15  };
16
17  const CounterScreen = () => {
18      const [state, dispatch] = useReducer(reducer, { count: 0 });
19      //dispatch = callMyReducer
20
21      return (
22          <View style={styles.viewStyle}>
23              <Button title="Increase" onPress={() => {
24                  dispatch({ type: 'increment', payload: 1 })
25              }} />
26              <Button title="dicrease" onPress={() => {
27                  dispatch({ type: 'decrement', payload: 1 })
28              }} />
29              <Text>Current Count: {state.count}</Text>
30          </View>
31      );
32  };
```

10. Handling Text Input


Show a text input

When a user types in the input, show the same text immediately underneath the input


Validate the user's input, show a message if it doesn't meet some criteria

When a user types in the input, show the same text immediately underneath the input

JS TextScreen.js ×

src > screens > JS TextScreen.js > ...

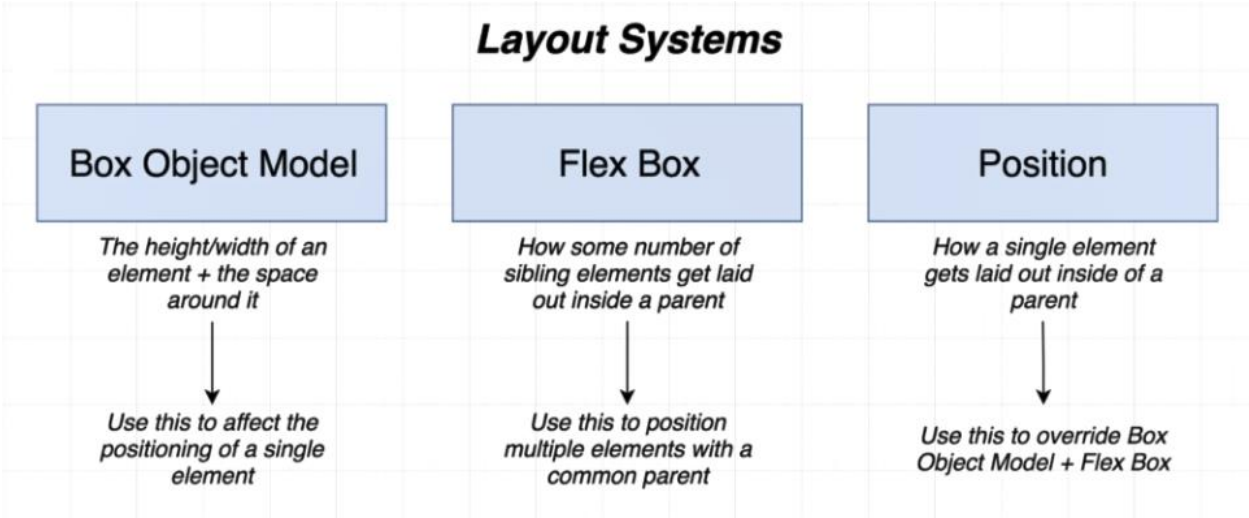
```
1   import React,{ useState } from 'react';
2   import { View, Text, StyleSheet, TextInput } from 'react-native';
3
4   const TextScreen = () => {
