Разработка мобильных приложений

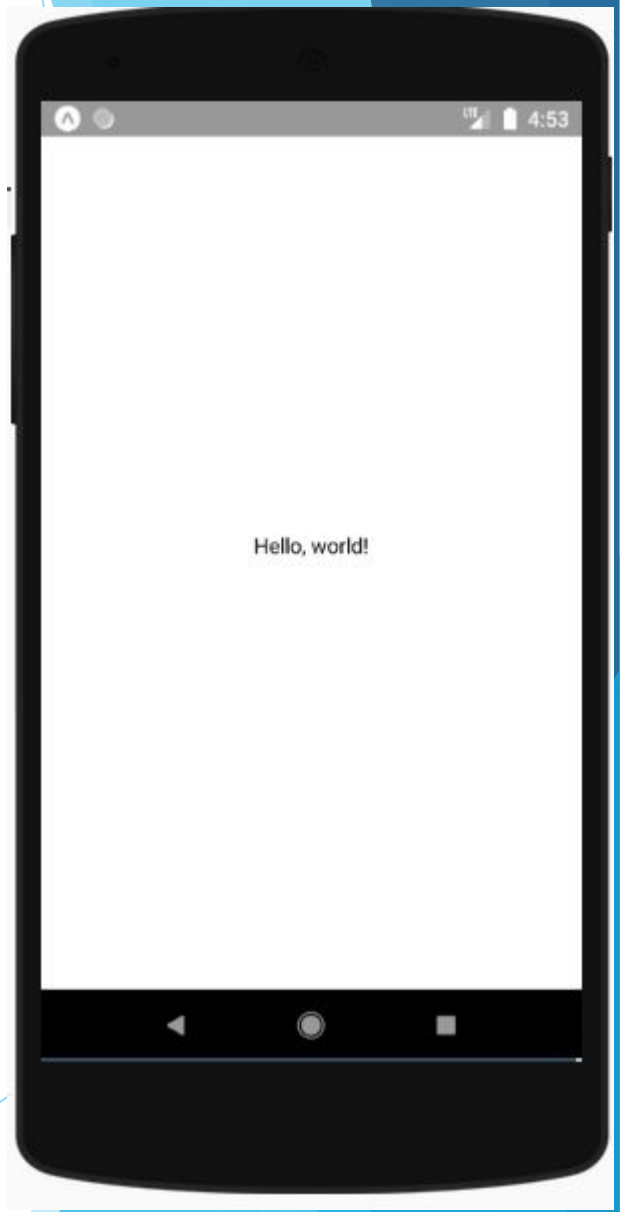
Лекция 10

React Native

- React Native похож на React, но в качестве строительных блоков он использует нативные компоненты вместо веб-компонентов.

- React.js – javascript библиотека для создания пользовательских интерфейсов.

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

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center",
                     alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```



Что тут происходит?

- Прежде всего, ES2015 (также известный как ES6) - это набор улучшений JavaScript, который теперь является частью официального стандарта, но еще не поддерживается всеми браузерами, поэтому часто он еще не используется в веб-разработке. React Native поставляется с поддержкой ES2015.

- Другая необычная вещь в этом примере кода

```
<View><Text>Hello, world!</Text></View>
```

- Это JSX - синтаксис для встраивания XML в JavaScript. Многие фреймворки используют специализированный язык шаблонов, который позволяет встраивать код в язык разметки. В React все наоборот. JSX позволяет вам писать свой язык разметки внутри кода. В Интернете он выглядит как HTML, за исключением того, что вместо таких вещей, как или , вы используете компоненты React. В этом случае <Text> является базовым компонентом, который отображает некоторый текст, а View похож на или .

Встраивание выражений в JSX

- Для встраивания переменных внутри JSX используются фигурные скобки

```
const name = 'Иван Иванович';
const element = <Text>Привет, {name}</Text>;
```

Компоненты

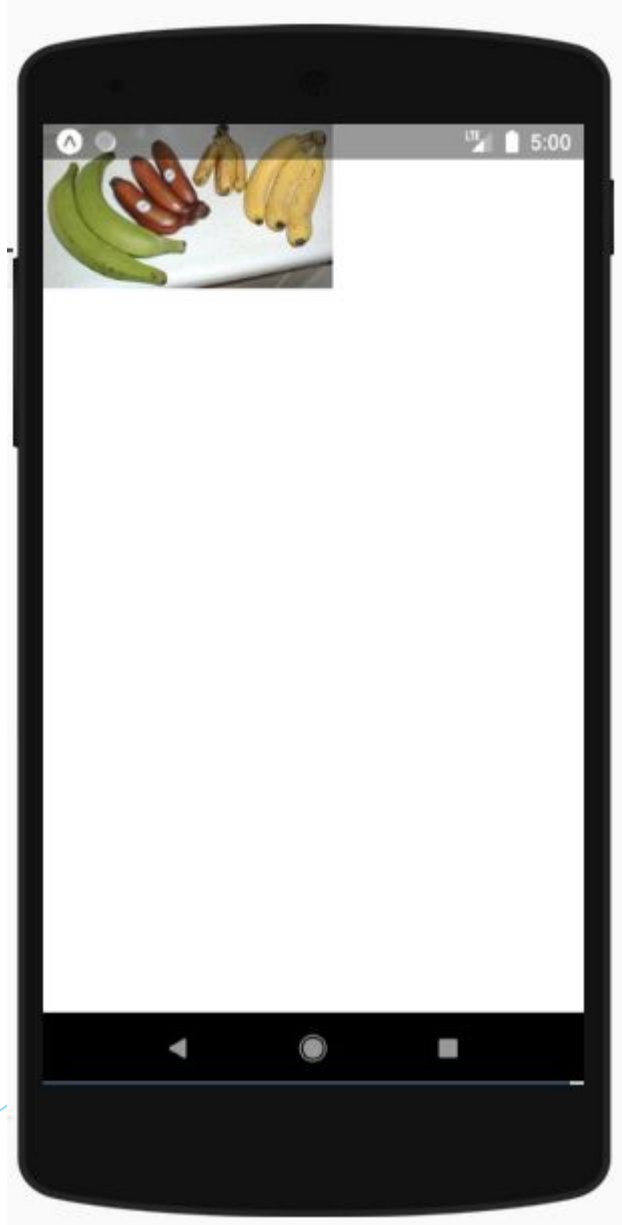
- Таким образом, этот код определяет в HelloWorldApp, новый Компонент. Когда вы создаете приложение React Native, вы будете создавать много компонентов. Все, что вы видите на экране, является своего рода компонентом. Компонент может быть довольно простым - единственное, что требуется, - это функция рендеринга (function render()), которая возвращает JSX для рендеринга.

Props (Properties)

- Большинство компонентов могут быть настроены при их создании с различными параметрами. Эти параметры создания называются props, сокращенно от properties (свойства).

- Например, одним из основных компонентов React Native является Image. Когда вы создаете изображение, вы можете использовать свойство с именем source, чтобы контролировать, какое изображение оно показывает.

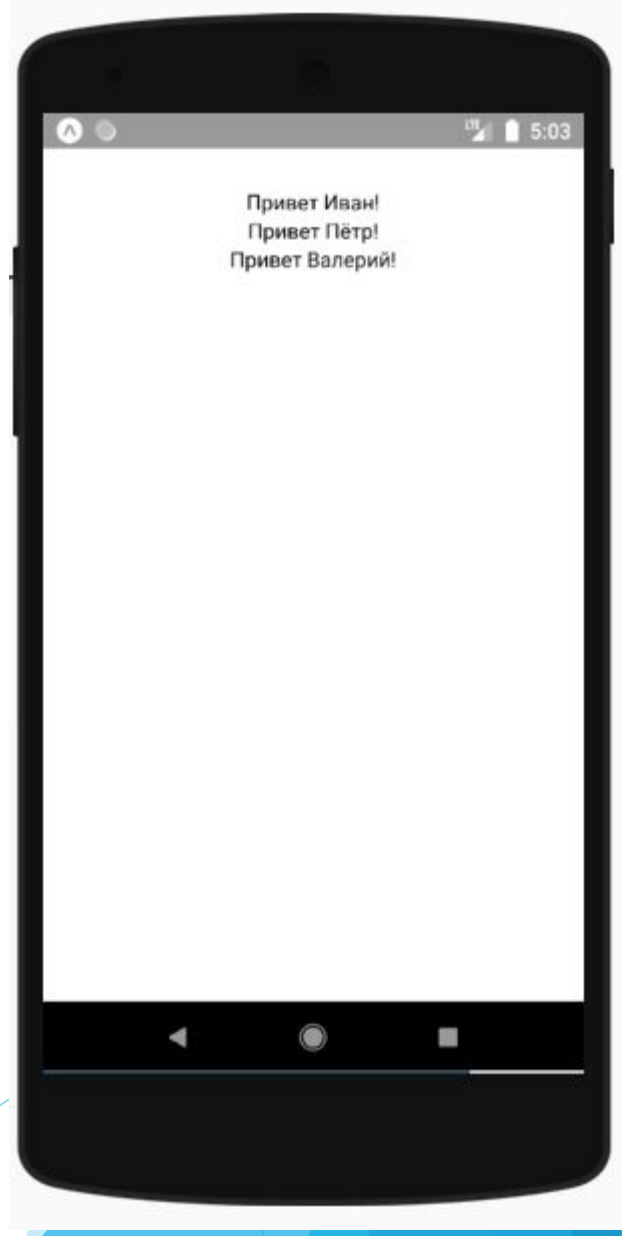
```
import React, { Component } from 'react';
import { Image } from 'react-native';

export default class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/
Bananavarieties.jpg'
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}
```



- Обратите внимание на фигурные скобки, окружающие {pic} - они вставляют переменную pic в JSX. Вы можете поместить любое выражение JavaScript в фигурные скобки в JSX.

- Ваши собственные компоненты также могут использовать свойства. Это позволяет вам создать один компонент, который используется в разных местах вашего приложения, с немного разными свойствами в каждом месте, ссылаясь на this.props в вашей функции рендеринга.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';
class Greeting extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Text>Привет {this.props.name}!</Text>
      </View>
    );
  }
}
export default class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', top: 50}}>
        <Greeting name='Иван' />
        <Greeting name='Пётр' />
        <Greeting name='Валерий' />
      </View>
    );
  }
}
```



- Использование свойства name позволяет нам настраивать компонент приветствия, поэтому мы можем повторно использовать этот компонент для каждого нашего приветствия. В этом примере также используется компонент приветствия в JSX, аналогичный основным компонентам. Возможность сделать это - то, что делает React таким крутым - если вы захотите, чтобы у вас был другой набор примитивов пользовательского интерфейса, вы можете изобретать новые.

- Другая новая вещь, происходящая здесь, это компонент View. Представление полезно в качестве контейнера для других компонентов, чтобы помочь контролировать стиль и макет.

- С помощью свойств и базовых компонентов Text, Image и View вы можете создавать разнообразные статические экраны.

Состояние (State)

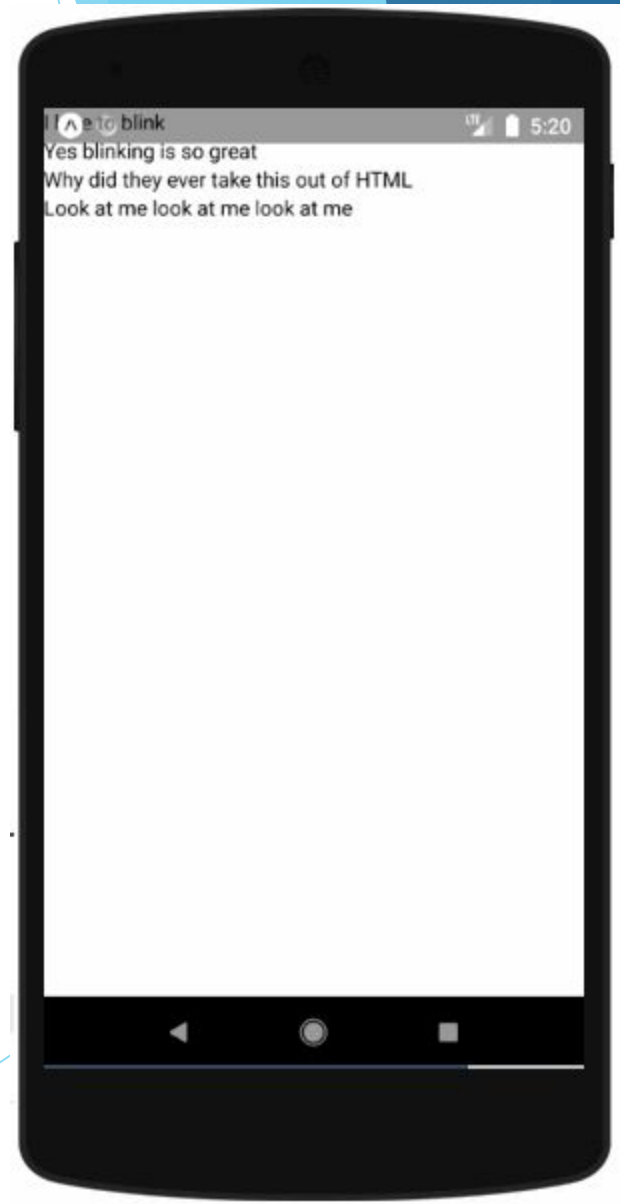
- Существует два типа данных, которые управляют компонентом: свойства и состояние. Свойства устанавливаются родителем и фиксируются на протяжении всего срока службы компонента. Для данных, которые будут меняться, мы должны использовать состояние.

- В общем, вы должны инициализировать состояние в конструкторе, а затем вызывать setState, когда вы хотите изменить его.

- Например, допустим, мы хотим сделать текст, который постоянно мигает. Сам текст устанавливается один раз, когда создается мигающий компонент, поэтому сам текст является свойством. «Включен ли текст в настоящее время или выключен» со временем изменяется, поэтому его следует сохранять в состоянии.

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

export default class BlinkApp extends Component {
    render() {
      return (
        <View>
          <Blink text='I love to blink' />
          <Blink text='Yes blinking is so great' />
          <Blink text='Why did they ever take this out
 of HTML' />
          <Blink text='Look at me look at me look at m
e' />
        </View>
      );
    }
  }
```

```
class Blink extends Component {
  componentDidMount(){
    // изменяем состояние каждую секунду
    setInterval(() => (
      this.setState(previousState => (
        { isShowingText: !previousState.isShowingText }
      ))
    ), 1000);
  }
  // объект состояния
  state = { isShowingText: true };
  render() {
    if (!this.state.isShowingText) {
      return null;
    }
    return (
      <Text>{this.props.text}</Text>
    );
  }
}
```

