REACT NATIVE LAYOUT EXAMPLE

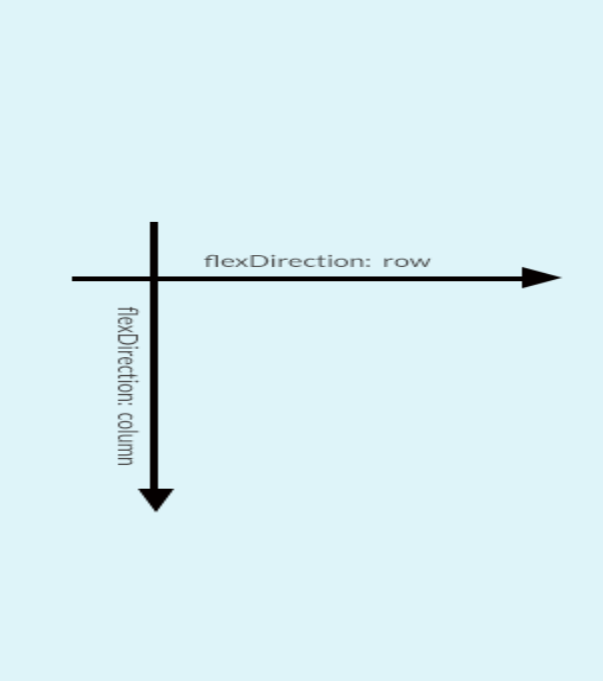
In react native, learning to layout and structure your application will help you create an inspiring and attractive interface your users will love.

React native component uses flexbox to layout its children. Flexbox works the same way in React Native as it does in CSS on the web, with a few exceptions. The defaults are different, with flexDirection defaulting to column instead of row, and the flex parameter only supporting a single number.

Apart from these slight differences, if you have used flexbox in CSS, you will be familiar with it and will get going at the end of this tutorial.

The first thing to have in mind is that react native flexbox uses these properties – flexDirection, alignItems and justifyContent to decide placement of children in a layout.

The flexDirection of a component determines the ***primary axis of its layout***. If the flexDirection is column then the axis will be vertical otherwise it will be horizontal. The diagram below illustrate how flexDirection, primary and secondary axis are related.



Let start with a simple example with the default flexDirection column. We are going to add three child Views inside a parent View.

The flex style property does not accept percentage and values with unit rather it uses the concept of ratio to apportion space to child components.

The main or parent content View is assigned a flex value of 1. This implies that the component will fill all the available space in the window.

Let try and see some example that will help us understand this concept much better.

CREATE A NEW REACT NATIVE APP

In other to follow this tutorial and get your hand dirty with code, create a new react native application and give it a name of your choice.

If you have not created a react native application before, you can follow my tutorial on React Native setup to get going.
Once you are done with creating your react native project, run either of these command depending on the platform you are targeting.

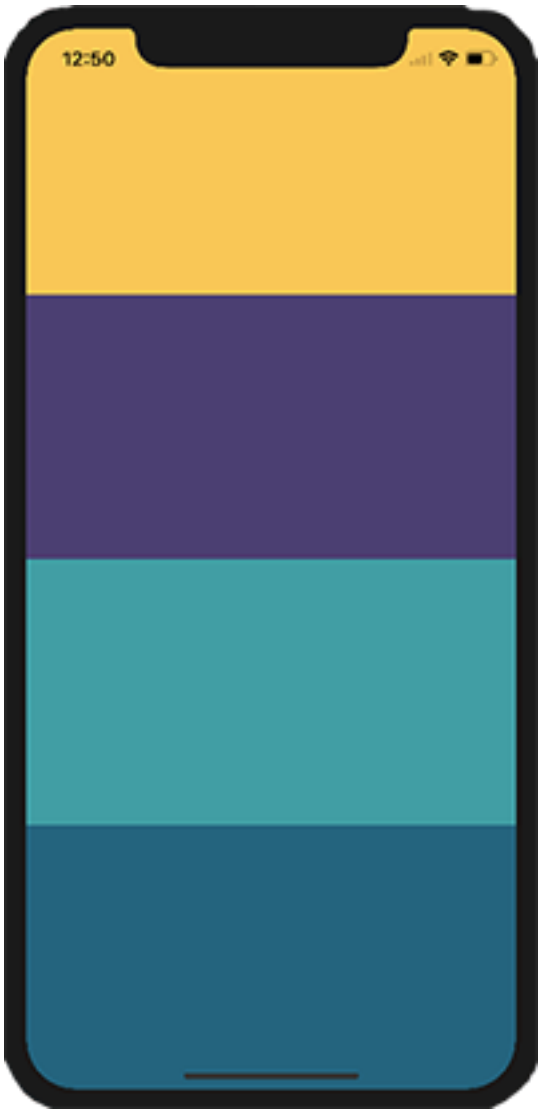
```
React-native run-ios or react-native run-android
```

Now that you have started your emulator and you can see the default boilerplate page that comes with react native project setup, Go to the root directory of your project, open the app.js file in any Javascript IDE of your choice.