5       const [name, setName] = useState('');
6       return (
7           <View style={styles.viewStyle}>
8               <Text>Enter Name: </Text>
9               <TextInput
10                  style={styles.input}
11                  autoCapitalize="none"
12                  autoCorrect={false}
13                  value={name}
14                  onChangeText={(newValue) => setName(newValue)}
15              />
16              <Text>My name is {name}</Text>
17          </View>
18      );
19  }
20
21  const styles = StyleSheet.create({
22      input: {
23          marginVertical: 15,
24          borderColor: 'black',
25          borderWidth: 1
26      },
27      viewStyle: {
28          margin: 15
29      }
30  });
31
32  export default TextScreen;
```

JS TextScreen.js ●

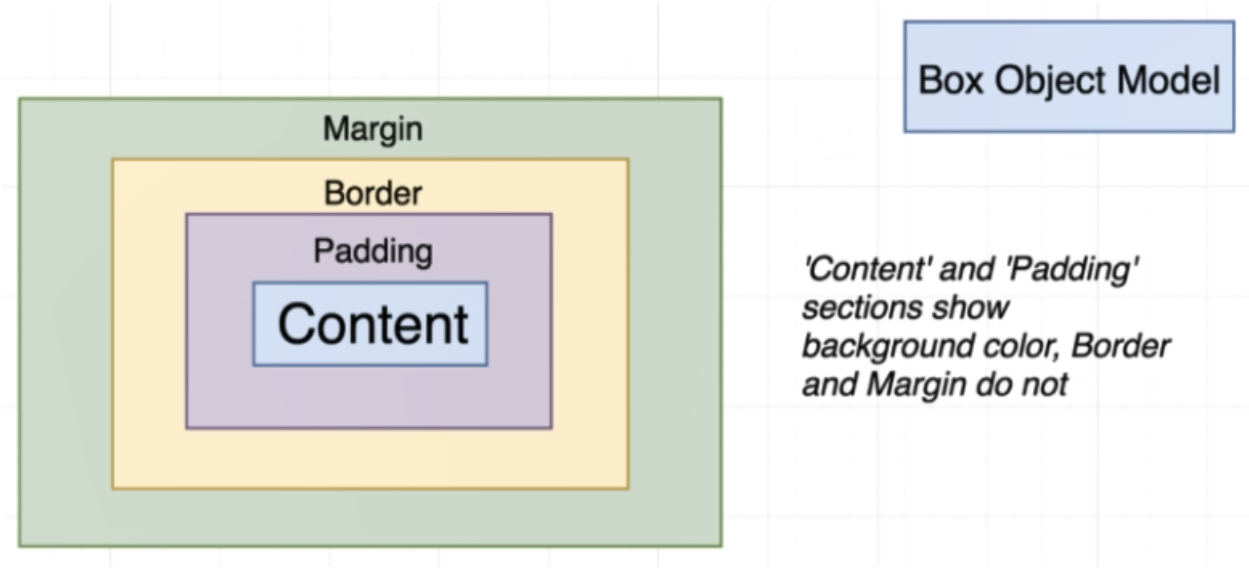
src > screens > JS TextScreen.js > ...

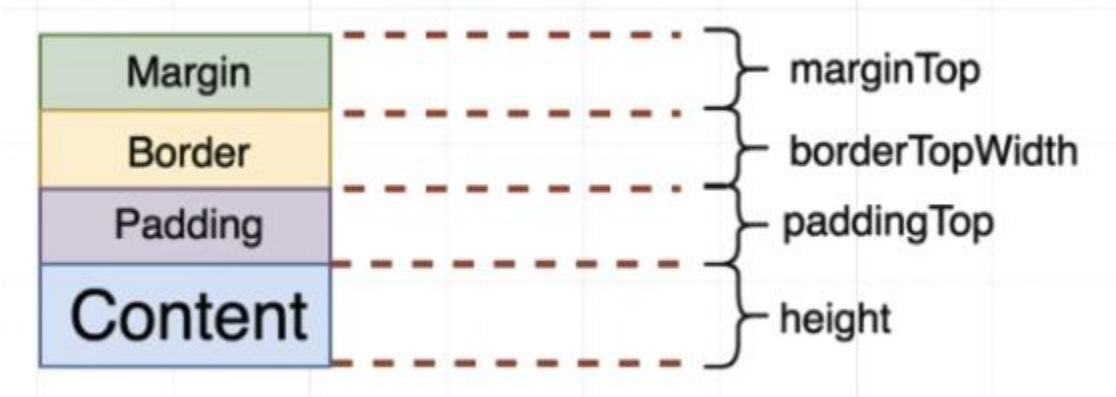
```
1   import React,{ useState } from 'react';
2   import { View, Text, StyleSheet, TextInput } from 'react-native';
3
4   const TextScreen = () => {
5       const [password, setPassword] = useState('');
6       return (
7           <View style={styles.viewStyle}>