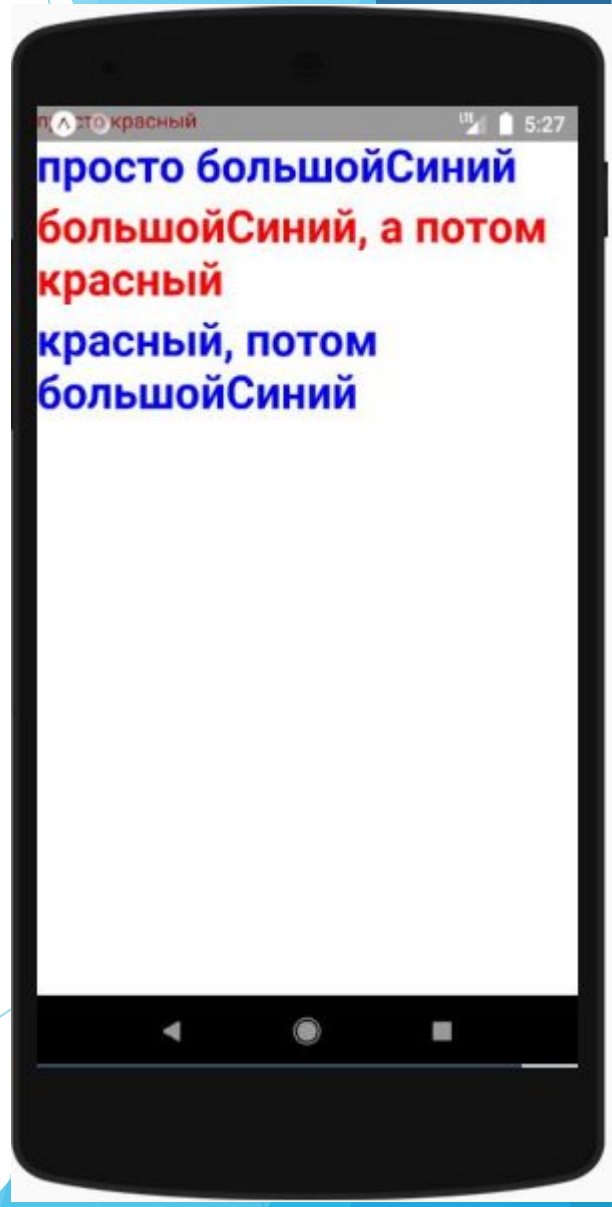


- В реальном приложении вы, вероятно, не будете устанавливать состояние с помощью таймера. Вы можете установить состояние, когда у вас есть новые данные с сервера или из пользовательского ввода. Вы также можете использовать контейнер состояний, такой как Redux или Mobx, для управления потоком данных. В этом случае вы будете использовать Redux или Mobx для изменения своего состояния, а не вызывать setState напрямую.
- Когда вызывается setState, BlinkApp повторно рендерит свой компонент. Вызывая setState внутри Timer, компонент будет рендериться каждый раз, когда Timer тикает.

Style (Стиль)

- С React Native вы разрабатываете свое приложение, используя JavaScript. Все основные компоненты поддерживают свойство style. Имена и значения стилей обычно совпадают с тем, как работает CSS в вебе, за исключением того, что имена пишутся с использованием верблюжьей нотации, например backgroundColor, а не background-color.
- Свойство стиля может быть простым объектом JavaScript. Вы также можете передать массив стилей - последний стиль в массиве имеет приоритет, поэтому вы можете использовать его для наследования стилей.
- По мере усложнения компонента часто становится проще использовать StyleSheet.create для определения нескольких стилей в одном месте.

```
import React, { Component } from 'react';
import { StyleSheet, Text, View } from 'react-native';
const styles = StyleSheet.create({
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});
export default class LotsOfStyles extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>просто красный</Text>
        <Text style={styles.bigBlue}>просто большойСиний</Text>
        <Text style={[styles.bigBlue, styles.red]}>большойСиний, а потом красный</Text>
        <Text style={[styles.red, styles.bigBlue]}>красный, потом большойСиний</Text>
      </View>
    );
  }
```



- Один из распространенных шаблонов - заставить ваш компонент принимать свойство style, которое, в свою очередь, используется для стилизации подкомпонентов. Вы можете использовать это, чтобы сделать стили "каскадными", как в CSS.

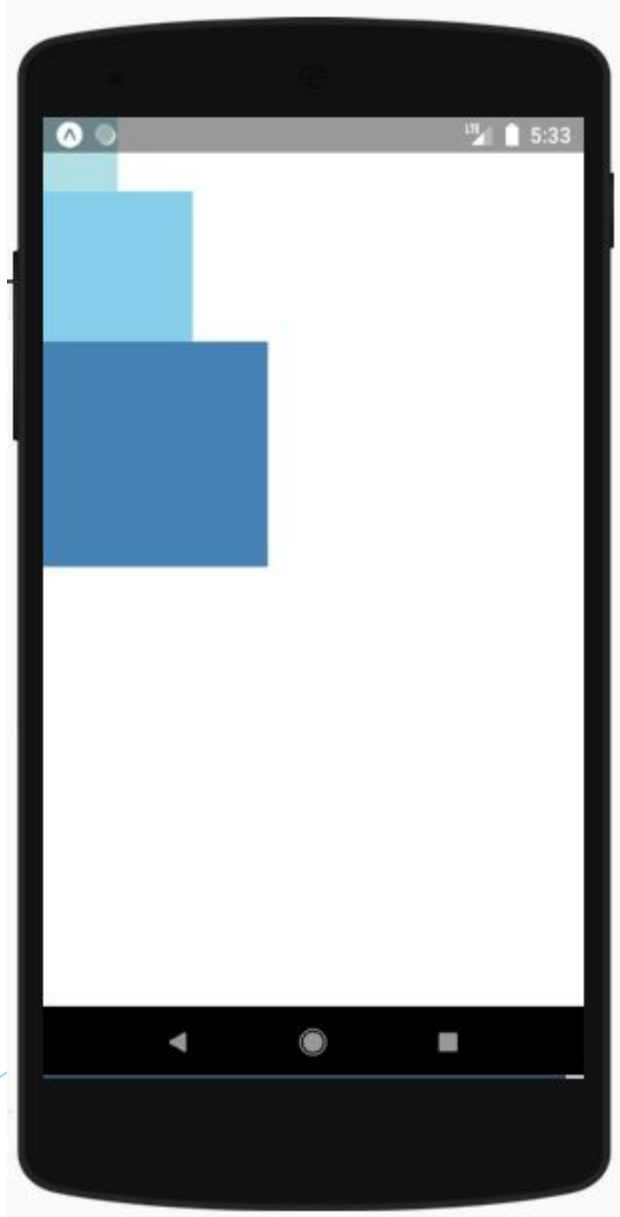
Высота и ширина

- Высота и ширина компонента определяют его размер на экране.

Фиксированные размеры

- Самый простой способ установить размеры компонента - добавить фиксированную ширину (width) и высоту (height) к стилю. Все измерения в React Native не имеют единиц измерения и представляют собой независимые от плотности пиксели.
- Установка размеров таким способом используется для компонентов, которые должны всегда отображаться с одинаковым размером, независимо от размеров экрана.

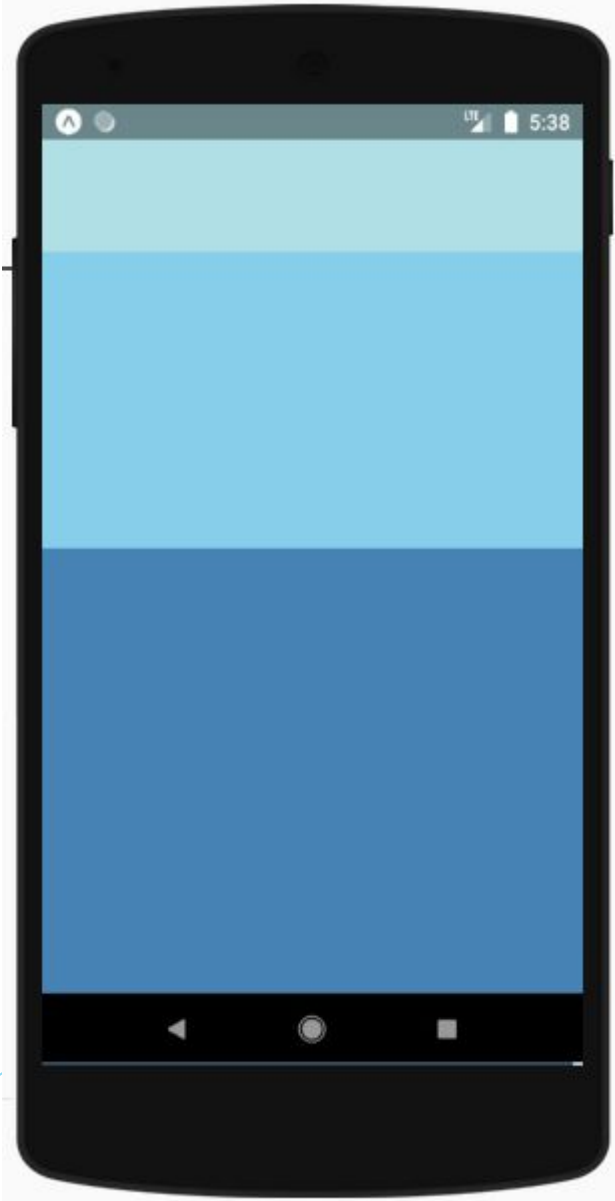
```
import React, { Component } from 'react';
import { View } from 'react-native';

export default class FixedDimensionsBasics extends Component {
  render() {
    return (
      <View>
        <View style={{width: 50, height: 50, backgroundColor:
'powderblue'}} />
        <View style={{width: 100, height: 100, backgroundColor:
'skyblue'}} />
        <View style={{width: 150, height: 150, backgroundColor:
'steelblue'}} />
      </View>
    );
  }
}
```



Гибкие размеры

- Используйте flex в стиле компонента, чтобы компонент динамически расширялся и уменьшался в зависимости от доступного пространства. Обычно используется flex: 1, который указывает компоненту заполнить все доступное пространство, равномерно распределенное между другими компонентами с тем же родителем. Чем больше значение flex, тем выше соотношение пространства, которое займет компонент по сравнению с его братьями и сестрами.
- Компонент может расширяться, чтобы заполнить доступное пространство, если его родительский элемент имеет размеры больше 0. Если родительский элемент не имеет фиксированной ширины, высоты или гибкости, родительский элемент будет иметь размеры 0, а дочерние элементы flex не будут видны.

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

export default class FlexDimensionsBasics extends Component {
  render() {
    return (
      <View style={{flex: 1}}>
        <View style={{flex: 1, backgroundColor: 'powderblue'}} />
        <View style={{flex: 2, backgroundColor: 'skyblue'}} />
        <View style={{flex: 3, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```

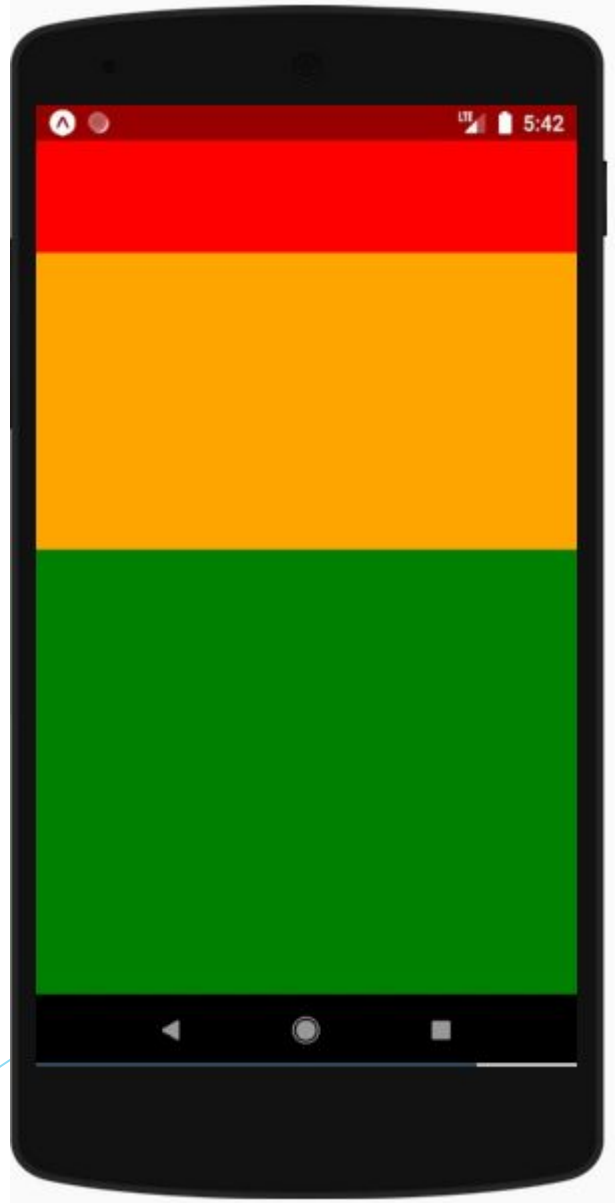


Layout с Flexbox

- Компонент может указать макет своих дочерних элементов, используя алгоритм flexbox. Flexbox разработан для обеспечения согласованного расположения на экранах разных размеров.
- Обычно вы используете комбинацию flexDirection, alignItems и justifyContent для достижения правильного макета.
- Flexbox работает в React Native так же, как и в CSS в вебе, за некоторыми исключениями. Значения по умолчанию различаются: во FlexDirection по умолчанию используется столбец, а не строка, а параметр flex поддерживает только одно число.