Copy the code below and paste it in the file and the reload your project. If everything works out for you, you will see a screen-shot like below.

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


export default class App extends Component<{}> {
  render() {
    return (
      <View style={styles.container}>
      <View style={styles.firstrow}></View>
      <View style={styles.secondrow}></View>
      <View style={styles.thirdrow}></View>
      <View style={styles.fourthrow}></View>
      </View>
    );
  }
```

```
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#1f2041',
  },

  firstrow: {
    flex: 1,
    backgroundColor: "#ffc857"
  },

  secondrow: {
    flex: 1,
    backgroundColor: "#4b3f72"
  },

  thirdrow: {
    flex: 1,
    backgroundColor: "#119da4"
  },

  fourthrow: {
    flex: 1,
    backgroundColor: "#19647e"
  }

});
```



FLEXDIRECTION ROW

Let try and change the flexDirection of the parent component from the default column to row.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row',
    backgroundColor: '#1f2041',
  },
}
```

Below is the result of the changes.

Now that we have an idea of how flex works, we will move over to justifycontent. The code snippet is below and note that the only thing we will be changing is the justifyContent value.

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

```
export default class App extends Component<{}> {
  render() {
    return (
      <View style={styles.container}>
      <View style={styles.firstrow}></View>
      <View style={styles.secondrow}></View>
      <View style={styles.thirdrow}></View>
      <View style={styles.fourthrow}></View>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 40,
    backgroundColor: '#1f2041',
    justifyContent: 'flex-start'
  },

  firstrow: {
    width: 160,
    height: 160,
    backgroundColor: "#ffc857"
  },

  secondrow: {
    width: 160,
    height: 160,
    backgroundColor: "#4b3f72"
  },

  thirdrow: {
    width: 160,
    height: 160,
    backgroundColor: "#119da4"
  },

  fourthrow: {
    width: 160,
    height: 160,
```

```
    backgroundColor: "#19647e"
  }

});
```

JUSTIFYCONTENT

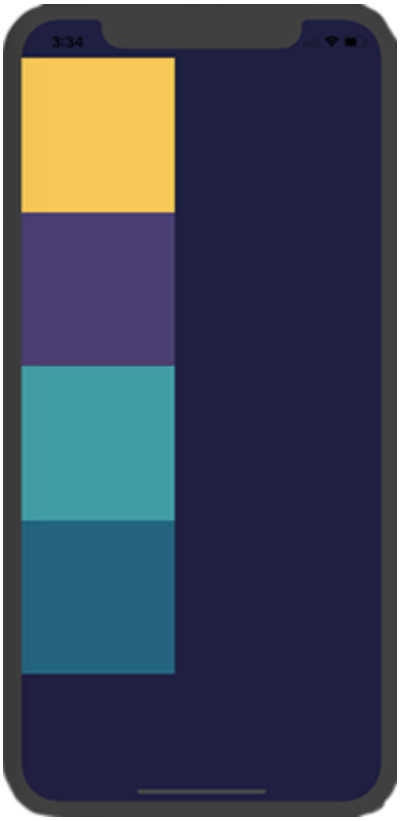
When you want to distribute child components in their parent component along the primary axis you will need to use justifyContent to achieve that.

In other to see the effect of this style property, we are going to assign our child component's width and height some values because it is very hard to notice any effect if we use flex since the component will fill up the width and height space of the component.