8               <Text>Enter password: </Text>
9               <TextInput
10                  style={styles.input}
11                  autoCapitalize="none"
12                  autoCorrect={false}
13                  value={password}
14                  onChangeText={(newValue) => setPassword(newValue)}
15              />
16              {password.length < 4 ? <Text>Password must be 4 characters</Text> : null}
17          </View>
18      );
19  }
20
21  const styles = StyleSheet.create({
22      input: {
23          marginVertical: 15,
24          borderColor: 'black',
25          borderWidth: 1
26      },
27      viewStyle: {
28          margin: 15
29      }
30  });
31
32  export default TextScreen;
```

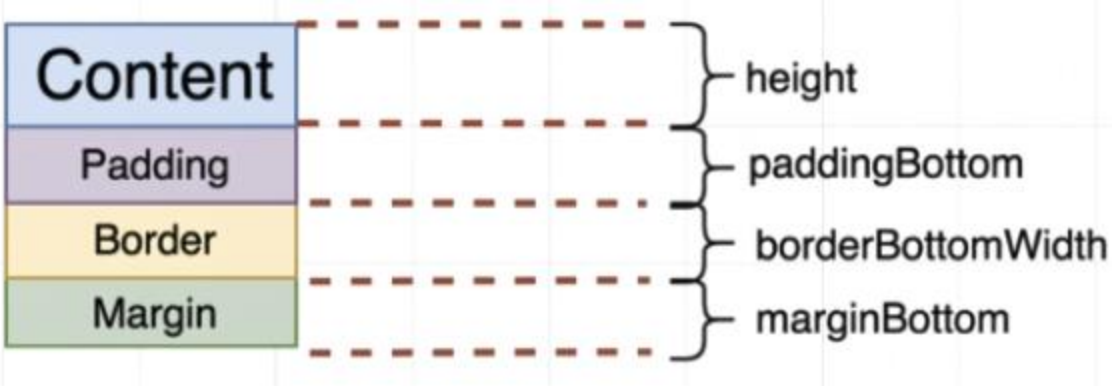
# 7. How to Handle Screen Layout

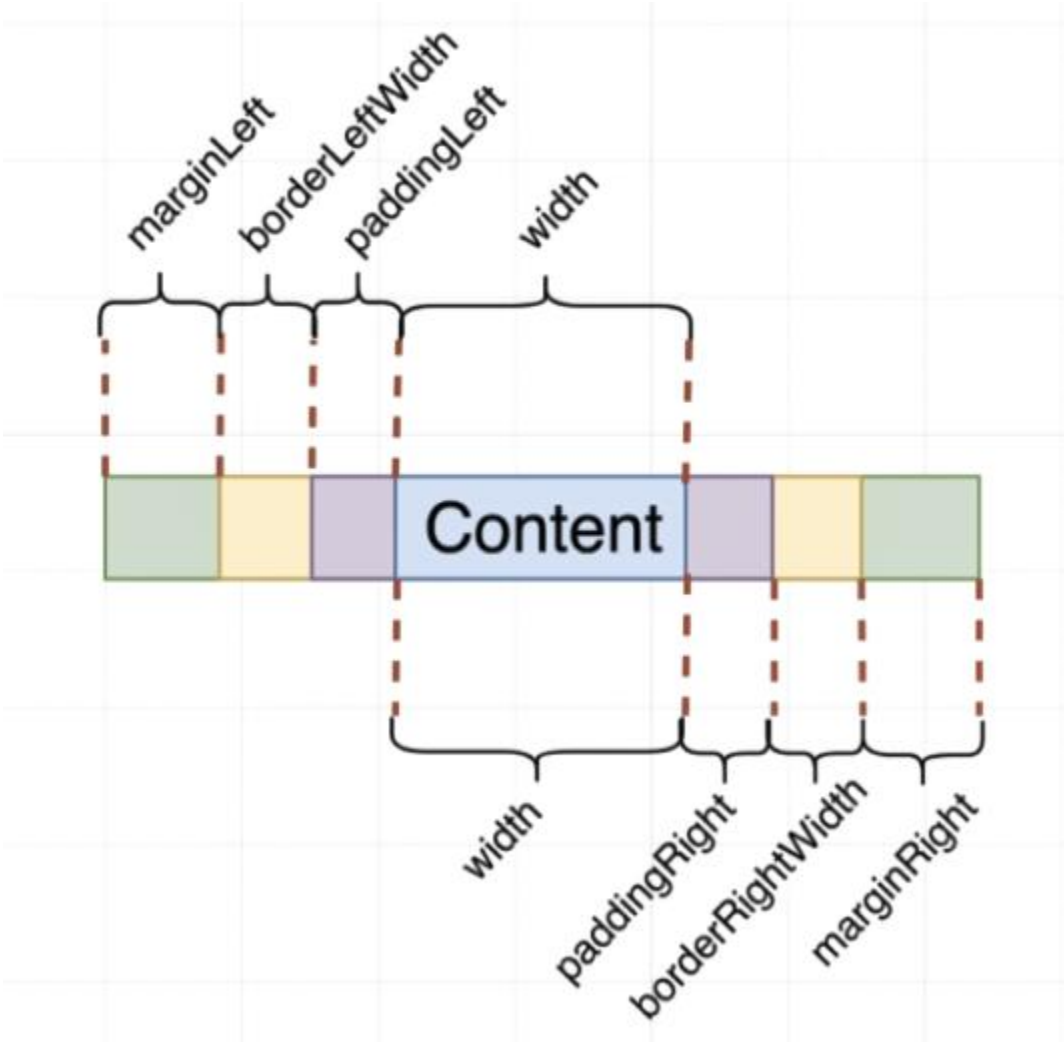



1. Basics of Box Object Model




Content
Padding
Border
Margin
height
paddingBottom
borderBottomWidth
marginBottom


marginLeft
borderLeftWidth
paddingLeft
width
Content
width
paddingRight
borderRightWidth
marginRight

# Shortcuts

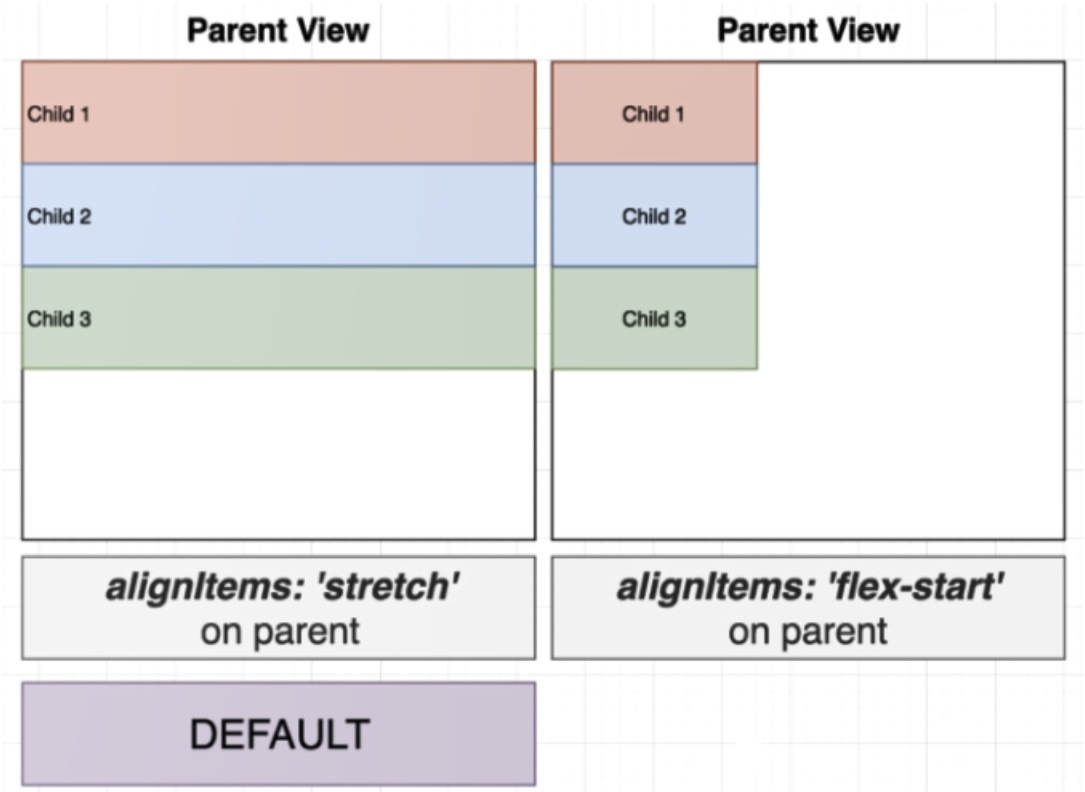
| | |
|---|---|