Flex

- flex определит, как ваши элементы будут «заполнять» доступное пространство вдоль вашей главной оси. Пространство будет разделено в соответствии со свойством flex каждого элемента.
- В примере все представления красного, желтого и зеленого являются дочерними в представлении контейнера, для которого установлено значение flex: 1. Красный вид использует flex: 1, желтый - flex: 2, а зеленый - flex: 3. 1 + 2 + 3 = 6, что означает, что красный вид получит 1/6 пространства, желтый 2/6 пространства и зеленый 3/6 пространства.



Flex Direction

- flexDirection управляет направлением расположения дочерних узлов. Это также называется главной осью. Ось креста - это ось, перпендикулярная главной оси, или ось, на которой расположены линии обертывания.

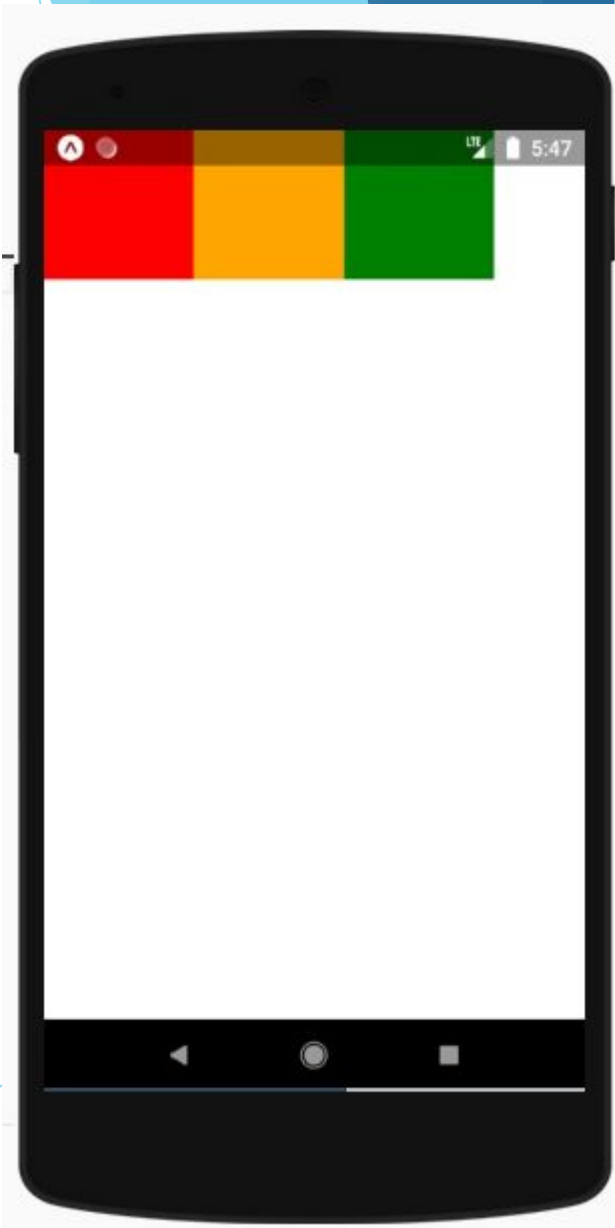
- **row** – выравнивает дочерние элементы слева направо. Если перенос включен, следующая строка начнется под первым элементом слева от контейнера.
- **column** (значение по умолчанию) - выровнять дочерние элементы сверху вниз. Если перенос включен, следующая строка будет начинаться слева от первого элемента в верхней части контейнера.

Flex Direction

- **row-reverse** - выравнивает дочерние элементы справа налево. Если перенос включен, следующая строка начнется под первым элементом справа от контейнера.

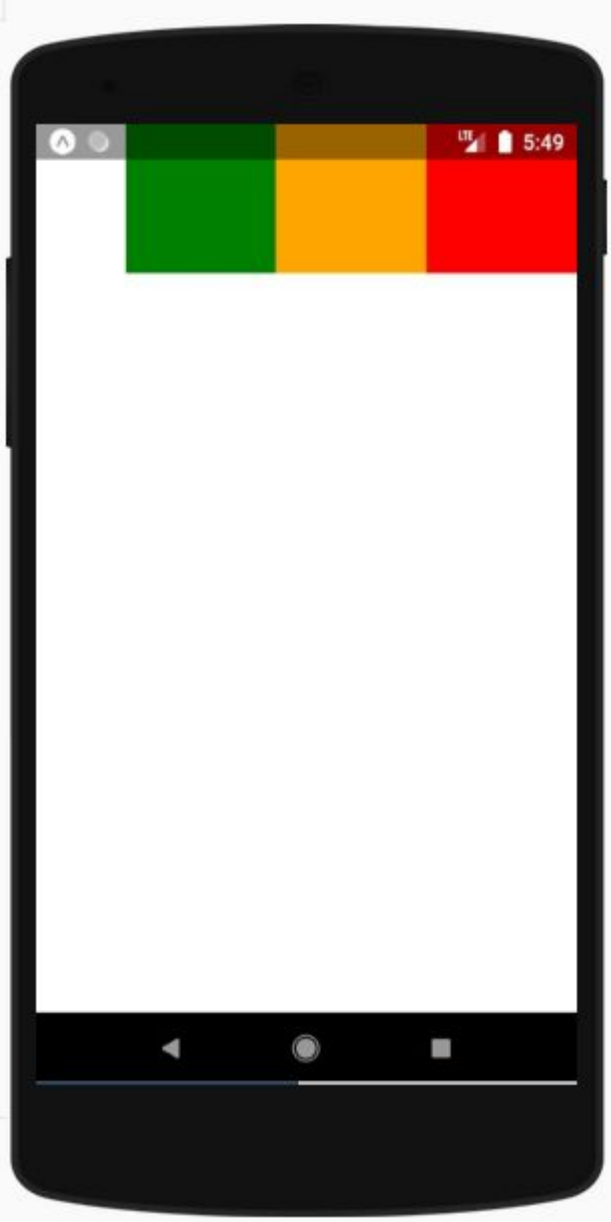
- **column-reverse** - выравнивает дочерние элементы снизу вверх. Если перенос включен, следующая строка будет начинаться слева от первого элемента в нижней части контейнера.

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

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      <View style={{flex: 1, flexDirection: 'row'}}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
};
```



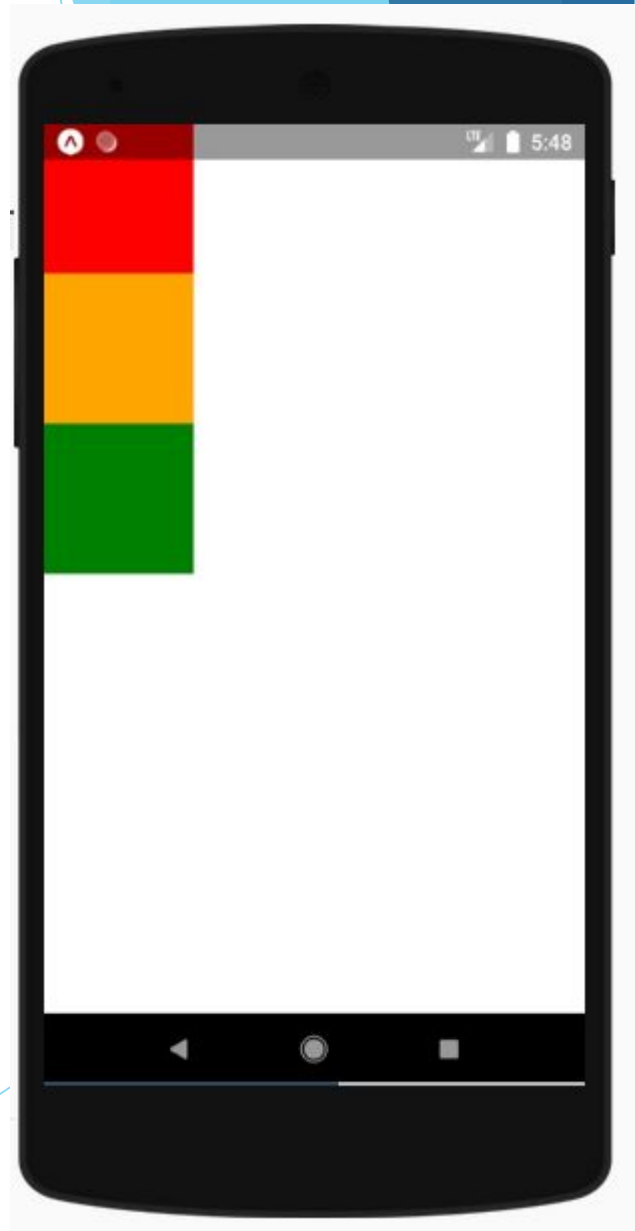
flexDirection: row-reverse

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

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      <View style={{flex: 1, flexDirection: 'row-reverse'}}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
};
```



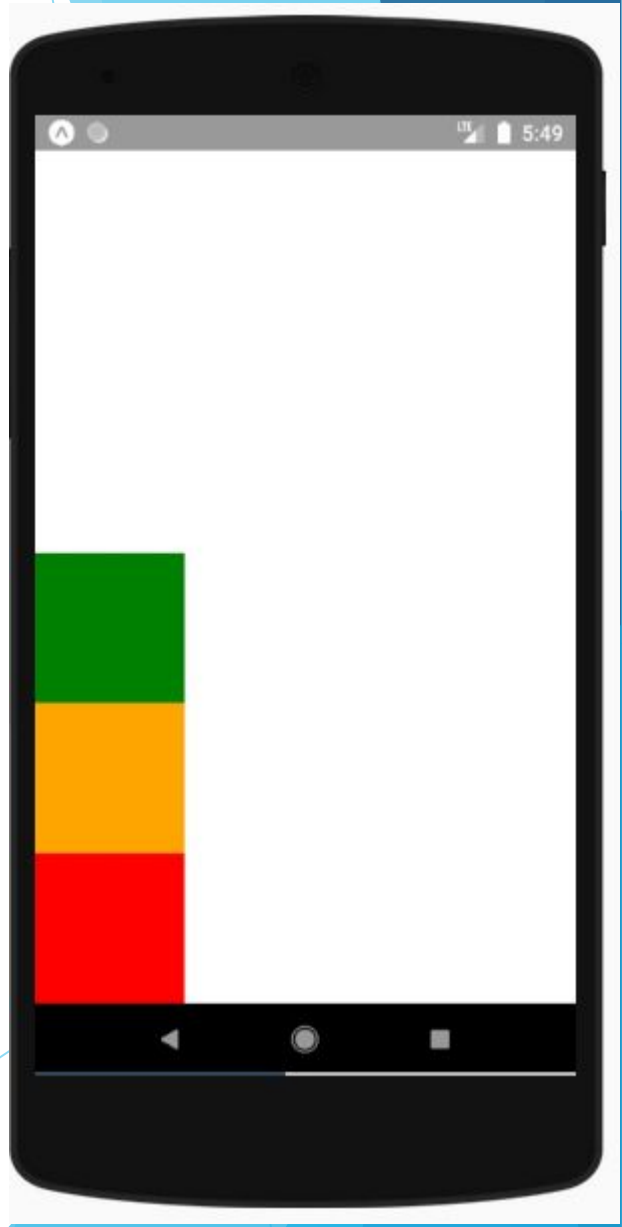
flexDirection: column

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

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      <View style={{flex: 1, flexDirection: 'column'}}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
};
```



flexDirection: column-reverse

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

export default class FlexDirectionBasics extends Component {
  render() {
    return (
      <View style={{flex: 1, flexDirection: 'column-reverse'}}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
};
```



Layout Direction

- Направление макета указывает направление, в котором должны быть размещены дочерние элементы и текст в иерархии. Направление расположения также влияет на начало и конец ребра. По умолчанию React Native располагается в направлении расположения LTR. В этом режиме начало относится к левому, а конец относится к правому.

- LTR (значение по умолчанию) - текст и дочерние элементы располагаются слева направо. Применяемые поля (marge) и отступы (padding) начала элемента применяются с левой стороны.

- RTL - текст и дочерние элементы располагаются справа налево. Поля (marge) и отступы (padding), примененный к началу элемента, применяются с правой стороны.

Justify Content

- justifyContent описывает, как выровнять дочерние элементы в пределах главной оси их контейнера. Например, это свойство можно использовать для центрирования дочернего элемента по горизонтали в контейнере с параметром flexDirection, установленным в строку, или по вертикали в контейнере с параметром flexDirection, установленным в столбец.

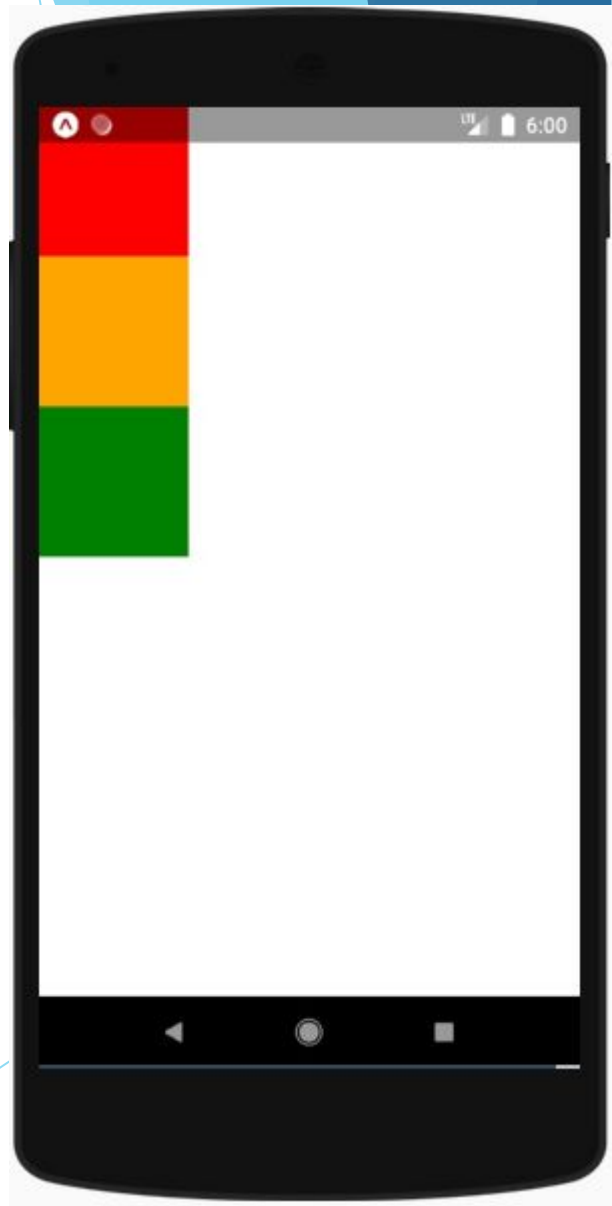
- **flex-start** (значение по умолчанию) Выровнять дочерние элементы контейнера по началу главной оси контейнера.
- **flex-end** Выровнять дочерние элементы контейнера по концу главной оси контейнера.
- **center** Выровнять дочерние элементы контейнера по центру главной оси контейнера.
- **space-between** Равномерное пространство дочерних элементов по главной оси контейнера, распределяя оставшееся пространство между дочерними элементами.