JustifyContent has five values which we will see how they differ from each other. They are `flex-start, center, flex-end, space-around and space-between`

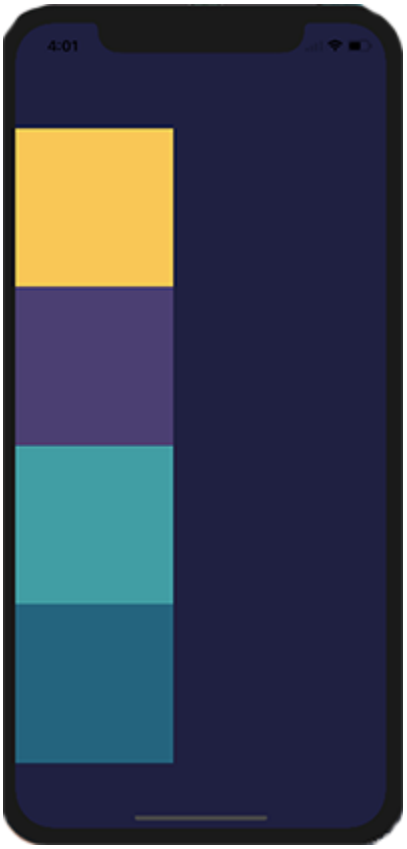
Flex-start

In flex-start, the child components are align at the beginning of the page. Each subsequent component follows the one proceeding it. You can see that a space is left at the bottom of the page.



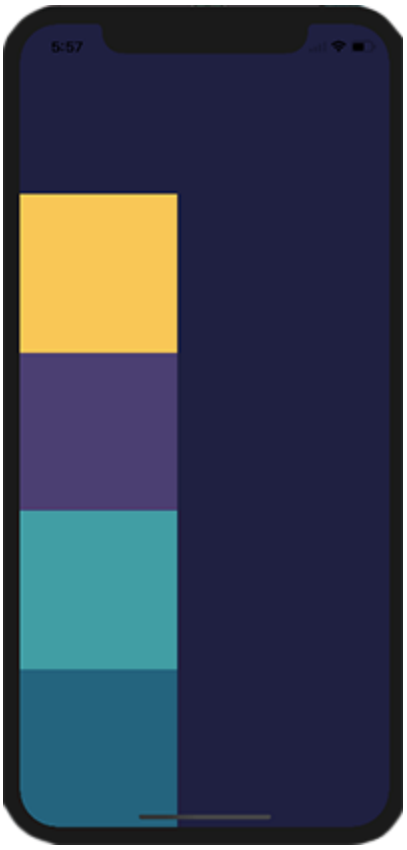
Center

When you pass a value of center to justifyContent , the child view are placed in the center of the screen either vertical or horizontal depending on the primary axis direction. Notice the equal space distribution in the top and bottom of the page.



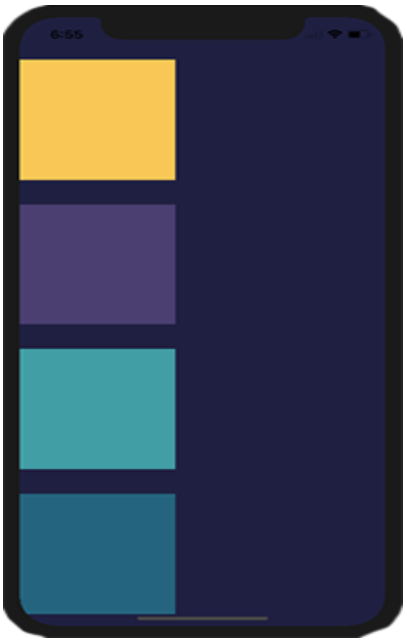
Flex-end

When you set the value of justifyContent to flex-end, it will give you the opposite of what you get when you use flex-start. Instead of placing the starting point of the child components at the top, it will push them down to the bottom.



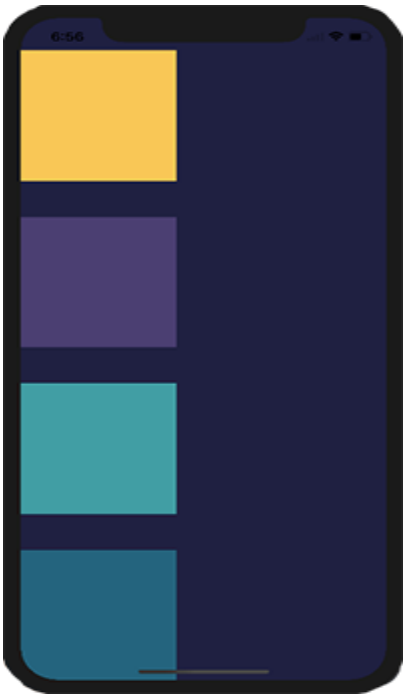
Space-around

With space-around, all child components are distributed with equal space around them.