| **margin** | Set the margin on all sides |
| **marginVertical** | Set the margin on top and bottom |
| **marginHorizontal** | Set the margin on left and right |
| **padding** | Set the padding on all sides |
| **paddingVertical** | Set the padding on top and bottom |
| **paddingHorizontal** | Set the padding on left and right |
| **borderWidth** | Set border width on all sides |

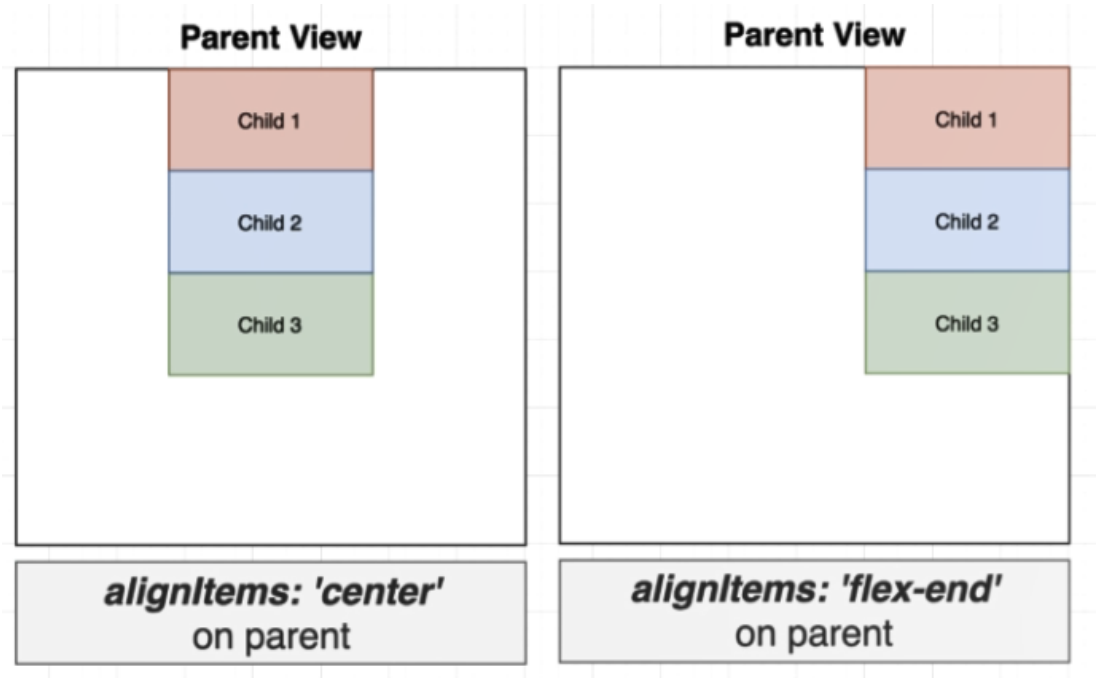
JS BoxScreen.js ●

src > screens > JS BoxScreen.js > ...

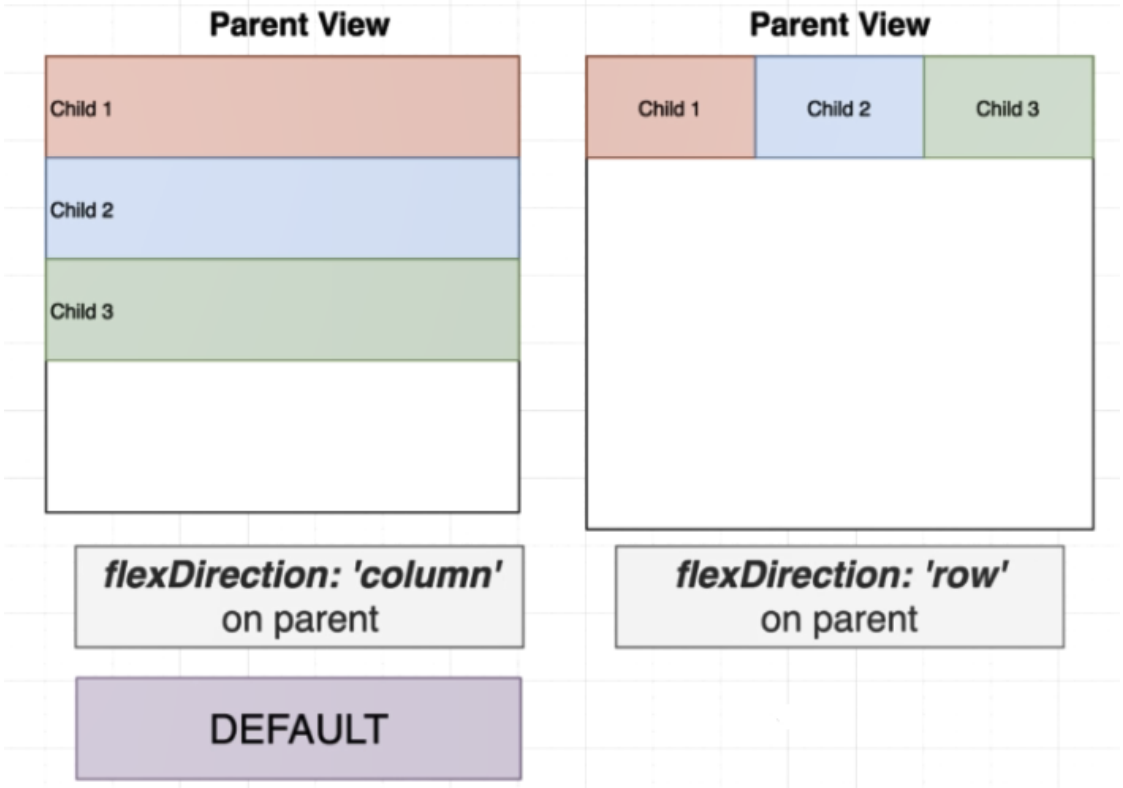
```
1   import React from 'react';
2   import { Text, View, StyleSheet } from 'react-native';
3
4   const BoxScreen = () => {
5       return (
6           <View style={styles.viewStyle}>
7               <Text style={styles.textStyle}>BoxScreen</Text>
8           </View>
9       );
10  };
11
12  const styles = StyleSheet.create({
13      viewStyle: {
14          borderWidth: 1,
15          borderColor: 'black'
16      },
17      textStyle: {
18          borderWidth: 1,
19          borderColor: 'red',
20          margin: 70
21      }
22  });
23
24  export default BoxScreen;
```

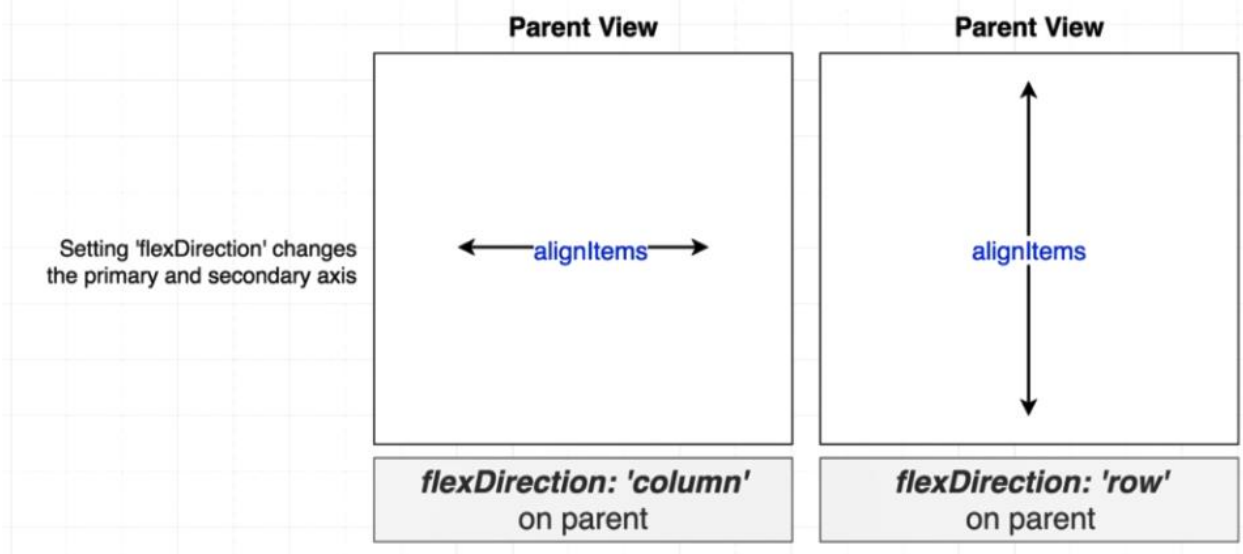
2. Align Items with Flex