- **space-around** – равномерно распределить дочерние элементы внутри контейнера, а затем распределить свободное пространство между ними. По сравнению с пробелом между использованием пробела будет распределено пространство между началом первого и концом последнего потомка.

- **space-evenly** - равномерно распределить внутри контейнера выравнивания вдоль главной оси. Интервал между каждой парой смежных элементов, краем основного начала и первым элементом, краем основного конца и последним элементом абсолютно одинаков.

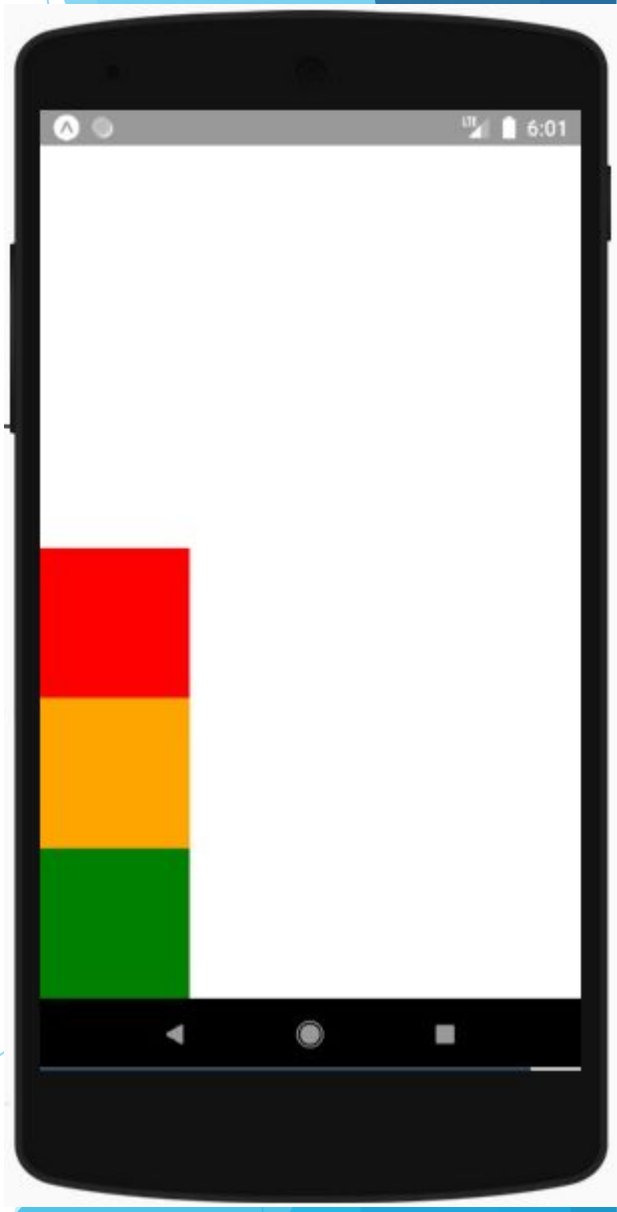
flex-start

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'flex-start',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



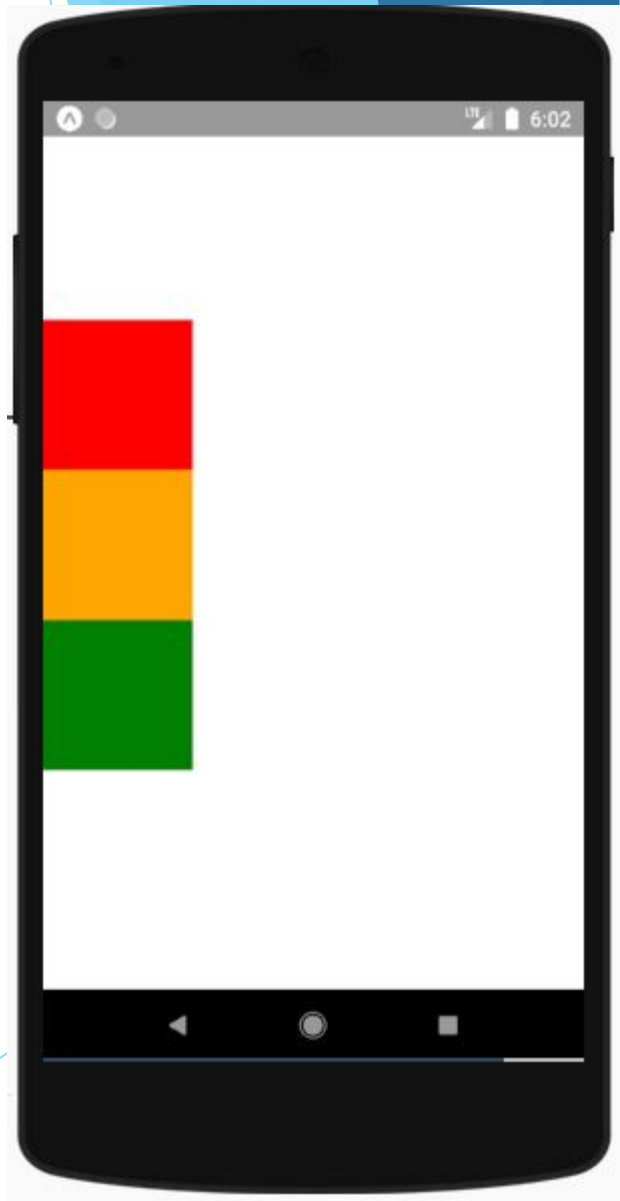
flex-end

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'flex-end',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



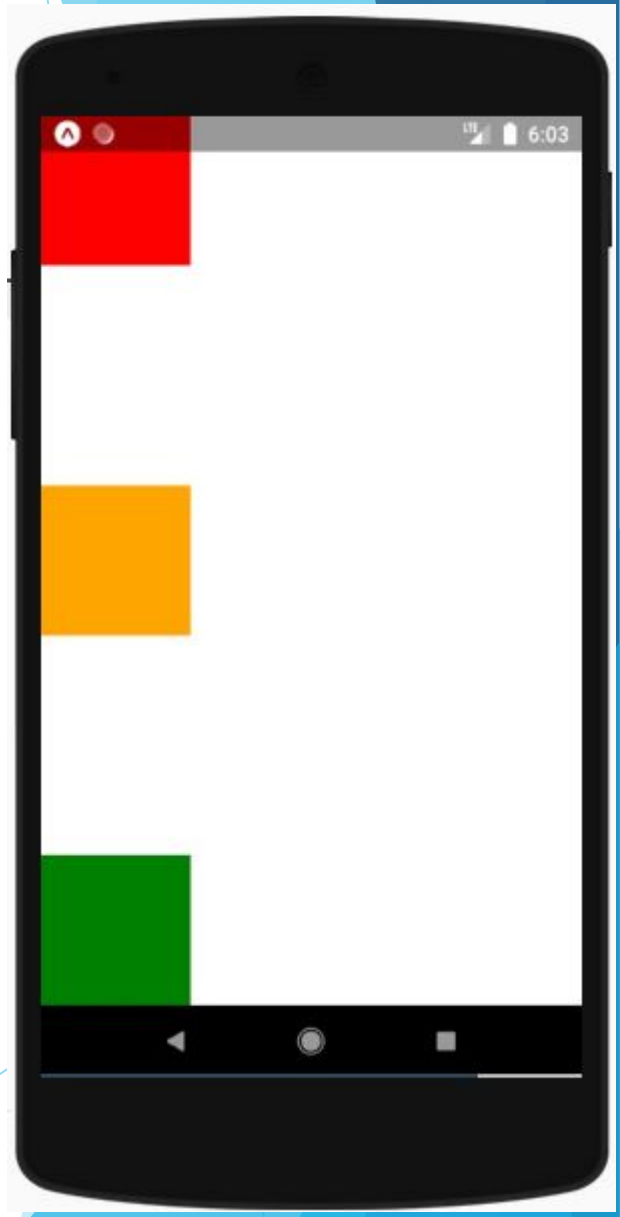
center

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



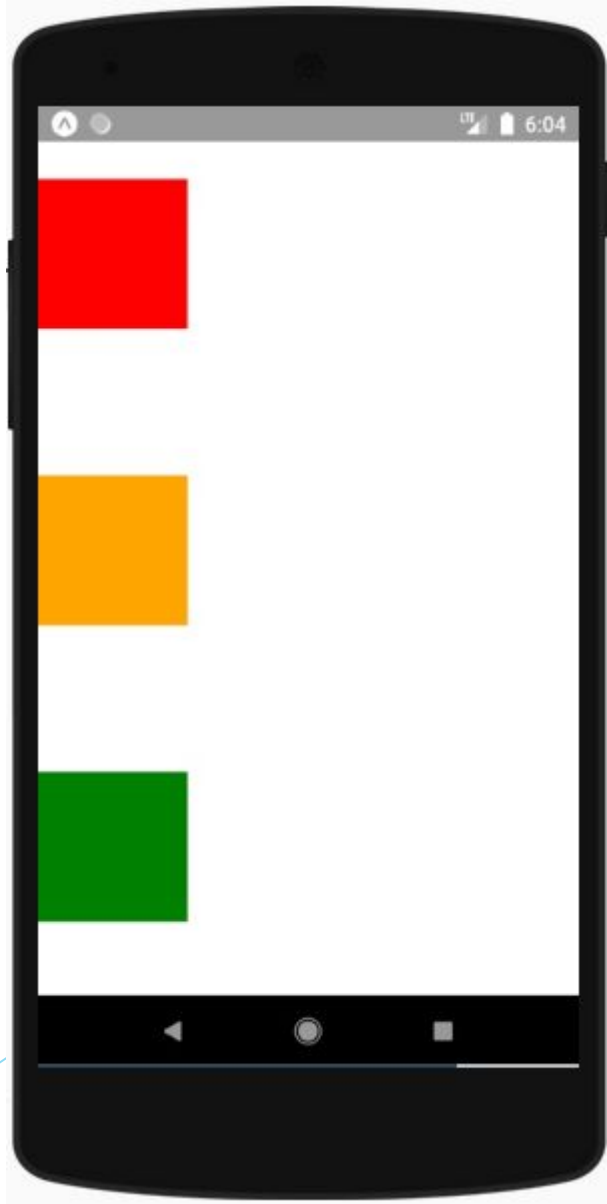
space-between

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'space-between',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



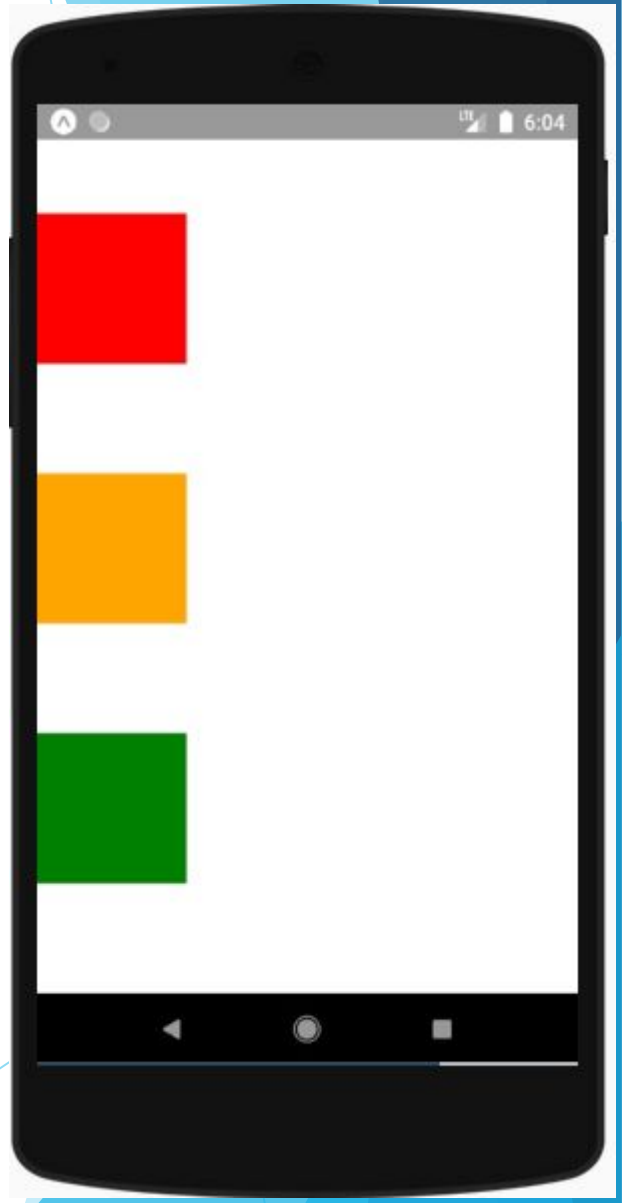
space-around

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'space-around',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```