Space-between

With space-width, the child components are given equal space between then as shown in the diagram below.



ALIGNITEMS

AlignItems is use to distribute child components along the secondary axis. Like the justifyContent, alignItems can take any of the following values – flex-start, center, flex-end and stretch

For stretch to have an effect, children must not have a fixed dimension along the secondary axis. In the following, setting alignItems: stretch does nothing until the width: 50 is removed from the children.

flex-start center flex-end

Other Option to distribute child components in react native

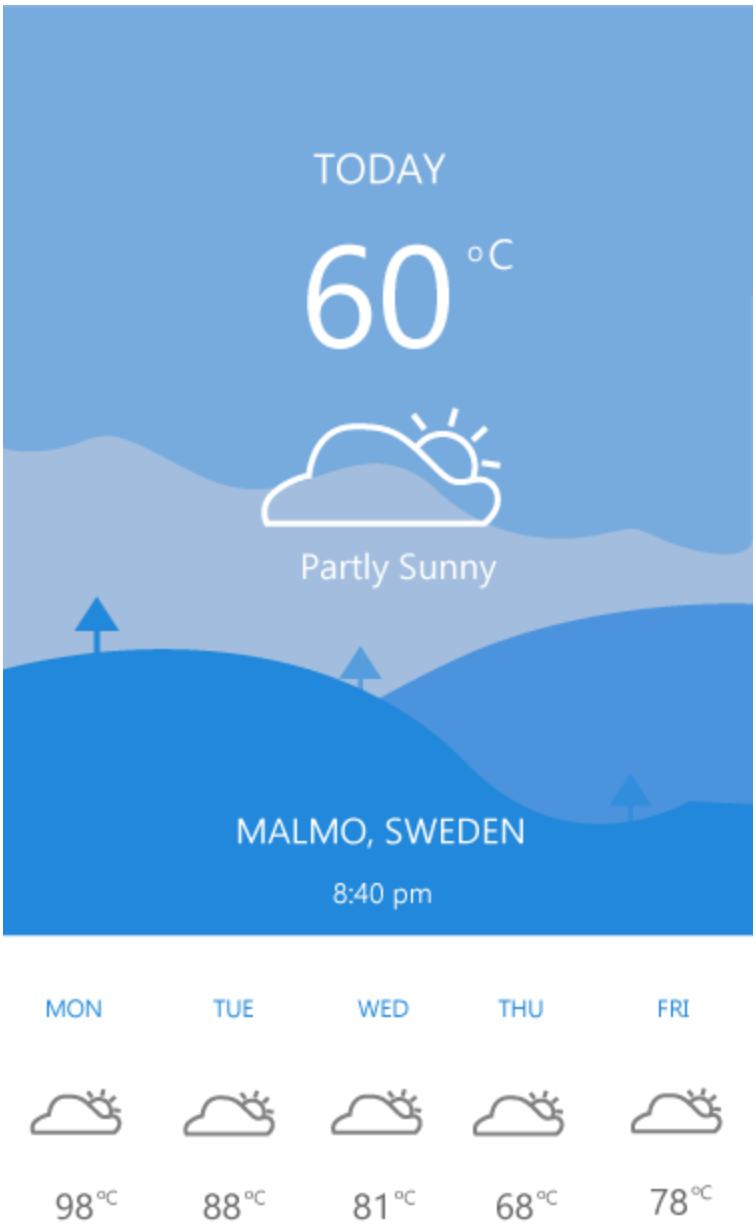
Apart from what we have discussed here, there are other few options you can employ to create a complex layout in your application. We will not discuss them but if you want to read more on this topic you can go the *react native developer page*

LET CREATE A REACT NATIVE INTERFACE BASE ON WHAT WE HAVE LEARNED

We have covered lots of ground on react native layout but if we go home now we might not remember some of the things we have learnt. Not just that, for us to become better and better, we need to practice as much as possible.

With this in mind, we are going to create a simple weather app user interface.

The screen-shot of what we will create is shown below.



Create a react native project and name it Weatherappui or any name of your choice.