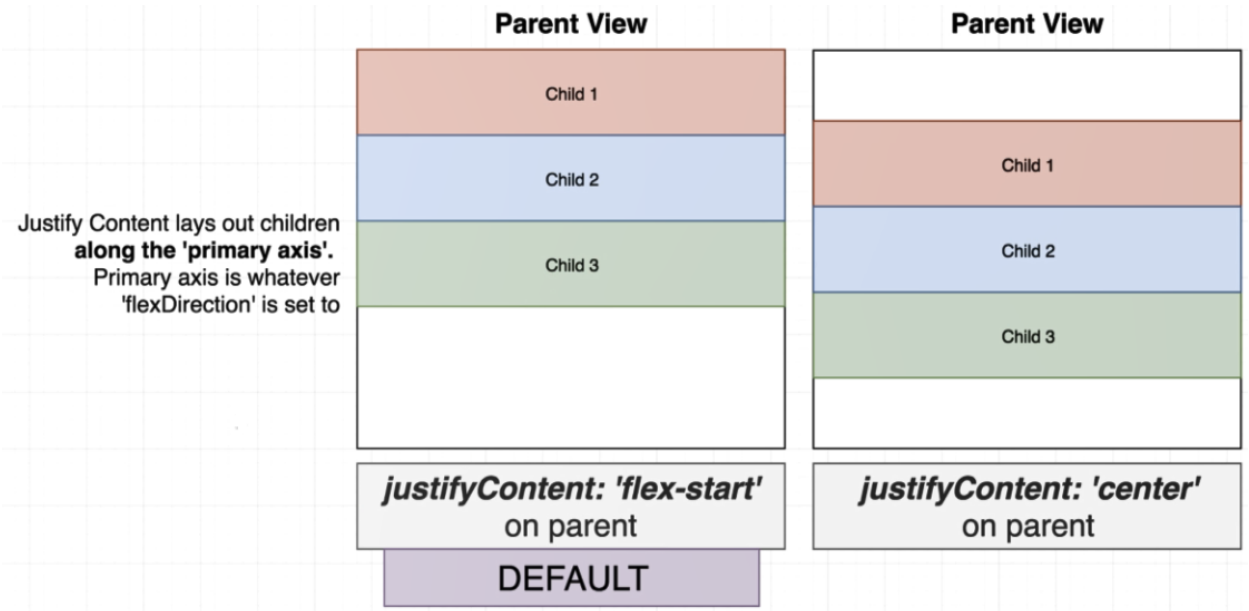



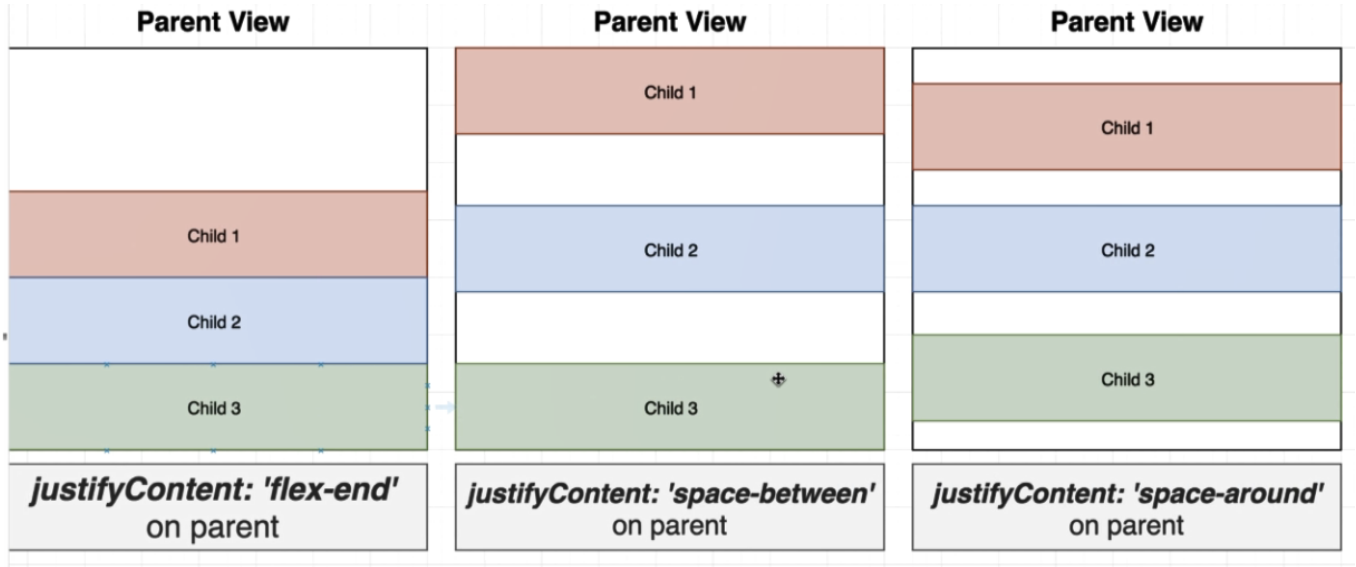
3. Flex Direction



4. Justify Content



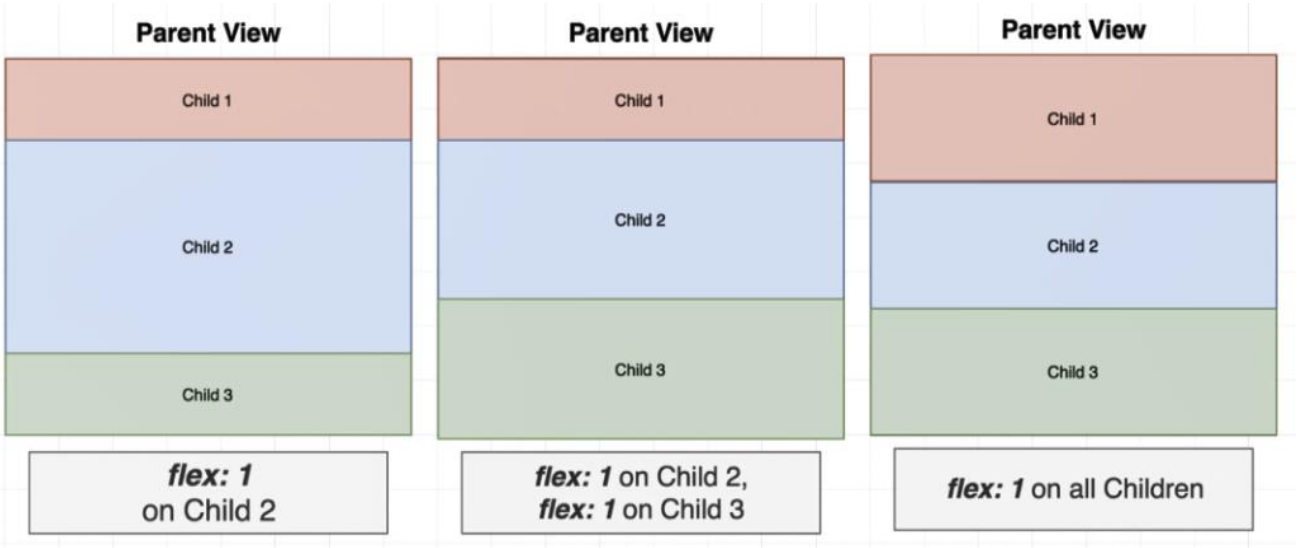



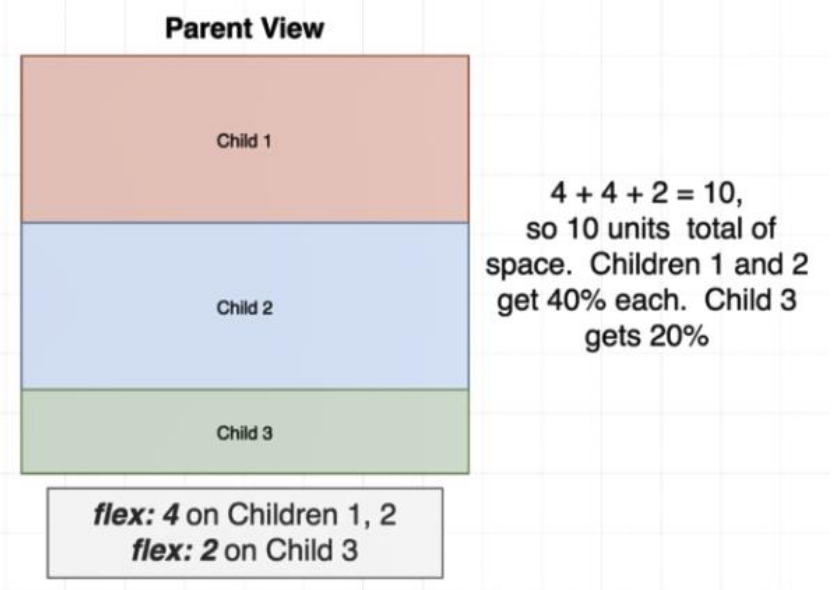
5. Flex Values

| Parent |
| --- |
| alignItems |
| justifyContent |
| flexDirection |

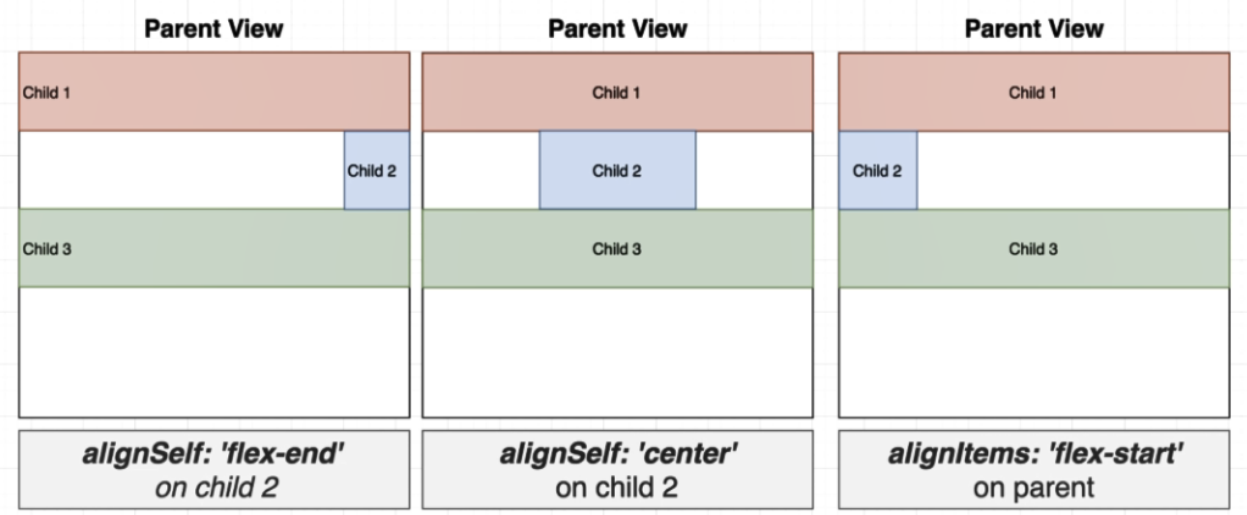
| Child |
| --- |
| flex |
| alignSelf |

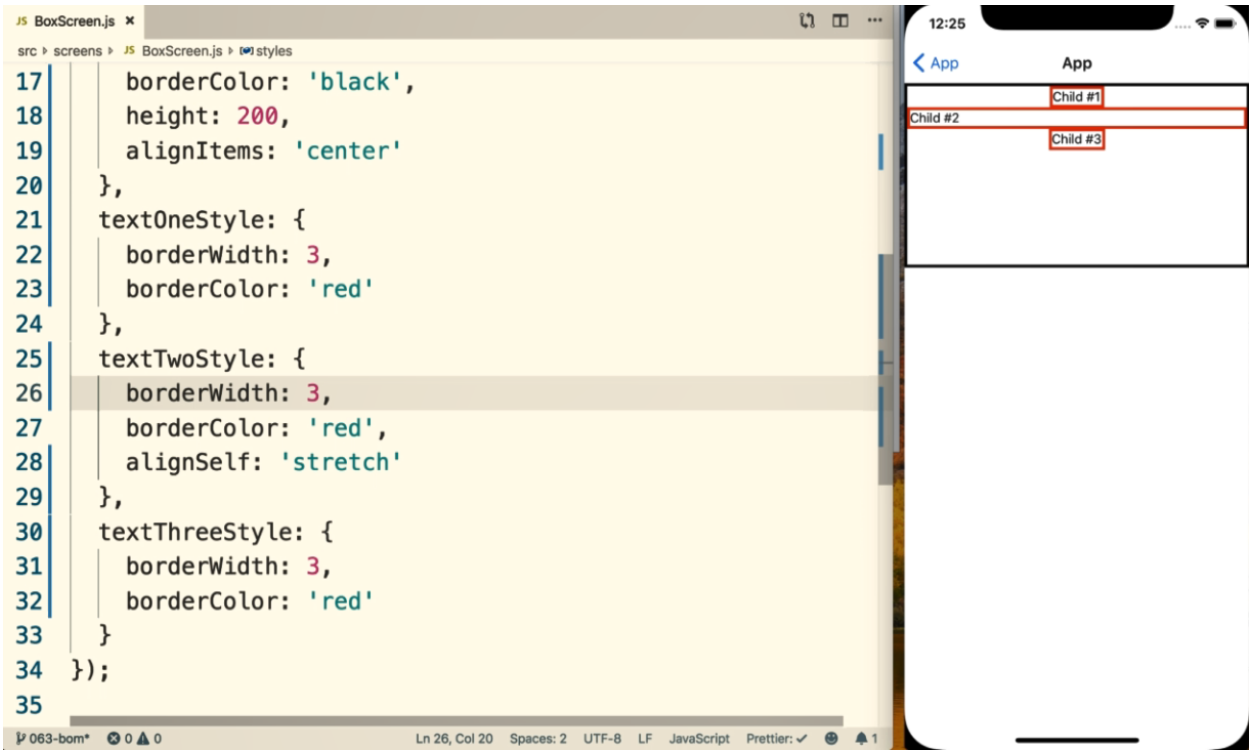
Flex makes a child in a parent try to take up as much space as possible.



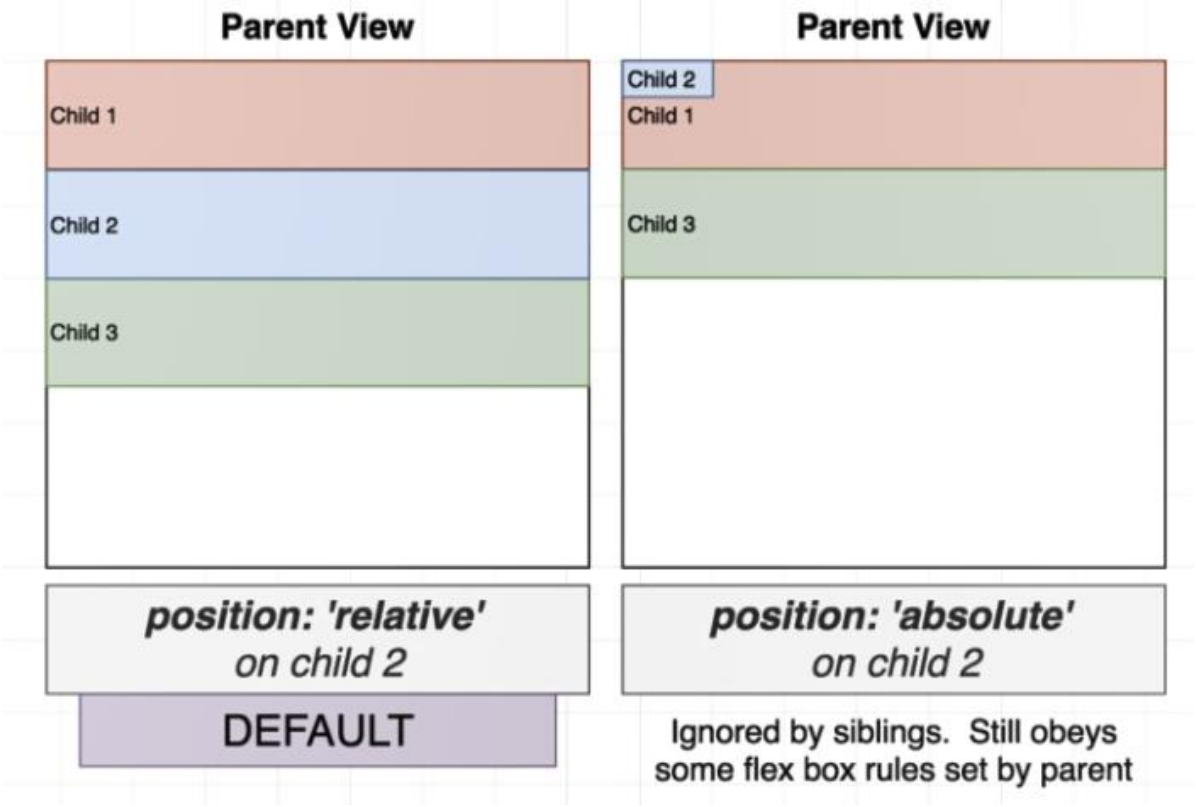



6. Align Self on Children

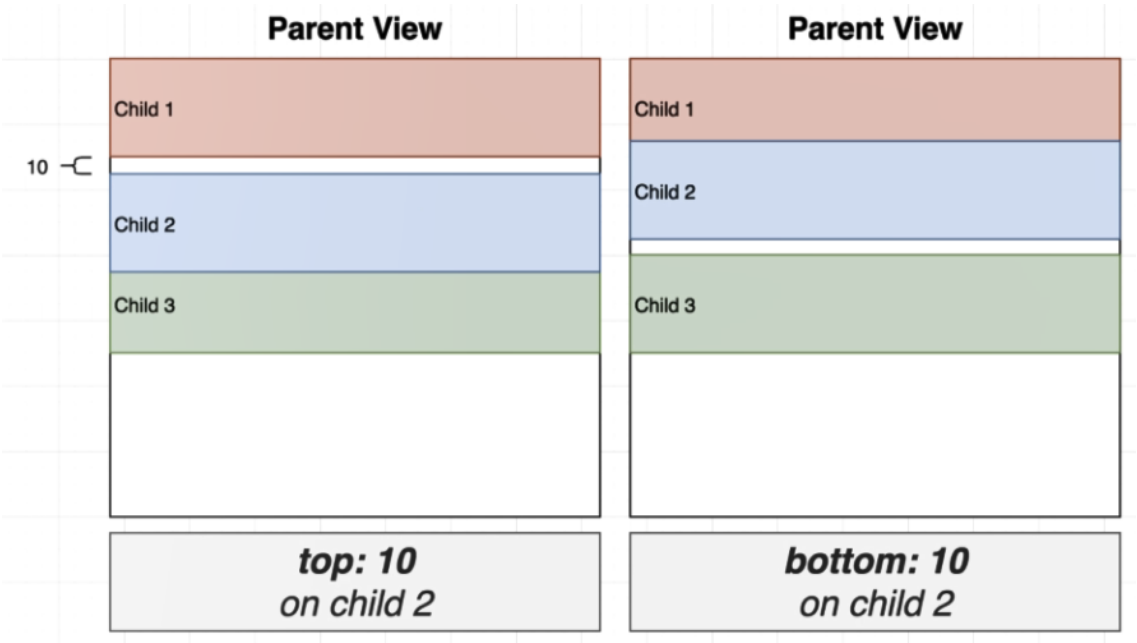


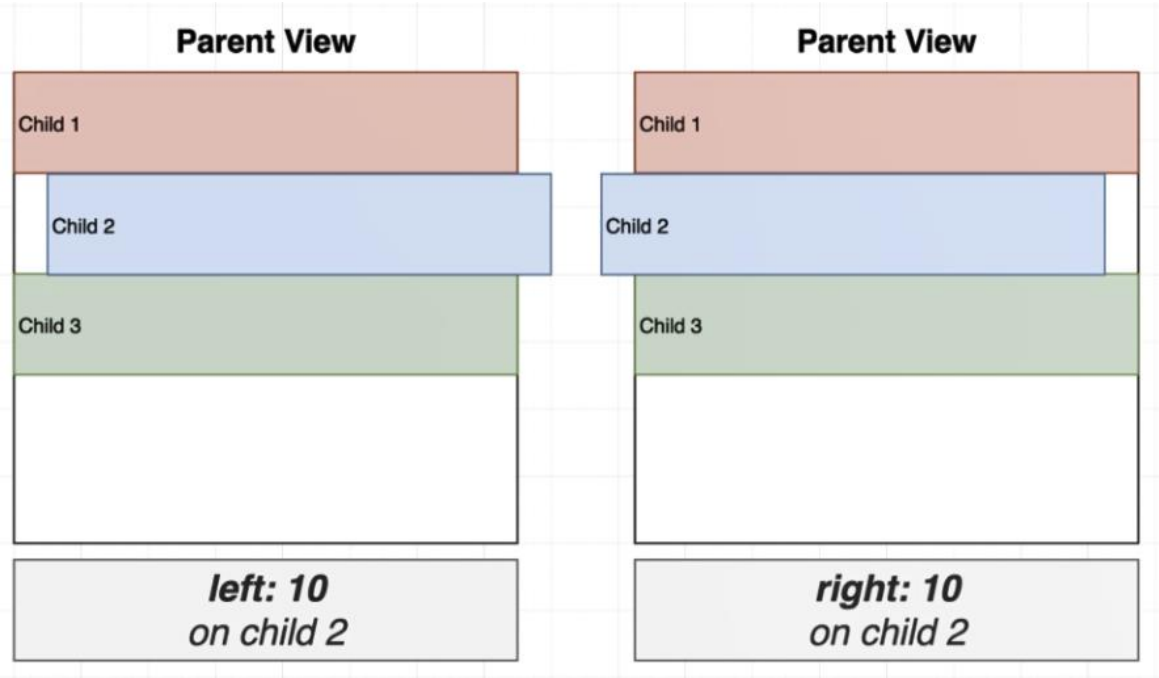



7. The Position Property

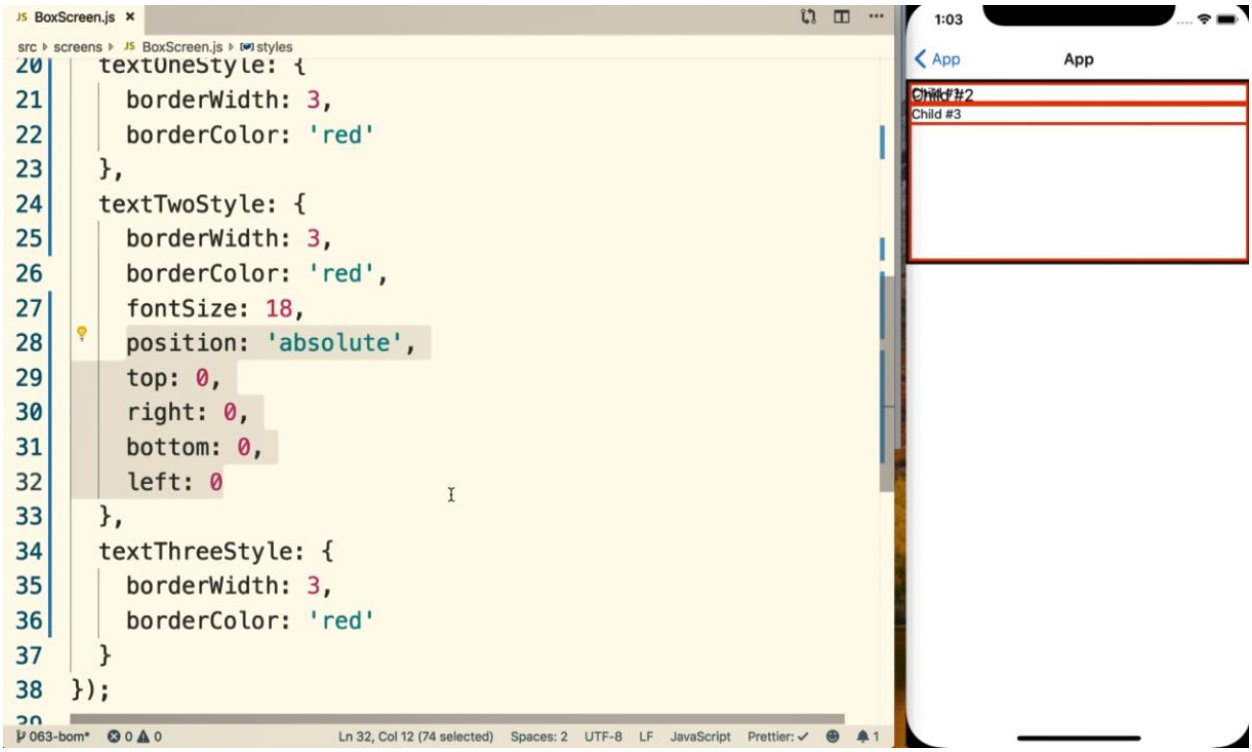



8. Top, Bottom, Left, Right

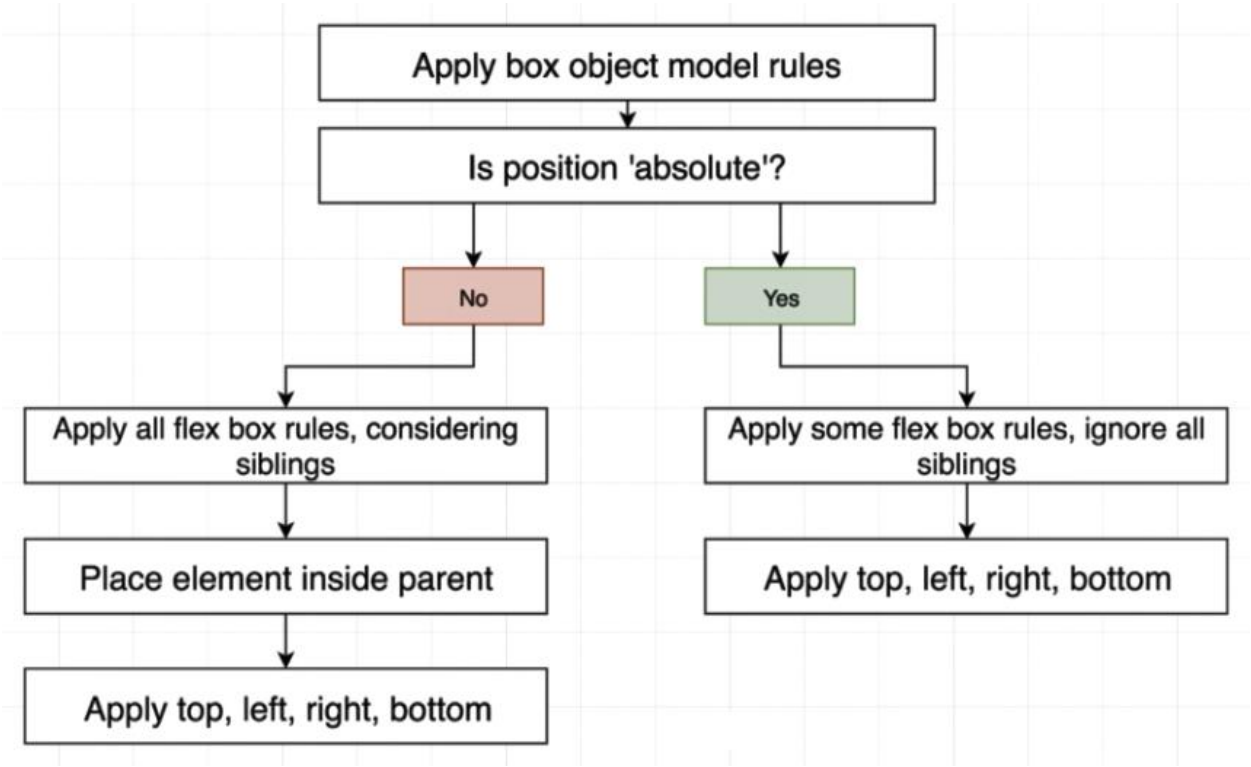




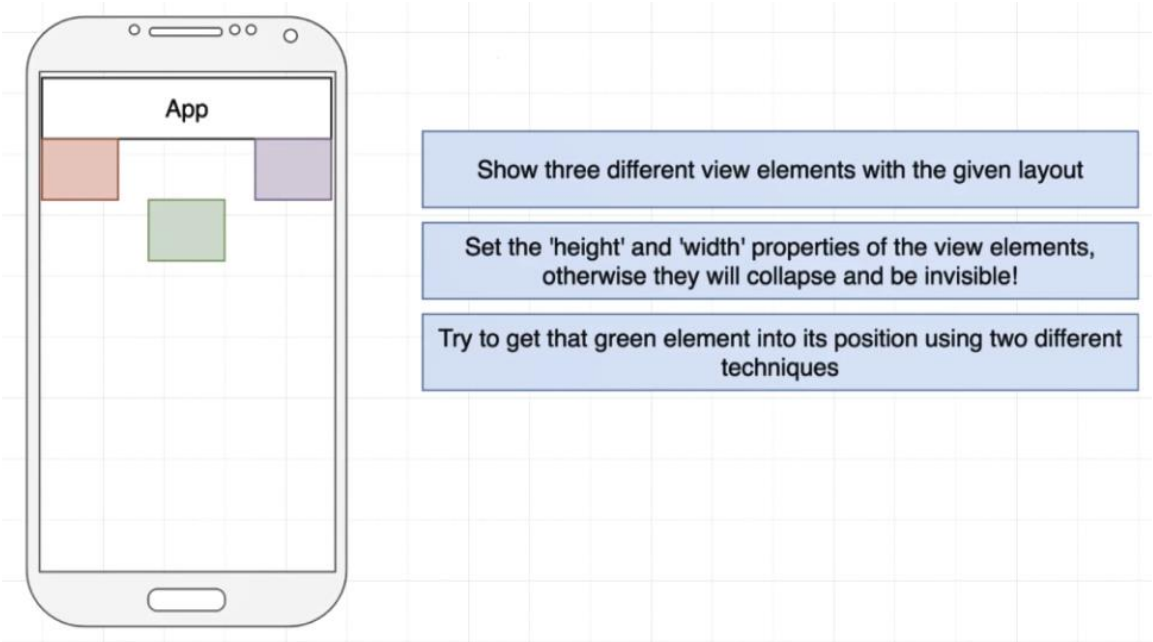

9. Absolute Fill Objects



10. Applying Layout Systems




11. Exercise Solution



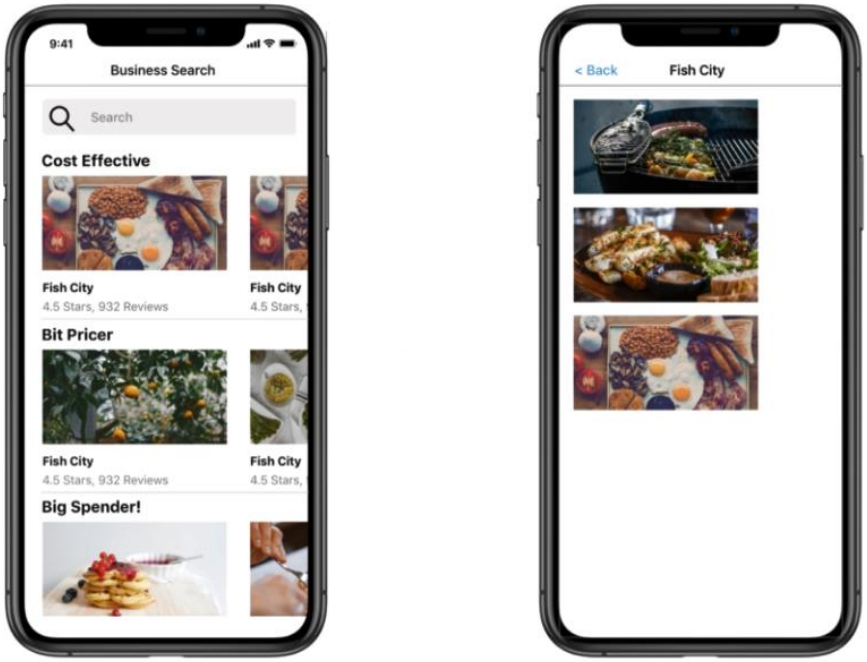
JS BoxScreen.js ×

src > screens > JS BoxScreen.js > styles

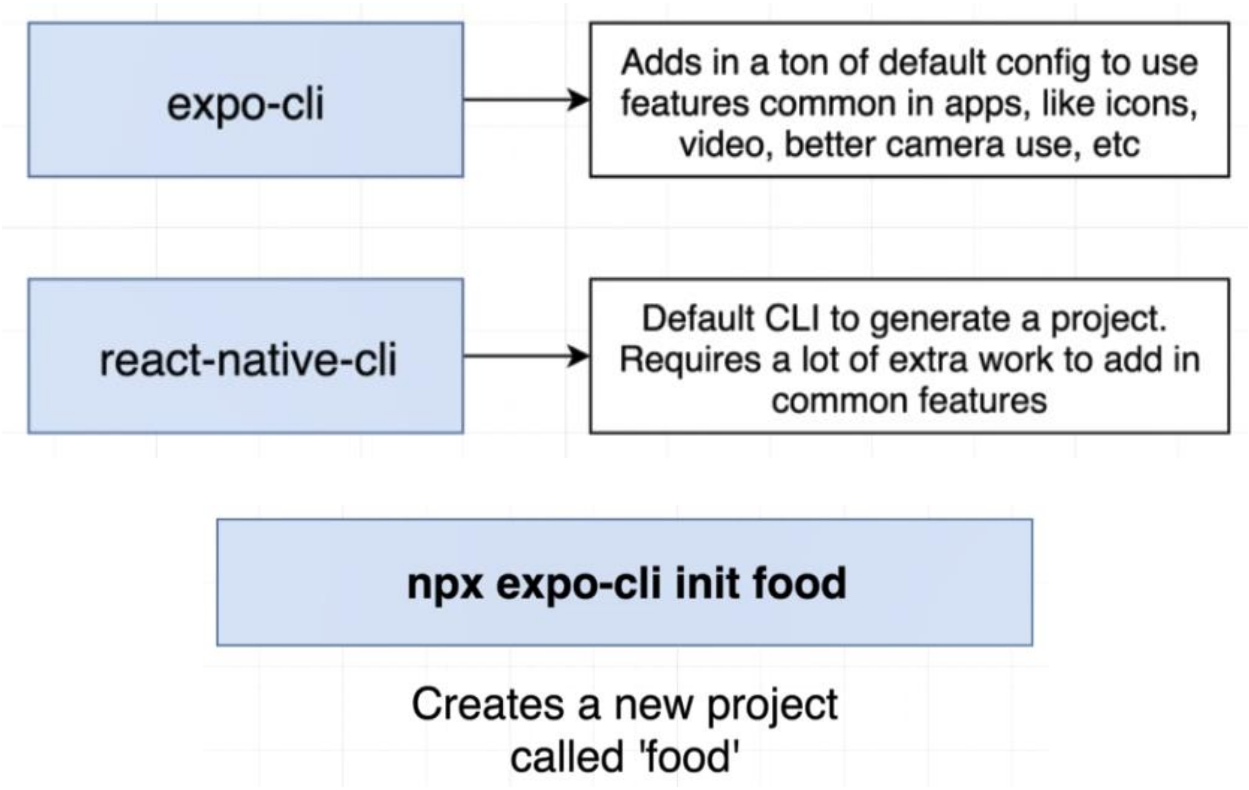
```
import React from 'react';
import { View, StyleSheet } from 'react-native';