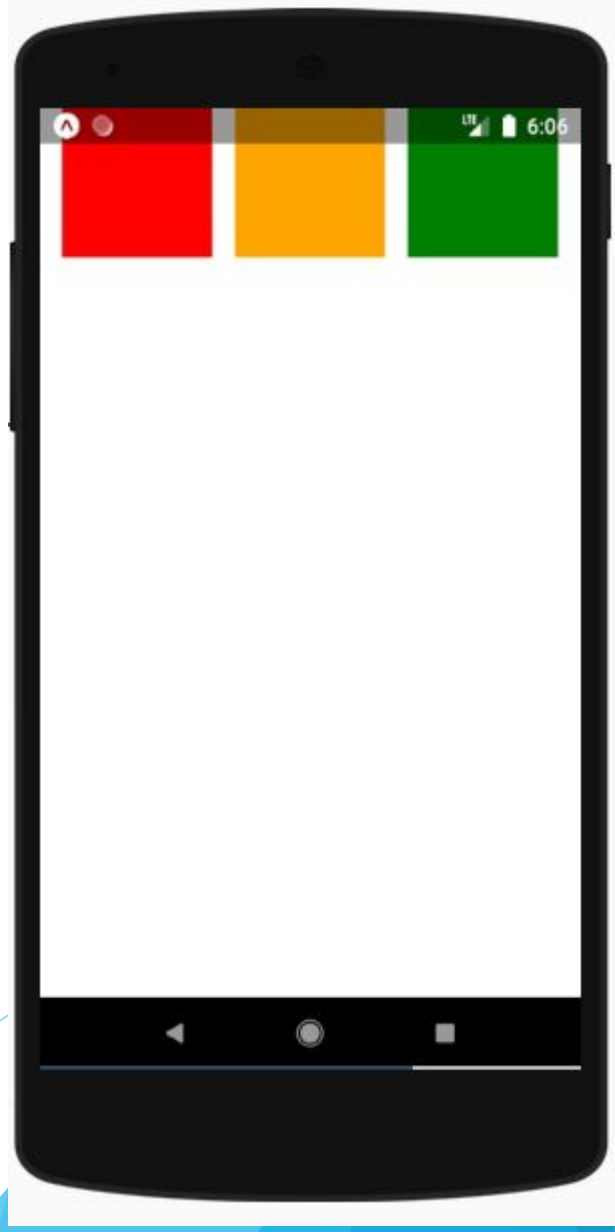


space-evenly

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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'space-evenly',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



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

export default class JustifyContentBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'row',
        justifyContent: 'space-evenly',
      }}>
        <View style={{width: 100, height: 100, backgroundColor: 'red'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'orange'}} />
        <View style={{width: 100, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



Align Items

- alignItems описывает, как выровнять дочерние элементы вдоль поперечной оси их контейнера. Выравнивание элементов очень похоже на justifyContent, но вместо применения к главной оси, alignItems применяется к поперечной оси.

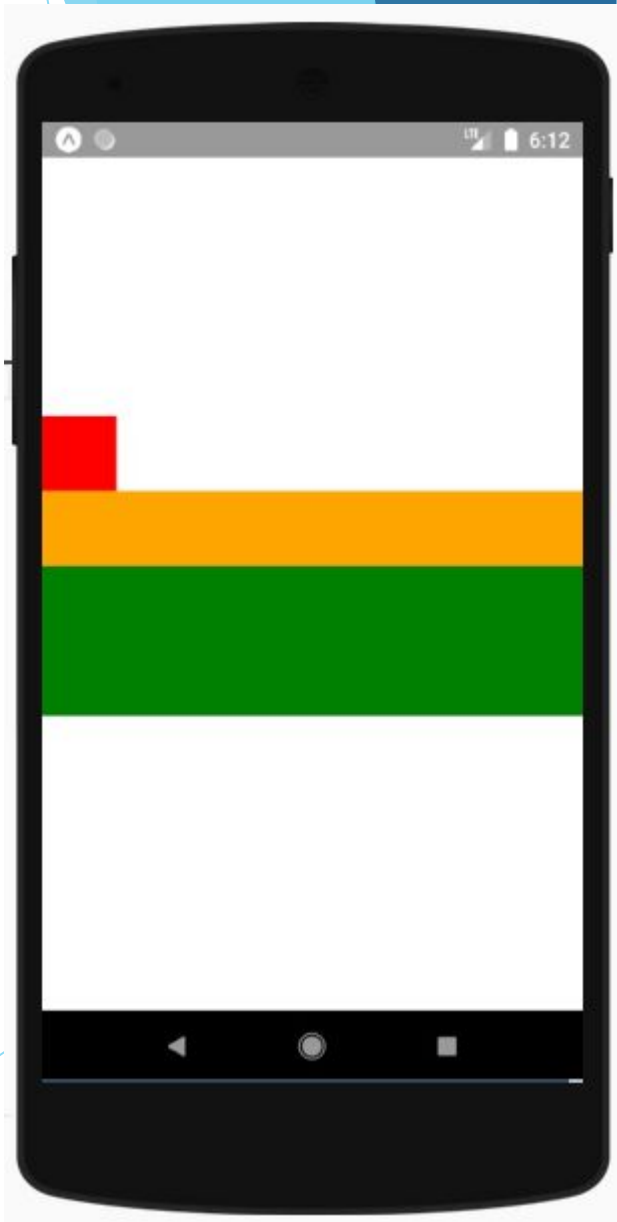
- **stretch** (значение по умолчанию) Растянуть дочерние элементы контейнера в соответствии с высотой поперечной оси контейнера (Чтобы растяжение имело эффект, дочерние элементы не должны иметь фиксированный размер вдоль вторичной оси).

Align Items

- **flex-start** Выровнять дочерние элементы контейнера по началу поперечной оси контейнера.
- **flex-end** Выровнять дочерние элементы контейнера по концу поперечной оси контейнера.
- **center** Выровнять дочерние элементы контейнера по центру поперечной оси контейнера.
- **baseline** Выровнять дочерние элементы контейнера по общей базовой линии. Отдельные дочерние элементы могут быть установлены в качестве базового уровня для своих родителей.

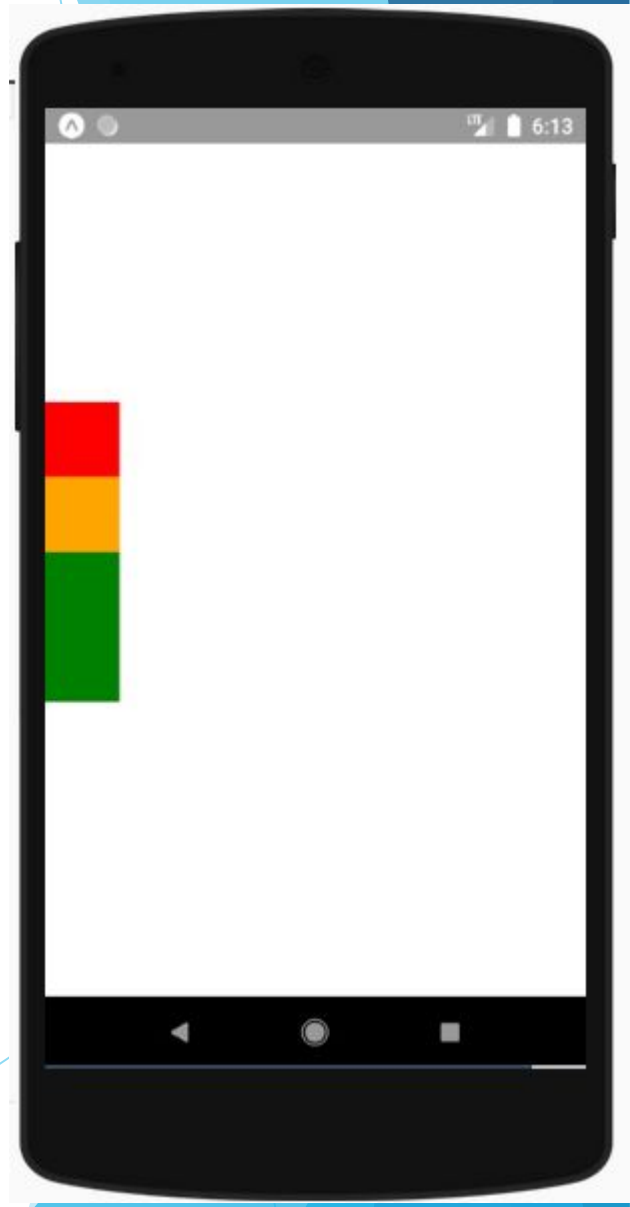
stretch

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

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'stretch',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'red'}} />
        <View style={{height: 50, backgroundColor: 'orange'}} />
        <View style={{height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```