Once you are done, open the App.js file and copy the code below and paste in the file.

```
import React, { Component } from 'react';
import {
  Platform,
```

```
  StyleSheet,
  Image,
  Text,
  View,
  ImageBackground
} from 'react-native';

import DegreeComponent from './app/DegreeComponent';


export default class App extends Component<{}> {
  render() {

    return (
      <View style={styles.container}>
      <ImageBackground style={styles.upperregion} source={require('./assets/mybg.png')}>
      <View style={styles.innerupperregion}>
        <Text style={styles.today}> TODAY </Text>
        <Text style={styles.degrees}> 23<DegreeComponent> </DegreeComponent>C</Text>

        <Image style={styles.weathericon} source={require('./assets/weathericon.png')}/>
        <Text style={styles.weathercondition}>Partly Sunny</Text>
      </View>

      <View style={styles.lowerinnerregion}>
      <Text style={[styles.addwhite, {fontSize: 18}]}>MALMO, SWEDEN </Text>
      <Text style={[styles.addwhite, {fontSize: 14}]}>8:40 pm</Text>
      </View>

      </ImageBackground>

      <View style={styles.lowerregion}>

      <View style={styles.fivecolumns}>
      <Text style={styles.columntitle}>MON</Text>
      <Image style={styles.iconsmall} source={require('./assets/smallicon.png')} />
      <Text style={styles.temp}>6/1</Text>
      </View>

      <View style={styles.fivecolumns}>
      <Text style={styles.columntitle}>TUE</Text>
      <Image style={styles.iconsmall} source={require('./assets/smallicon.png')} />
      <Text style={styles.temp}>6/4</Text>
      </View>
```

```
    <View style={styles.fivecolumns}>
    <Text style={styles.columntitle}>WED</Text>
    <Image style={styles.iconsmall} source={require('./assets/smallicon.png')} />
    <Text style={styles.temp}>6/20</Text>
    </View>

    <View style={styles.fivecolumns}>
    <Text style={styles.columntitle}>THU</Text>
    <Image style={styles.iconsmall} source={require('./assets/smallicon.png')} />
    <Text style={styles.temp}>5/15</Text>
    </View>

    <View style={styles.fivecolumns}>
    <Text style={styles.columntitle}>FRI</Text>
    <Image style={styles.iconsmall} source={require('./assets/smallicon.png')} />
    <Text style={styles.temp}>7/18</Text>
    </View>
    </View>

    </View>
   );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#ffffff',
  },
  upperregion: {
    flex: 2,
    alignItems: 'center',
  },

  lowerregion: {
    flex: 1,
    flexDirection: 'row',
    backgroundColor: '#f4f4f4'
  },

  innerupperregion: {
    marginTop: 120,
    backgroundColor: 'transparent',
```

```
  alignItems: 'center',
  flex: 2
},

lowerinnerregion: {
  flex: 1,
  justifyContent: 'flex-end',
  backgroundColor: 'transparent',
  alignItems: 'center',
  paddingBottom: 20
},

today: {
  fontSize: 21,
  color: 'white'
},

degrees: {
  fontSize: 72,
  color: 'white'
},

weathericon: {
  width: 120,
  height: 60,
  marginTop: 40,
},

weathercondition: {
  fontSize: 18,
  color: 'white',
  marginTop: 20,

},

addwhite: {
  color: 'white'
},

fivecolumns: {
  flex: 1,
  paddingTop: 40,
  alignItems: 'center'
},
```

```
  columntitle: {
    fontSize: 13,
    color: '#2188db',
    fontWeight: 'bold'
  },

  iconsmall: {
    width: 56,
    height: 32,
    marginTop: 30
  },

  temp: {
    fontSize: 16,
    color: '#666666',
    marginTop: 20
  }

});
```

Take time and go through the code. You will see how we have used all the knowledge we got from react native layout tutorial to create this weather app interface.

To create the degree sign, we have to create our own custom component.

Go to your react native project root directory, create a new folder called app.

Inside the app folder, create a new file and name it DegreeComponent.js.

Open the created file and paste the code below in it.

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


class DegreeComponent extends Component{

    render(){
```

```
      return(
        <View style={styles.circleicon}>
        </View>
      );
   };


}

const styles = StyleSheet.create({

   circleicon: {
      width: 10,
      height: 10,
      borderRadius: 5,
      borderWidth: 2,
      borderColor: '#FFFFFF',
      marginTop: -50
   },

});

module.exports = DegreeComponent;
```

You will also need to create another folder in the project root directory but now you can name it assets. This is where we will keep all our project assets like images.

Copy the required images and paste them inside the assets folder.

Finally, run the application. If everything works out for you, you will get a similar interface like the one above.