const BoxScreen = () => {
    return (
        <View style={styles.parentStyle}>
            <View style={styles.viewOneStyle} />
            <View style={styles.viewTwoStyle} />
            <View style={styles.viewThreeStyle} />
        </View>
    );
};
```

```
const styles = StyleSheet.create({
    parentStyle: {
        borderWidth: 3,
        borderColor: 'black',
        height: 100,
        flexDirection: 'row',
        justifyContent: 'space-between'
    },
    viewOneStyle: {
        height: 50,
        width: 50,
        backgroundColor: 'red'
    },
    viewTwoStyle: {
        height: 50,
        width: 50,
        backgroundColor: 'blue',
        // marginTop: 50
        alignSelf: 'flex-end'
        // top: 50
    },
    viewThreeStyle: {
        height: 50,
        width: 50,
        backgroundColor: 'green'
    }

});

export default BoxScreen;
```

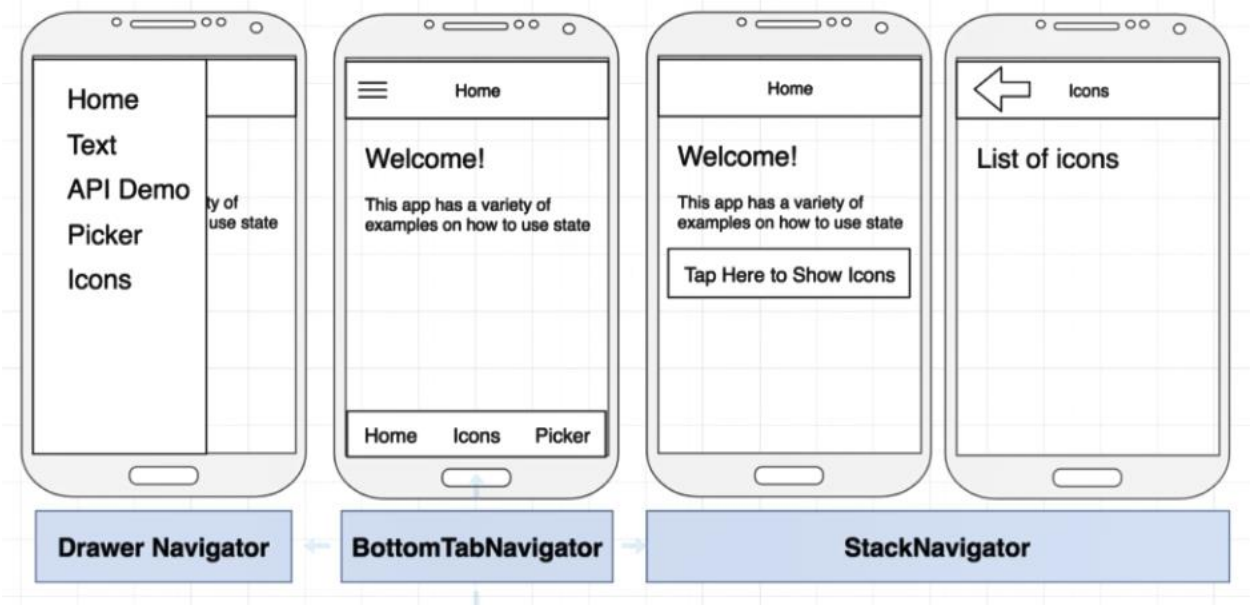
# 8. Putting It All Together - Restaurant Search App




1. Project Generation

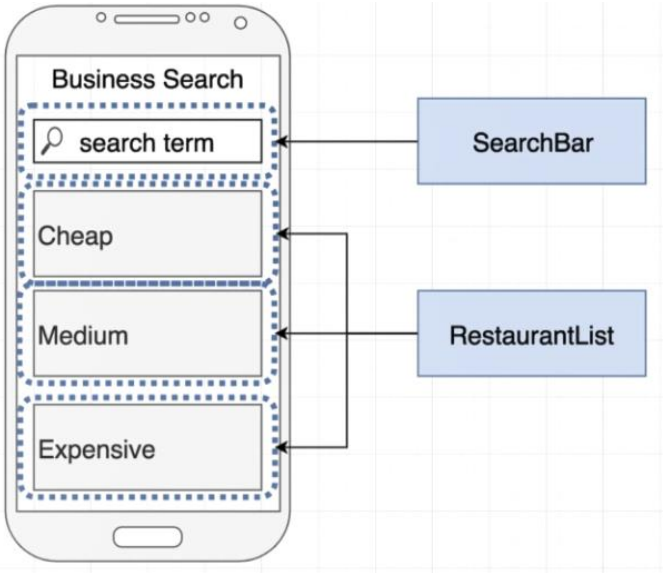


2. React Navigation



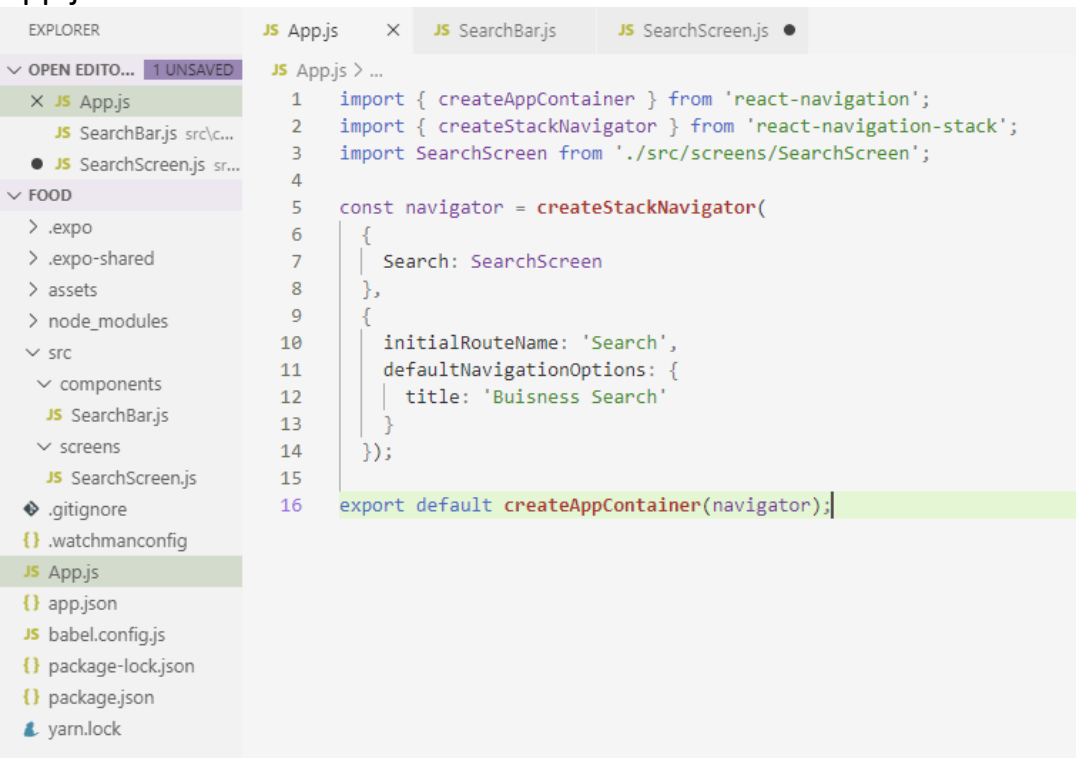

How to install React Navigation: **npm install react-navigation**
React Navigation requires two additional dependencies to work correctly. Be sure to also run this command in your terminal:

**npx expo-cli install react-native-gesture-handler react-native-reanimated**
**yarn add react-navigation-stack**
https://reactnavigation.org/docs/en/stack-navigator.html

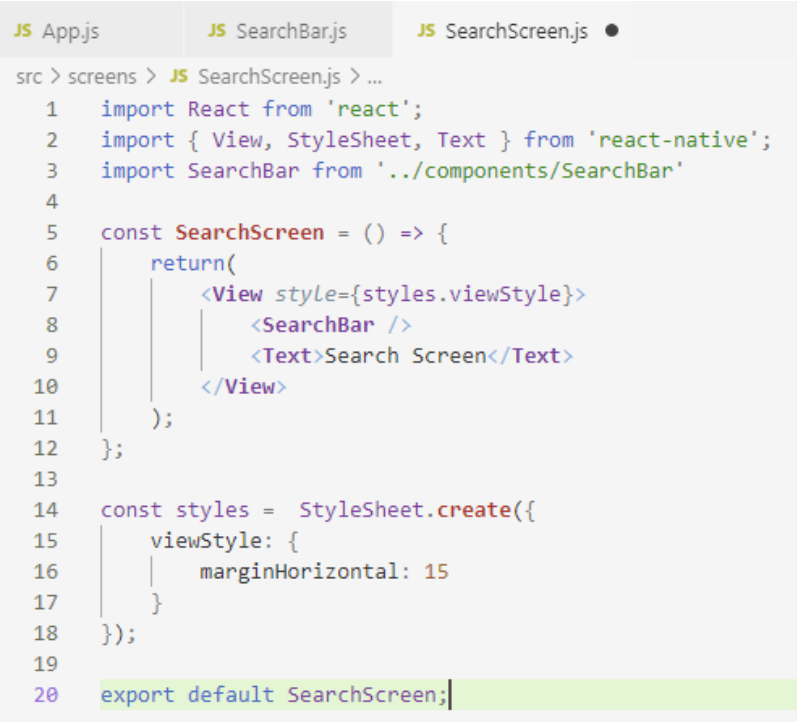


3. Starting the SearchBar

App.js




SearchScreen.js



SearchBar.js

App.js | SearchBar.js | SearchScreen.js

src > components > SearchBar.js > styles