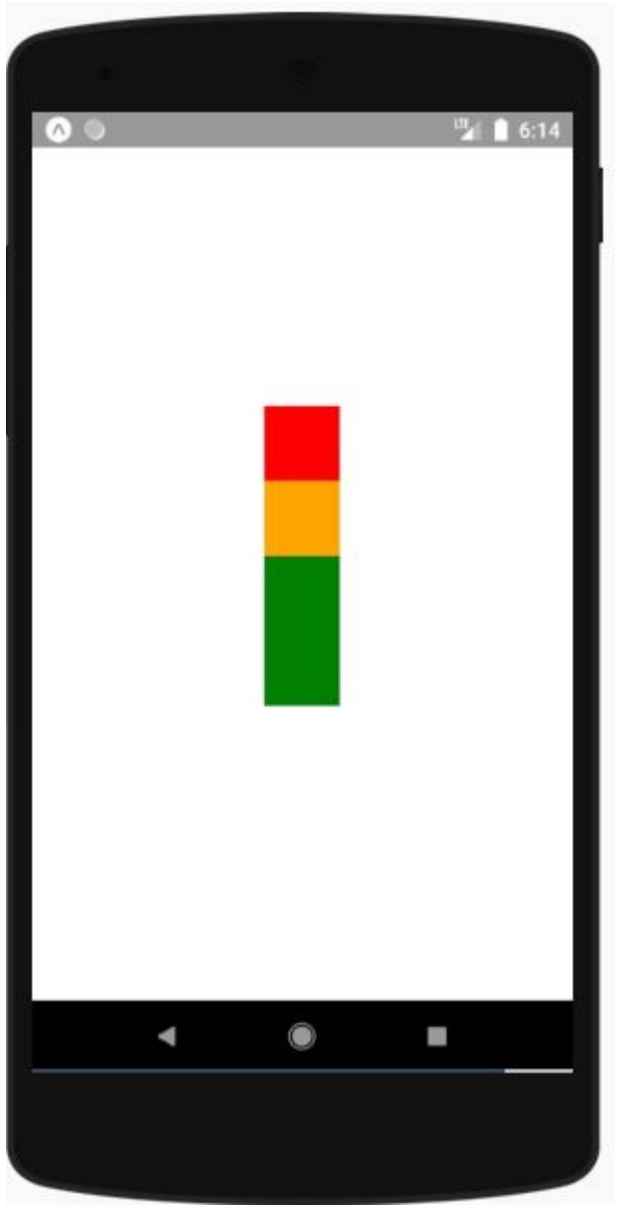


flex-start

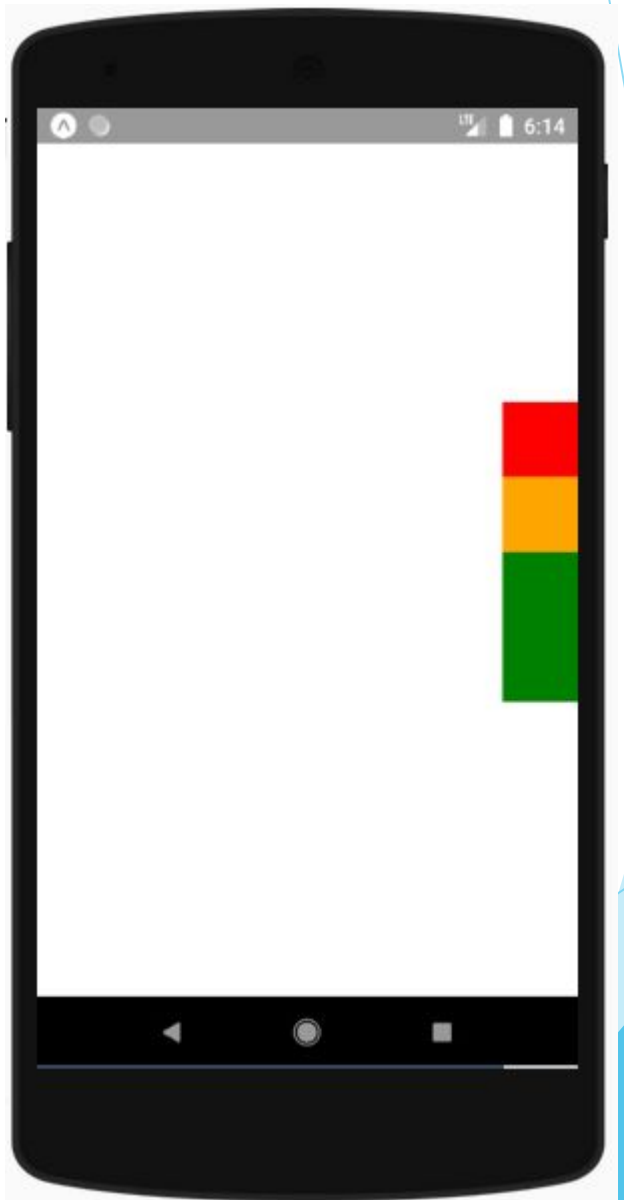
```
import React, { Component } from 'react';
import { View } from 'react-native';

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'flex-start',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'red'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'orange'}} />
        <View style={{width: 50, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



center



flex-end

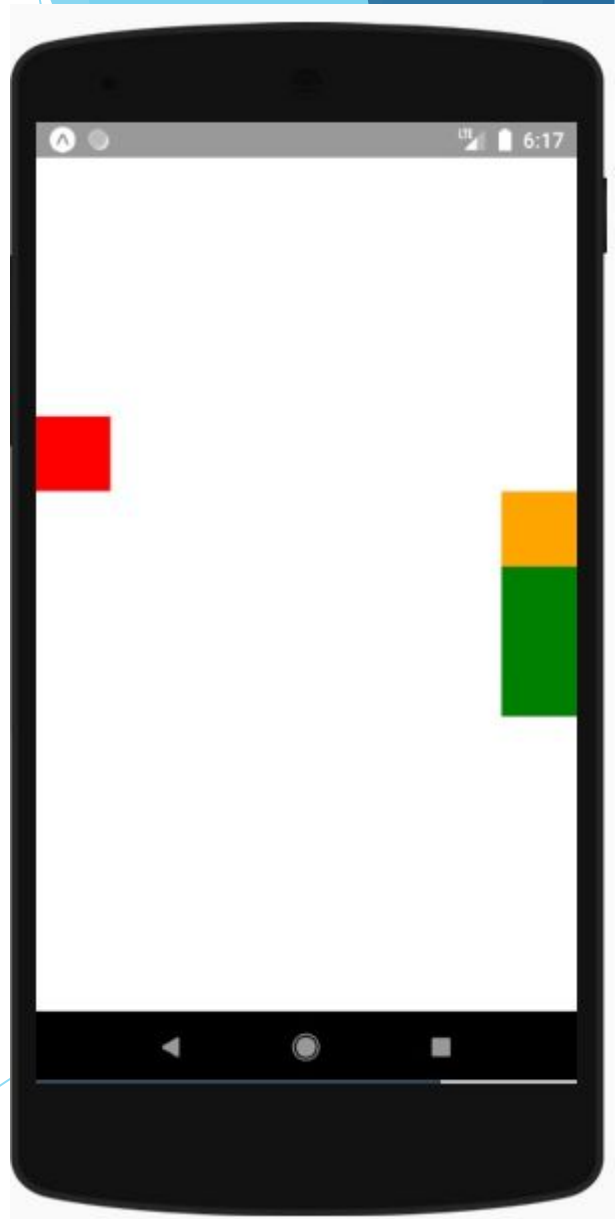


Align Self

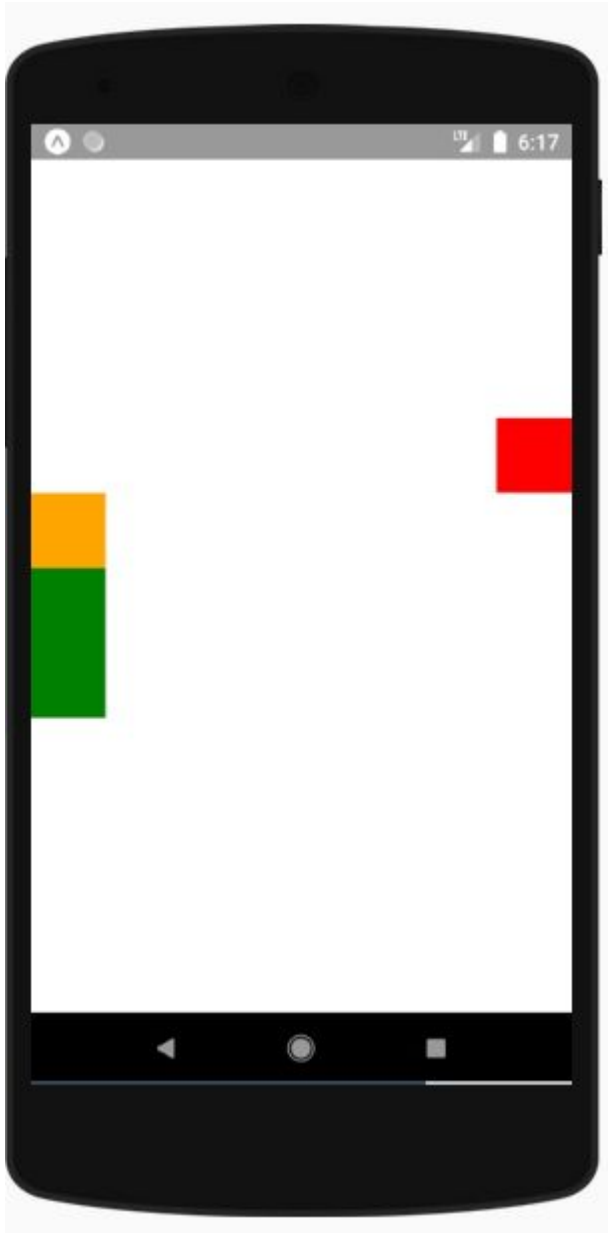
- Свойство alignSelf имеет те же параметры и эффект, что и alignItems, но вместо того, чтобы воздействовать на дочерние элементы в контейнере, вы можете применить это свойство к одному дочернему элементу, чтобы изменить его выравнивание в пределах родительского элемента. alignSelf переопределяет любой параметр, установленный родителем, с alignItems.

flex-start

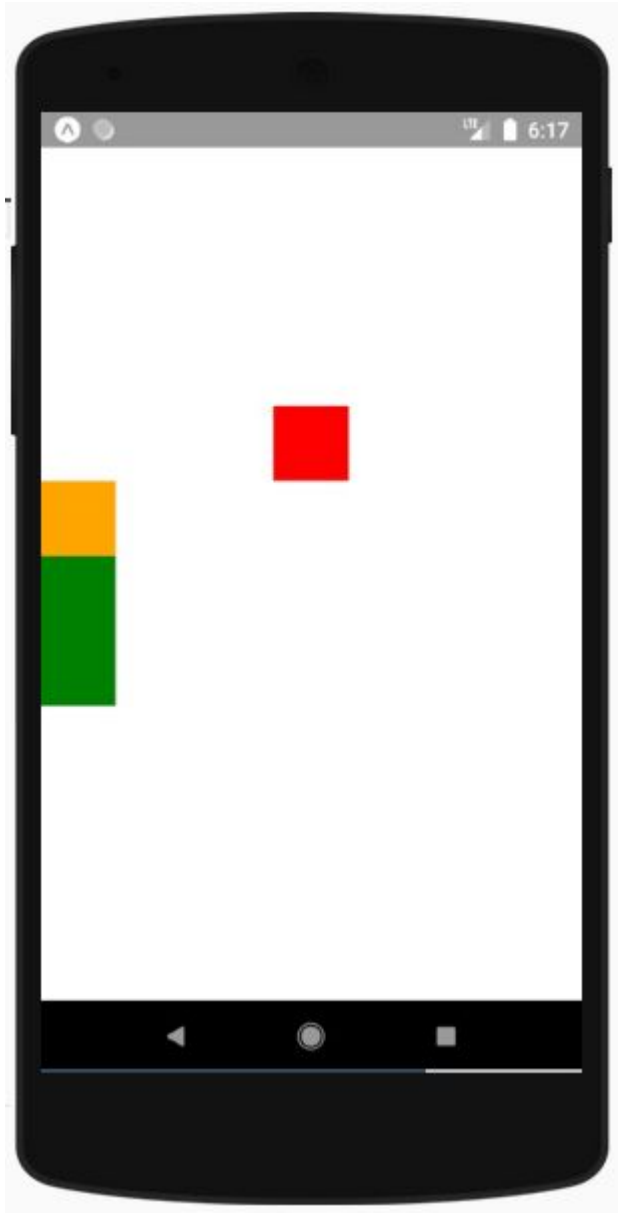
```
import React, { Component } from 'react';
import { View } from 'react-native';

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'flex-end',
      }}>
        <View style={{width: 50, height: 50, alignSelf: 'flex-start', backgroundColor: 'red'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'orange'}} />
        <View style={{width: 50, height: 100, backgroundColor: 'green'}} />
      </View>
    );
  }
}
```



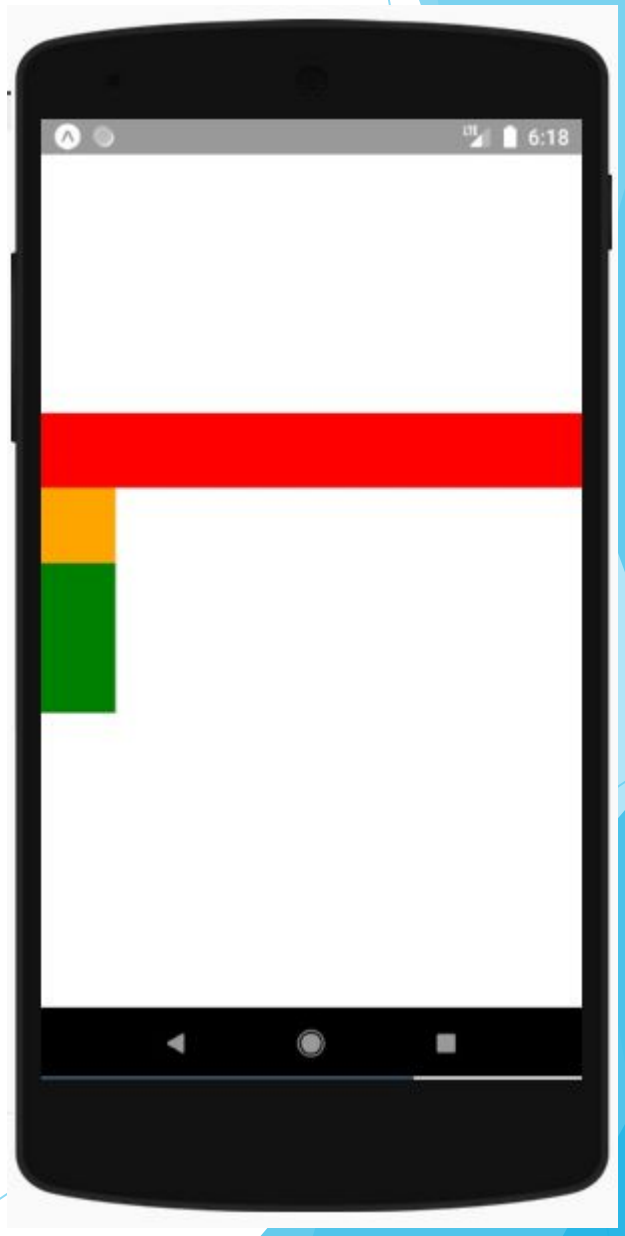
flex-end



center



stretch



Align Content

- alignContent определяет распределение линий вдоль поперечной оси. Это действует только тогда, когда элементы переносятся в несколько строк с помощью flexWrap.

- **flex-start** - (значение по умолчанию) выровнять перенос по началу поперечной оси контейнера.

- **flex-end** – выровнять перенос по концу поперечной оси контейнера.

- **stretch** – растянуть перенос, чтобы соответствовать высоте поперечной оси контейнера.

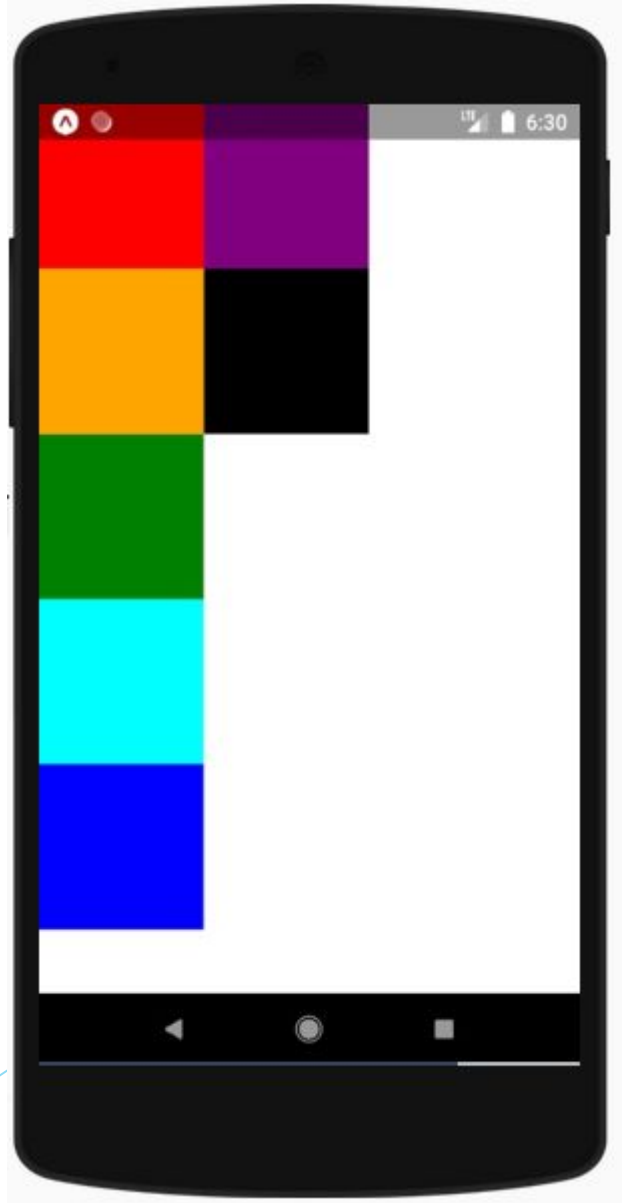
- **center** – выровнять перенос по центру поперечной оси контейнера.

- **space-between** - равномерно разделенные пробелами строки по главной оси контейнера, распределяющие оставшееся пространство между линиями.

- **space-around** – равномерный перенос, обернутый линиями вдоль главной оси контейнера, распределяя оставшееся пространство вокруг линий. По сравнению с пробелом между использованием пробела вокруг пространство будет распределено в начале первых строк и в конце последней строки.

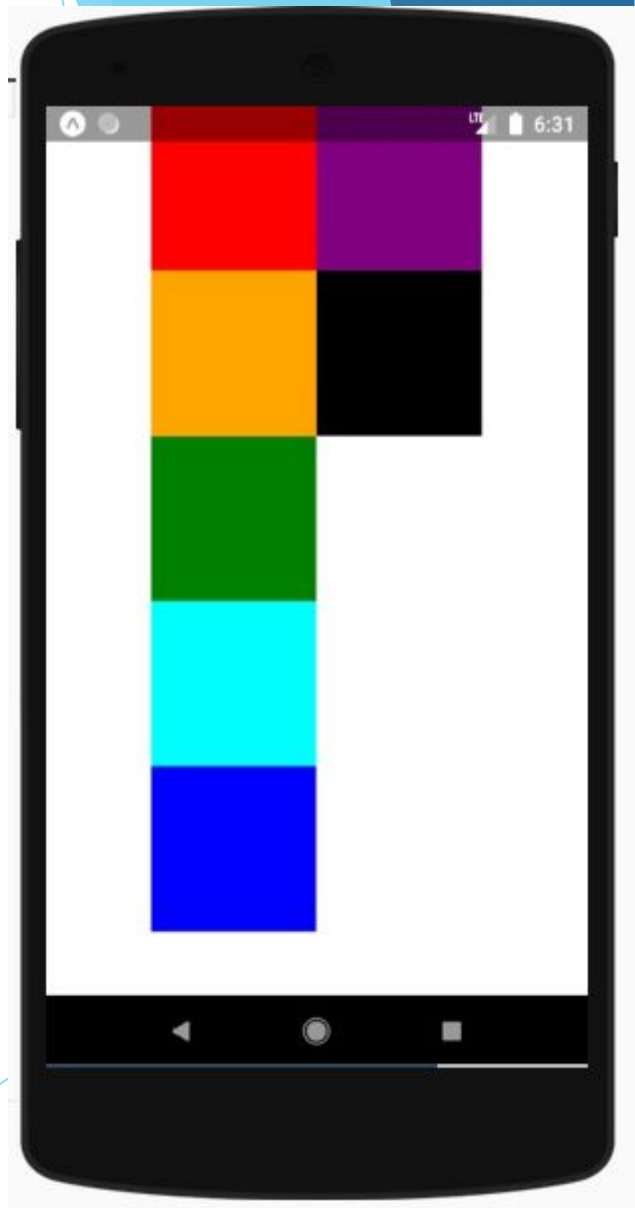
flex-start

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

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        alignContent: 'flex-start',
        flexWrap: "wrap"
      }}>
        <View style={{width: 110, height: 110, backgroundColor: 'red'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'orange'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'green'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'cyan'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'blue'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'purple'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'black'}} />
      </View>
    );
  }
}
```