```
import React from 'react';
import { View, StyleSheet, TextInput } from 'react-native';
import { Feather, FontAwesome } from '@expo/vector-icons';

const SearchBar = () => {
    return(
        <View style={styles.backgroundStyle}>
            <Feather name="search" style={styles.iconStyle} />
            {/* <FontAwesome name="Search" /> */}
            <TextInput style={styles.inputStyle}
                placeholder="Search"
            />
        </View>
    );
};
```

```
const styles =  StyleSheet.create({
    backgroundStyle:{
        backgroundColor: '#F0EEEE',
        height: 50,
        borderRadius: 5,
        flexDirection: 'row',
        marginTop: 10,
        marginBottom: 10
    },
    inputStyle: {
        flex: 1,
        fontSize: 18,
        color: '#17202A'
    },
    iconStyle: {
        fontSize: 35,
        alignSelf: 'center',
        marginHorizontal: 10
    }
});

export default SearchBar;
```

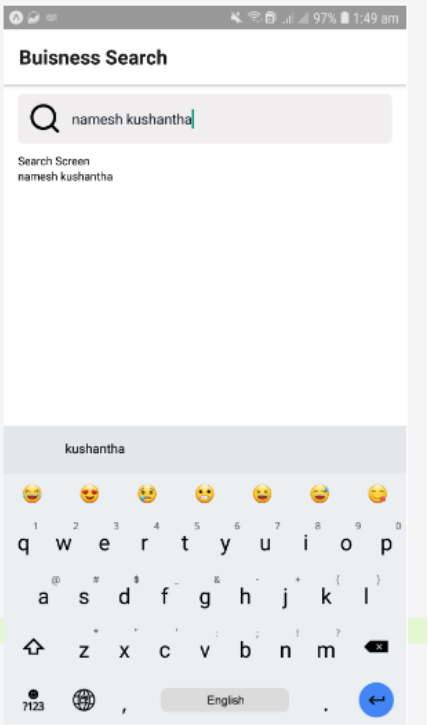
4. Managing State

https://github.com/expo/vector-icons

App.js | SearchBar.js ● | SearchScreen.js

src > components > SearchBar.js > styles

```
1   import React from 'react';
2   import { View, StyleSheet, TextInput } from 'react-native';
3   import { Feather } from '@expo/vector-icons';
4
5   const SearchBar = ({ term, onTermChange, onTermSubmit }) => {
6       return(
7           <View style={styles.backgroundStyle}>
8               <Feather name="search" style={styles.iconStyle} />
9               {/* <FontAwesome name="Search" /> */}
10              <TextInput
11                  autoCapitalize="none"
12                  autoCorrect={false}
13                  style={styles.inputStyle}
14                  placeholder="Search"
15                  value={term}
16                  onChangeText={newTerm => onTermChange(newTerm)}
17                  onEndEditing={() => onTermSubmit()}
18              />
19          </View>
20      );
21  };
22
23  const styles =  StyleSheet.create({
24      backgroundStyle:{
25          backgroundColor: '#F0EEEE',
26          height: 50,
27          borderRadius: 5,
28          flexDirection: 'row',
29          marginTop: 10,
30          marginBottom: 10
```



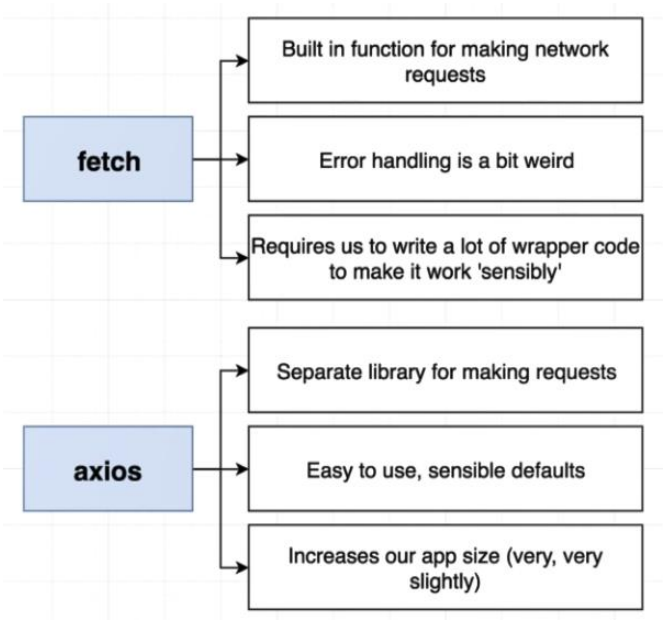

App.js | SearchBar.js ● | SearchScreen.js ×

src > screens > SearchScreen.js > ...

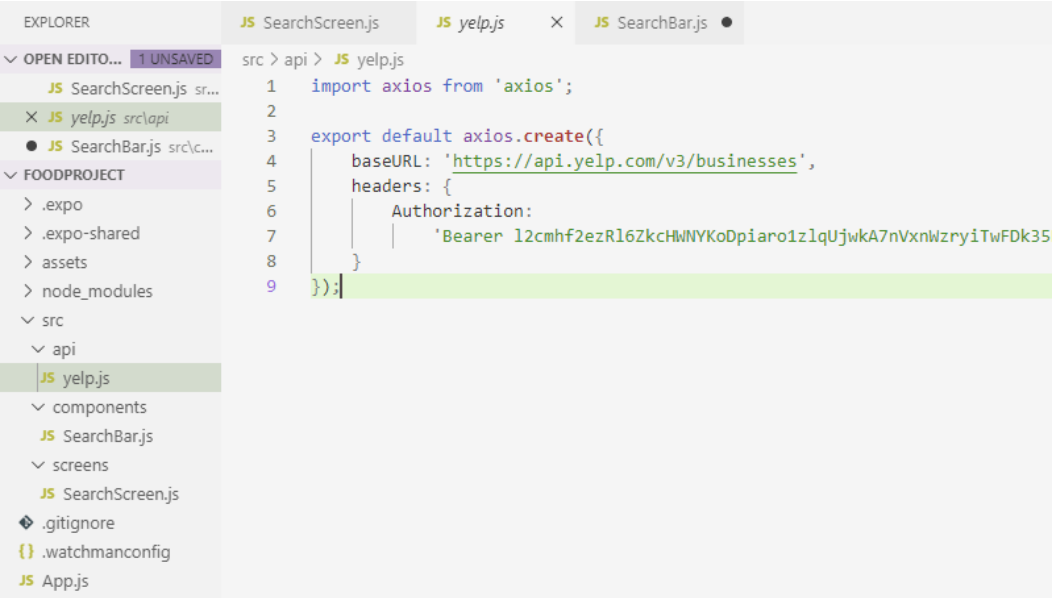
```
1   import React, { useState } from 'react';
2   import { View, StyleSheet, Text } from 'react-native';
3   import SearchBar from '../components/SearchBar'
4
5   const SearchScreen = () => {
6       const [term, setTerm] = useState('');
7       return(
8           <View style={styles.viewStyle}>
9               <SearchBar
10                  term={term}
11                  onTermChange={newTerm => setTerm(newTerm)}
12                  onTermSubmit={() => console.log('term was submitted')}
13              />
14              <Text>Search Screen</Text>
15              <Text>{term}</Text>
16          </View>
17      );
18  };
```

Draw a Relationship Diagram for this code. [ C8 – V13 ]

# 9. Using Outside API's




We use axios for this example: **npm install axios**
And create the yelp.js file in api directory.




Api Key:
l2cmhf2ezRl6ZkcHWNYKoDpiaro1zlqUjwkA7nVxnWzryiTwFDk35PJiucoLhjjFY9ECD
8GTBGaHBg5yv5YDLiszKQx8EMvm30ply0UWoHQOnYFjLozpYnZOx-UsXXYx

1. Making the Request

SearchScreen.js

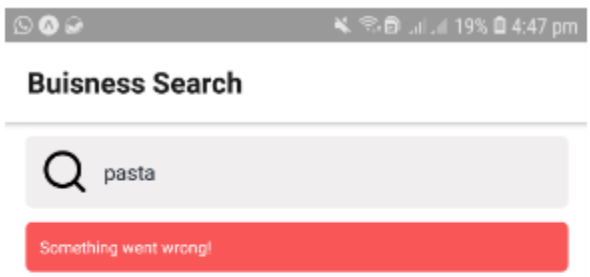
src > screens > SearchScreen.js > SearchScreen > searchApi

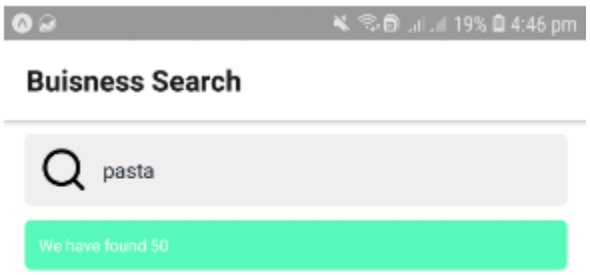
```
import React, { useState } from 'react';
import { View, StyleSheet, Text } from 'react-native';
import SearchBar from '../components/SearchBar';
import yelp from '../api/yelp';