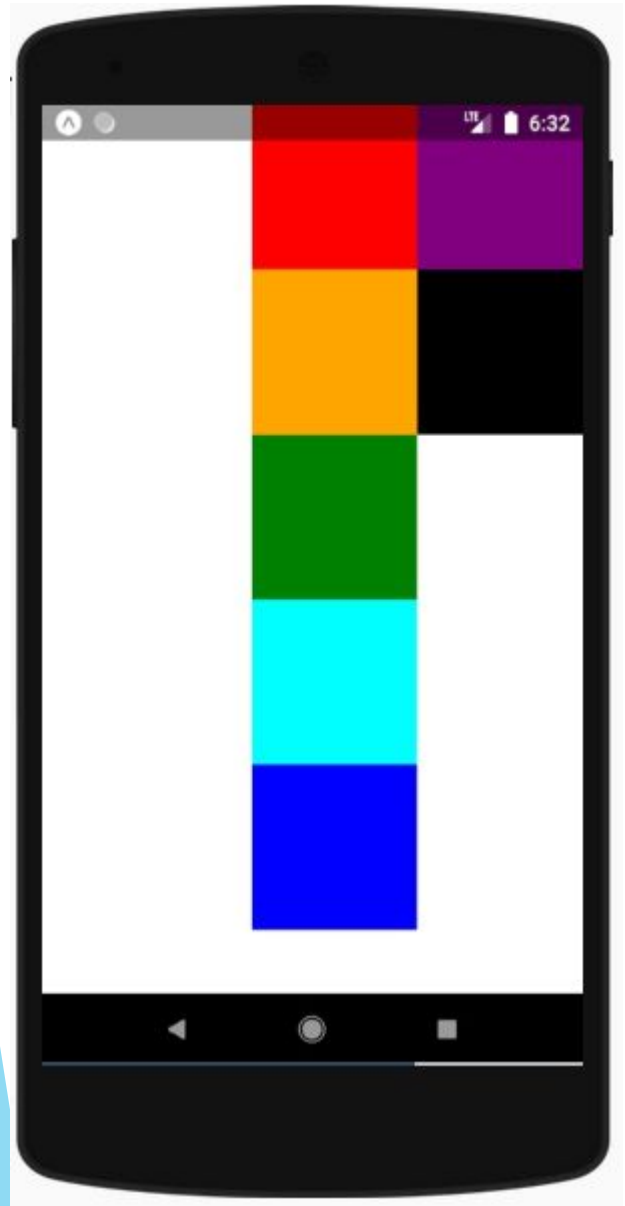


center

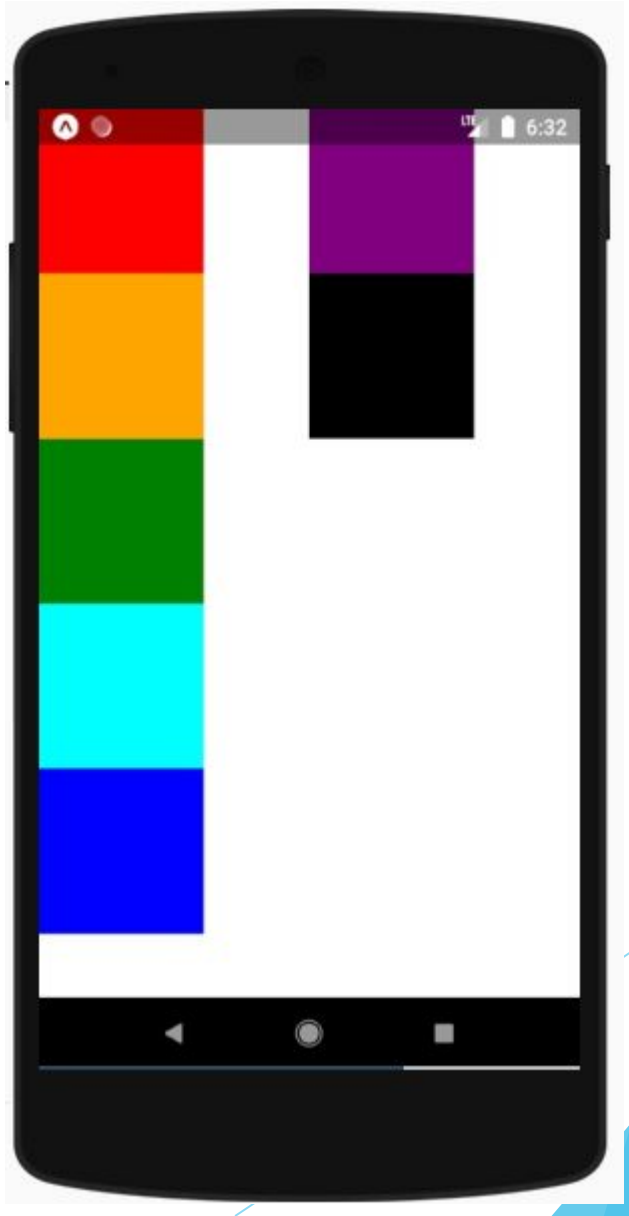
```
import React, { Component } from 'react';
import { View } from 'react-native';

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View style={{
        flex: 1,
        flexDirection: 'column',
        alignContent: 'center',
        flexWrap: "wrap"
      }}>
        <View style={{width: 110, height: 110, backgroundColor: 'red'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'orange'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'green'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'cyan'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'blue'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'purple'}} />
        <View style={{width: 110, height: 110, backgroundColor: 'black'}} />
      </View>
    );
  }
}
```