const SearchScreen = () => {
    const [term, setTerm] = useState('');
    const [results, setResults] = useState([]);
    const [errorMessage, setErrorMessage] = useState('');

    const searchApi = async () => {
        try {
            const response = await yelp.get('/search', {
                params: {
                    limit: 50,
                    term,
                    location: 'san jose'
                }
            });
            setResults(response.data.businesses);
        } catch (err) {
            setErrorMessage('Something went wrong!')
        }
    };
```

```
    return (
        <View style={styles.viewStyle}>
            <SearchBar
                term={term}
                onTermChange={newTerm => setTerm(newTerm)}
                onTermSubmit={() => searchApi()}
            />
            {errorMessage
                ? <View style={styles.errorStyle}>
                    <Text style={styles.textStyle}>{errorMessage}</Text>
                  </View>
                : null
            }
            {results.length > 1
                ? <View style={styles.resultsStyle}>
                    <Text style={styles.textStyle}>We have found {results.length}</Text>
                  </View>
                : null
            }
        </View>
    );

};
```

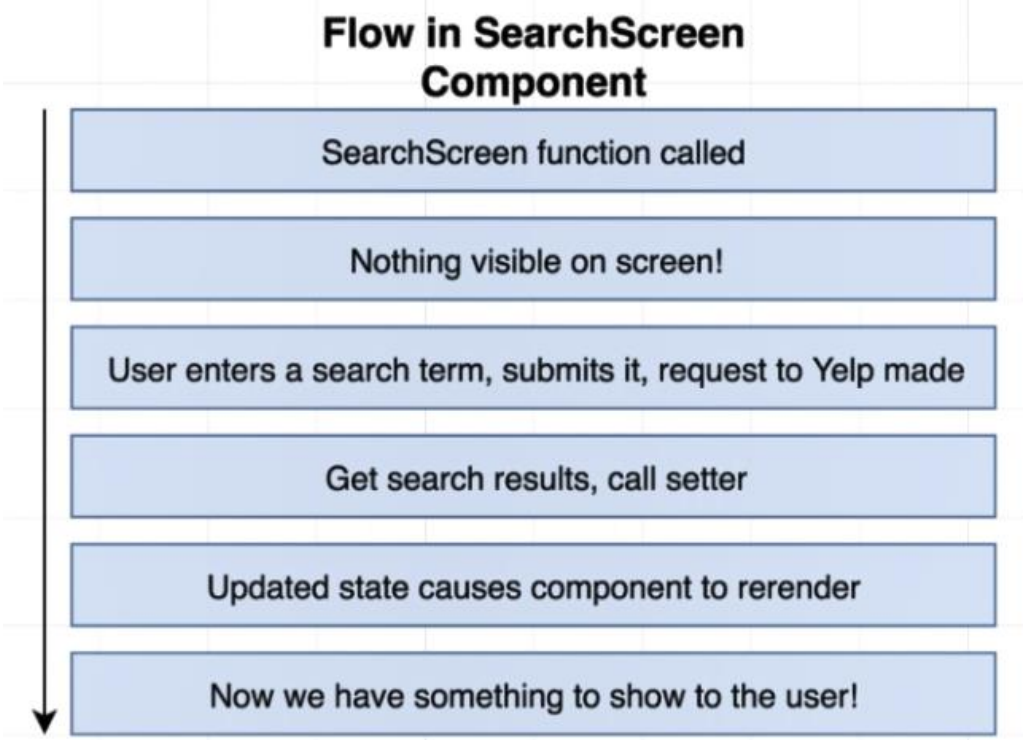
```
50  const styles = StyleSheet.create({
51      viewStyle: {
52          marginHorizontal: 15
53      },
54      errorStyle: {
55          backgroundColor: '#F95757',
56          height: 35,
57          borderRadius: 5,
58          flexDirection: 'row',
59          marginBottom: 10
60      },
61      textStyle: {
62          margin: 10,
63          color: 'white'
64      },
65      resultsStyle: {
66          backgroundColor: '#57F9B9',
67          height: 35,
68          borderRadius: 5,
69          flexDirection: 'row',
70          marginBottom: 10
71      },
72  });
73
74  export default SearchScreen;
```

Results:

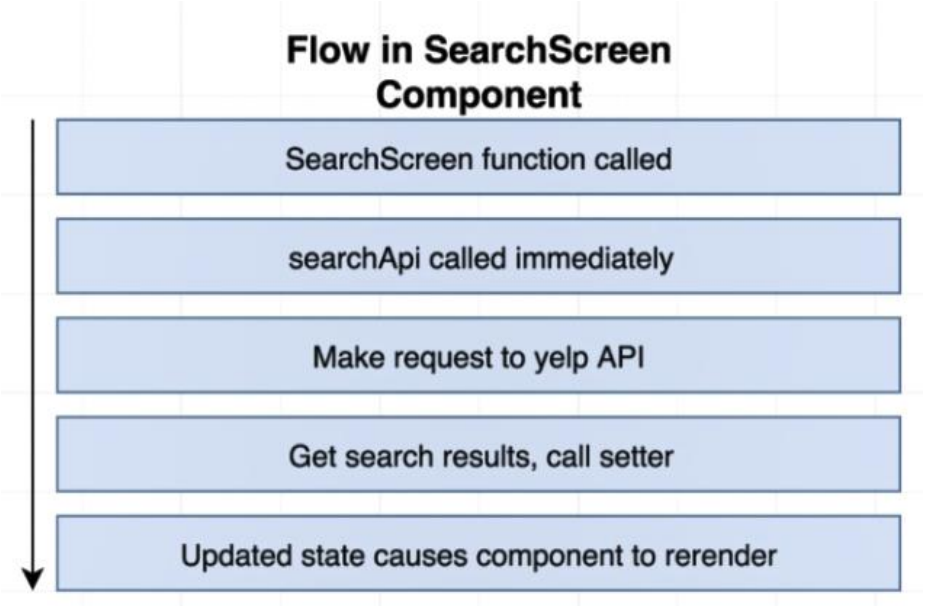


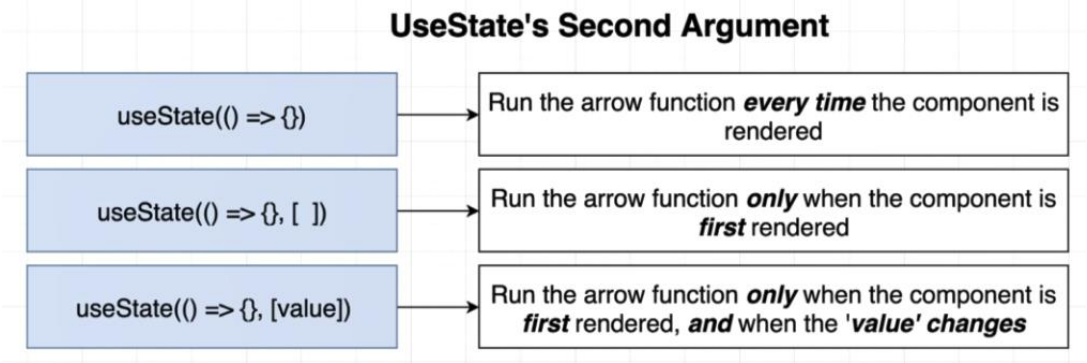



2. Running an Initial Search




How to create a running an initial search






JS SearchScreen.js ● JS yelp.js

src > screens > JS SearchScreen.js > SearchScreen

```
1   import React, { useState, useEffect } from 'react';
2   import { View, StyleSheet, Text } from 'react-native';
3   import SearchBar from '../components/SearchBar';
4   import yelp from '../api/yelp';
5
6   const SearchScreen = () => {
7       const [term, setTerm] = useState('');
8       const [results, setResults] = useState([]);
9       const [errorMessage, setErrorMessage] = useState('');
10
11      const searchApi = async (searchTerm) => {
12          try {
13              const response = await yelp.get('/search', {
14                  params: {
15                      limit: 50,
16                      term: searchTerm,
17                      location: 'san jose'
18                  }
19              });
20              setResults(response.data.businesses);
21          } catch (err) {
22              setErrorMessage('Something went wrong!')
23          }
24      };
25
26      useEffect(() => {
27          searchApi('pasta');
28      }, []);
29
```

```
30      return (
31          <View style={styles.viewStyle}>
32              <SearchBar
33                  term={term}
34                  onTermChange={newTerm => setTerm(newTerm)}
35                  onTermSubmit={() => searchApi(term)}
36              />
```