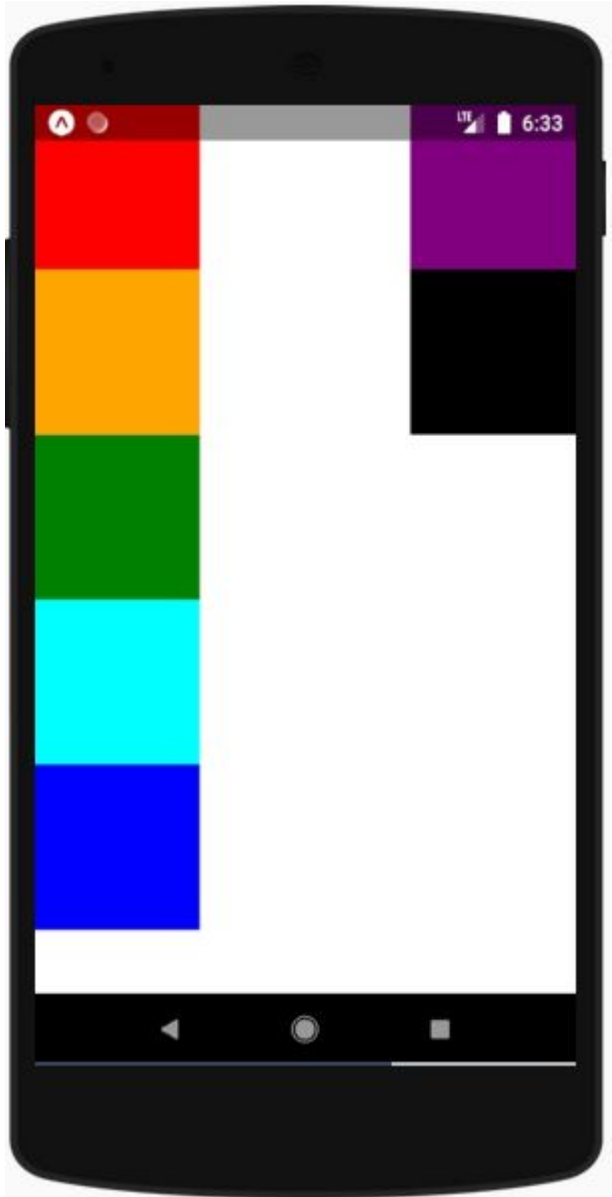
flex-end



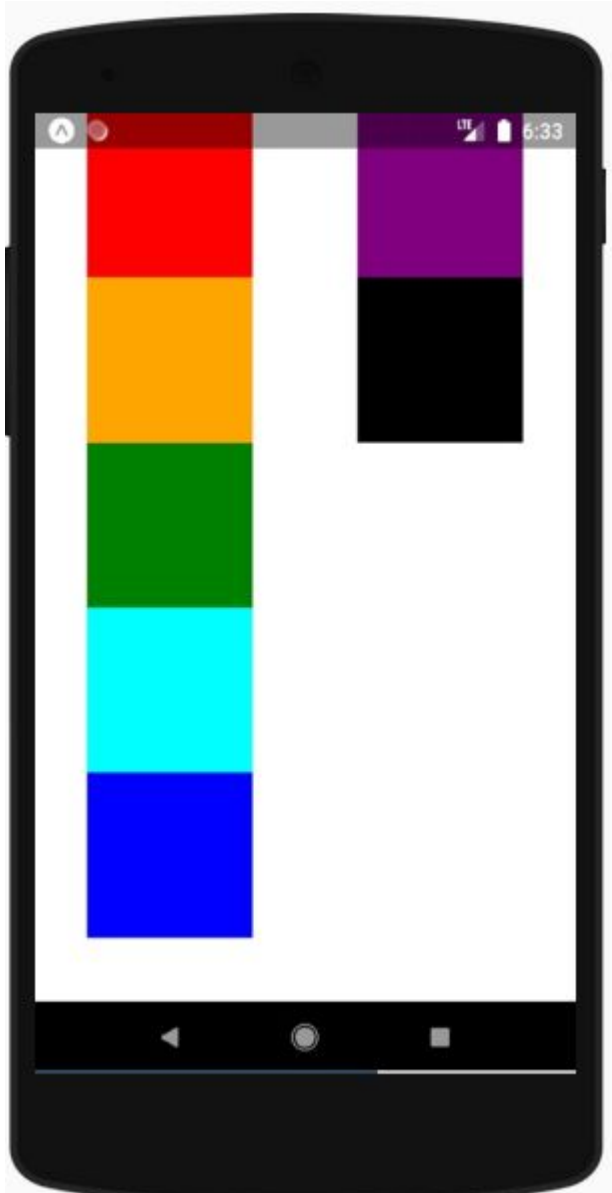
stretch



space-between



space-around

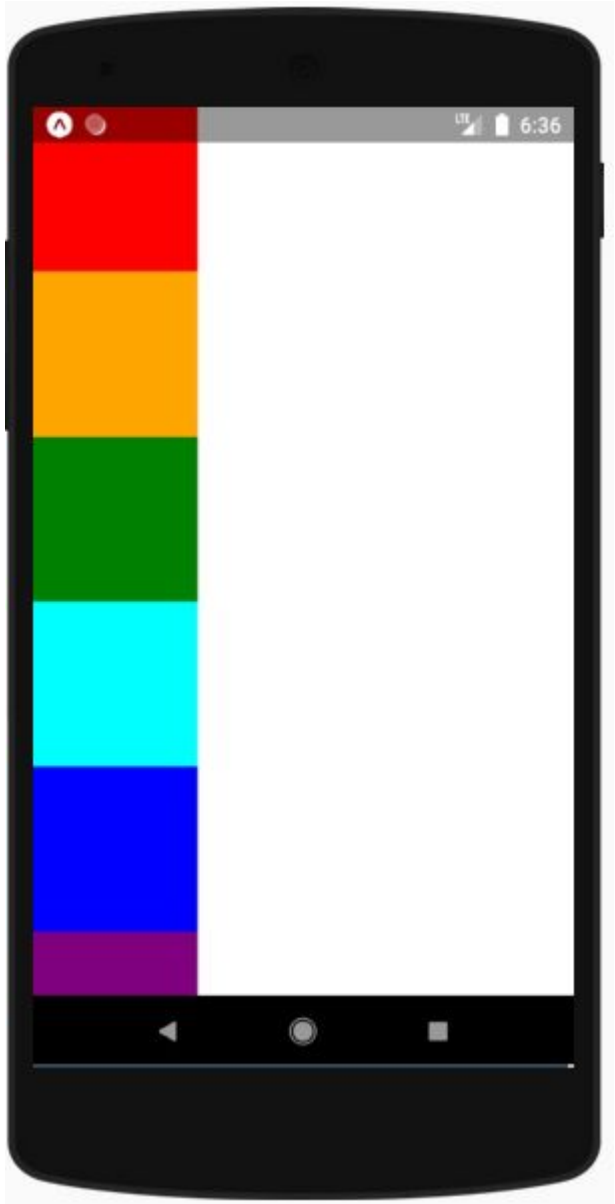


Flex Wrap

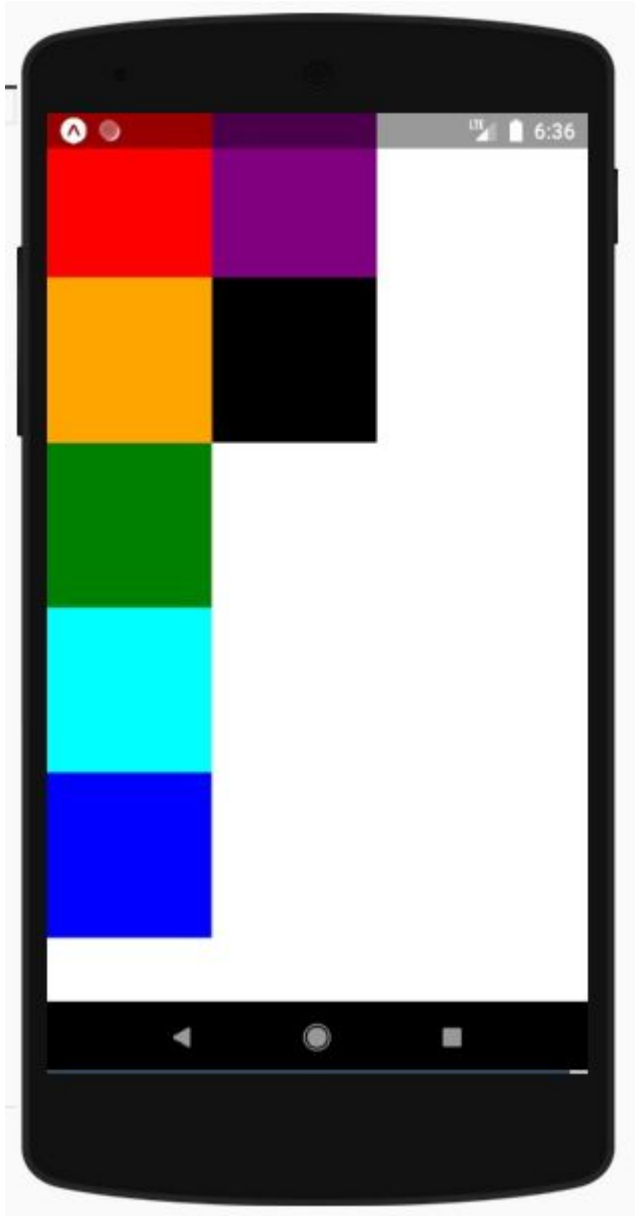
- Свойство flexWrap устанавливается для контейнеров и контролирует, что происходит, когда дочерние элементы переполняют размер контейнера вдоль главной оси. По умолчанию дочерние элементы заключены в одну строку (которая может уменьшать элементы). Если перенос разрешен, то при необходимости элементы оборачиваются в несколько линий вдоль главной оси.

- При переносе строк вы можете использовать alignContent, чтобы указать, как строки будут размещены в контейнере.

nowrap



wrap



Основы гибкости, Рост и Сжатие

- **flexGrow** описывает, как любое пространство внутри контейнера должно быть распределено среди его дочерних элементов вдоль главной оси. После размещения дочерних элементов контейнер будет распределять любое оставшееся пространство в соответствии со значениями flex grow, указанными его дочерними элементами.
- flexGrow принимает любое значение с плавающей запятой> = 0, при этом значение по умолчанию равно 0. Контейнер будет распределять любое оставшееся пространство между дочерними элементами, взвешенными по значению flexGrow дочернего элемента.

- **flexShrink** описывает, как уменьшить дочерние элементы вдоль главной оси в случае, если общий размер дочерних элементов превышает размер контейнера на главной оси. flexShrink очень похоже на flexGrow и может рассматриваться аналогично, если любой переполненный размер считается отрицательным оставшимся пространством. Эти два свойства также хорошо сочетаются друг с другом, позволяя детям расти и уменьшаться по мере необходимости.
- flexShrink принимает любое значение с плавающей запятой> = 0, где значение по умолчанию равно 1. Контейнер будет сокращать свои дочерние элементы, взвешенные по значению flexShrink дочернего элемента.

- **flexBasis** - это независимый от оси способ предоставления размера элемента по умолчанию вдоль главной оси. Установка базового элемента flex для дочернего элемента аналогична установке ширины этого дочернего элемента, если его родителем является контейнер с flexDirection: row, или установка высоты дочернего элемента, если его родительским элементом является контейнер с flexDirection: column.

Абсолютное и относительное расположение

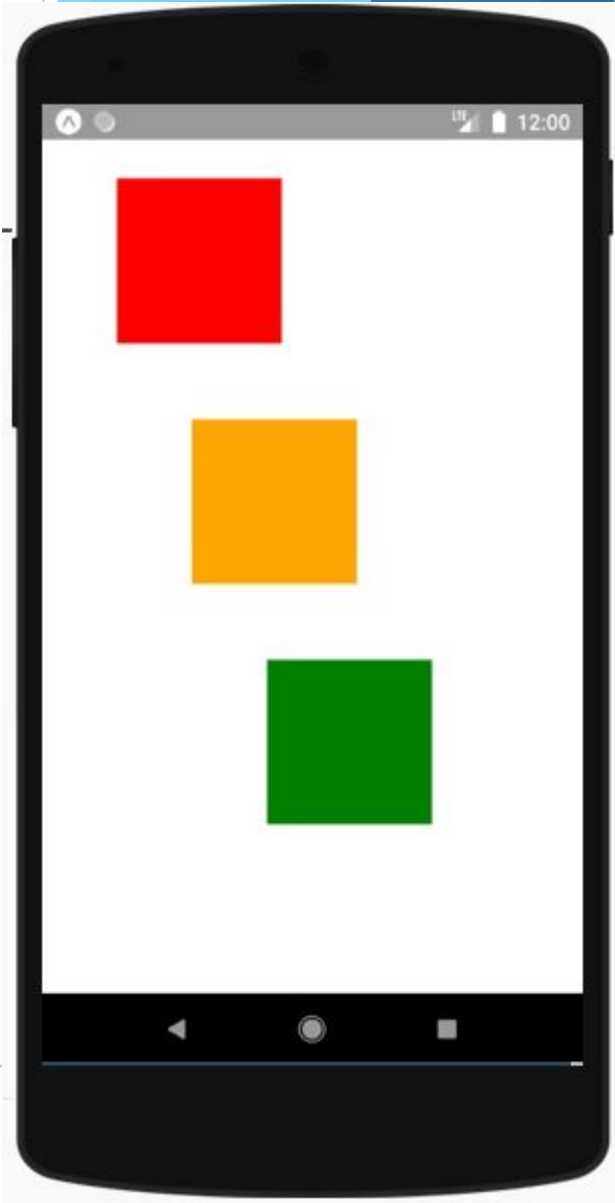
- position элемента определяет, как он позиционируется внутри своего родителя.
- **relative** (значение по умолчанию) – относительное расположение, это означает, что элемент позиционируется в соответствии с обычным потоком макета, а затем смещается относительно этой позиции на основе значений top, right, bottom и left. Смещение не влияет на положение любых родственных или родительских элементов.

Абсолютное и относительное расположение

- **absolute** - Когда позиционировано абсолютно, элемент не участвует в нормальном потоке макета. Вместо этого он расположен независимо от своих братьев и сестер. Положение определяется значениями параметров top, right, bottom, left.

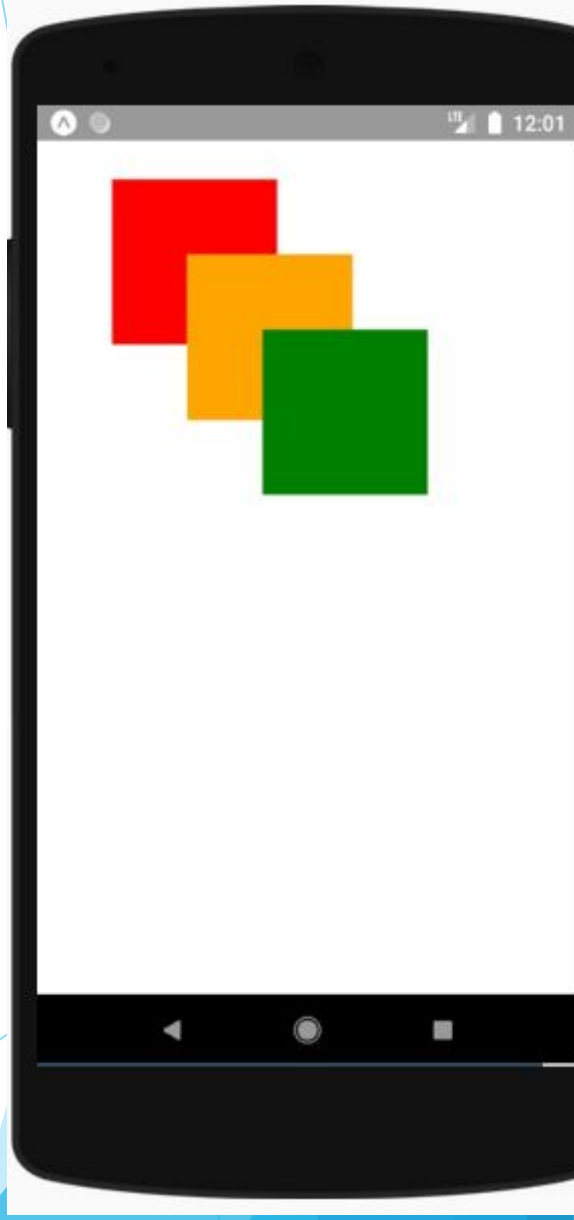
relative

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

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View >
        <View style={{width: 110, height: 110, backgroundColor: 'red', top: 50, left: 50}} />
        <View style={{width: 110, height: 110, backgroundColor: 'orange', top: 100, left: 100}} />
        <View style={{width: 110, height: 110, backgroundColor: 'green', top: 150, left: 150}} />
      </View>
    );
  }
}
```



absolute

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

export default class AlignItemsBasics extends Component {
  render() {
    return (
      <View >
        <View style={{width: 110, height: 110, backgroundColor: 'red', top: 50, left: 50, position: "absolute"}} />
        <View style={{width: 110, height: 110, backgroundColor: 'orange', top: 100, left: 100, position: "absolute"}} />
        <View style={{width: 110, height: 110, backgroundColor: 'green', top: 150, left: 150, position: "absolute"}} />
      </View>
    );
  }
}
```

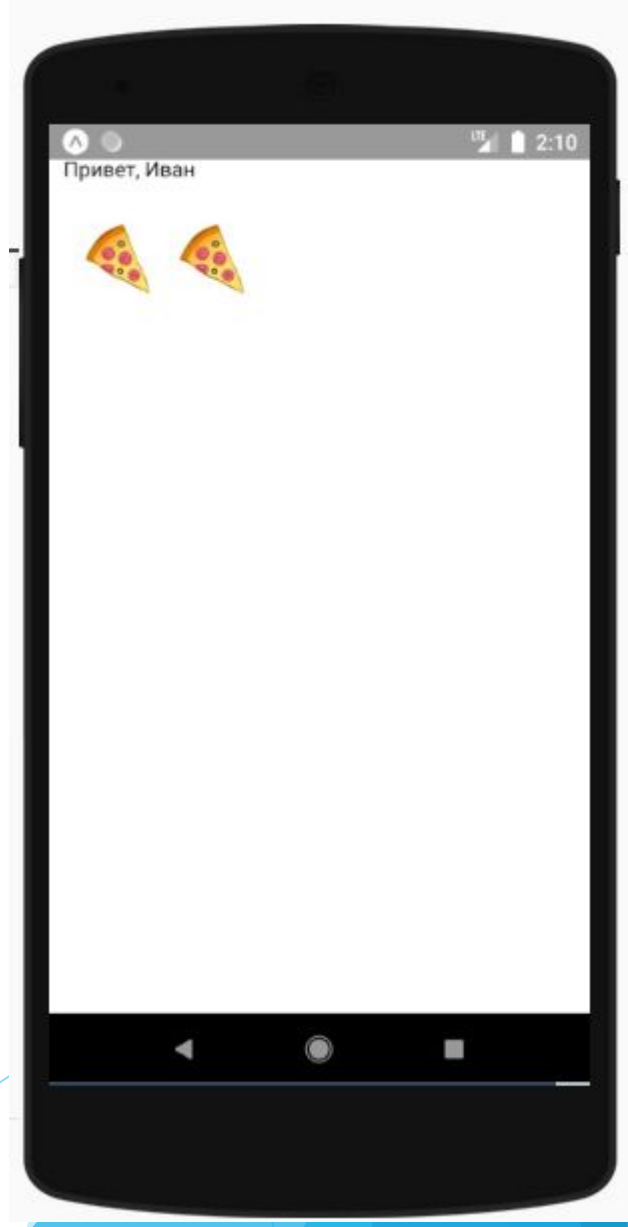


Обработка ввода текста

- TextInput один из основных компонентов, который позволяет пользователю вводить текст. Он имеет свойство onChangeText, который принимает функцию, вызываемую при каждом изменении текста, и свойство onSubmitEditing, который принимает функцию, вызываемую при отправке текста.

  Например, предположим, что когда пользователь печатает, мы переводим его слова на другой язык. На этом новом языке каждое слово пишется одинаково: 🍕. Таким образом, предложение «Привет, Иван» будет переведено как «🍕 🍕».

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

export default class PizzaTranslator extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ''};
  }

  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Напишите сюда то, что хотите перевести!"
          onChangeText={(text) => this.setState({text})}
          value={this.state.text}
        />
        <Text style={{padding: 10, fontSize: 42}}>
          {this.state.text.split(' ').map((word) => word && '🍕').join(' ')}
        </Text>
      </View>
    );
  }
}
```



- В этом примере мы храним текст в состоянии, потому что он меняется со временем.

- Есть еще много вещей, которые вы можете сделать с помощью ввода текста. Например, можно валидировать текст по мере ввода.