**React Native Overview**

**What is React Native?**

React Native is like React, but it uses native components instead of web components as building blocks. So to understand the basic structure of a React Native app, you need to understand some of the basic React concepts, like JSX, components, state, and props. If you already know React, you still need to learn some React-Native-specific stuff, like the native components. This tutorial is aimed at all audiences, whether you have React experience or not.

**Hello World**

According to ancient traditions of our people, we must first build an app that does nothing except say “Hello world”. Here it is:

|  |
| --- |
| 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>  );  }  } |



If you are feeling curious, you can play around with sample code directly in the web simulators. You can also paste it into your App.js file to create a real app on your local machine.

Some of the things in here might not look like JavaScript to you. Don't panic. This is the future.

First of all, ES2015 (also known as ES6) is a set of improvements to JavaScript that is now part of the official standard, but not yet supported by all browsers, so often it isn't used yet in web development. React Native ships with ES2015 support, so you can use this stuff without worrying about compatibility. import, from, class, and extends in the example above are all ES2015 features. If you aren't familiar with ES2015, you can probably pick it up just by reading through sample code like this tutorial has. If you want, this page has a good overview of ES2015 features.

The other unusual thing in this code example is <View><Text>Hello world!</Text></View>. This is JSX - a syntax for embedding XML within JavaScript. Many frameworks use a special templating language which lets you embed code inside markup language. In React, this is reversed. JSX lets you write your markup language inside code. It looks like HTML on the web, except instead of web things like <div> or <span>, you use React components. In this case, <Text> is a built-in component that just displays some text and View is like the <div> or <span>.

**Components**

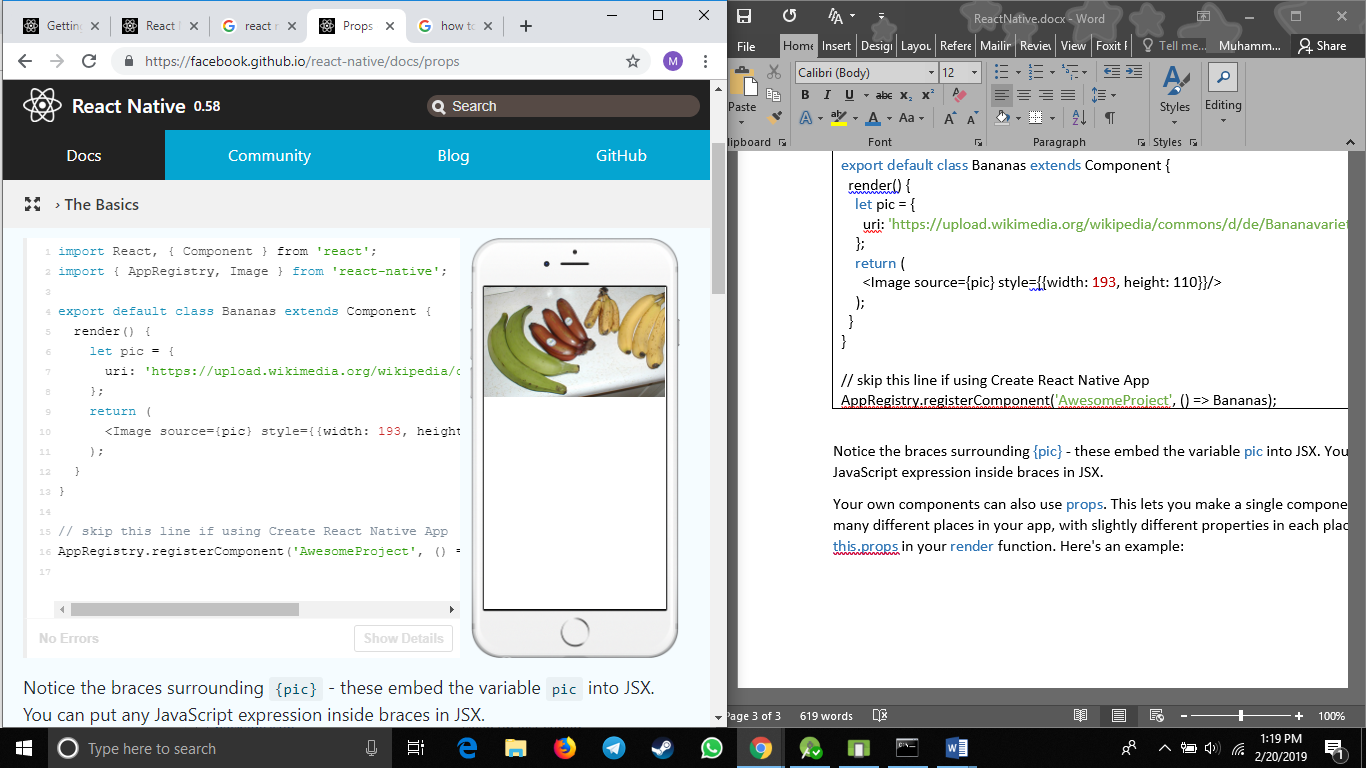
So this code is defining HelloWorldApp, a new Component. When you're building a React Native app, you'll be making new components a lot. Anything you see on the screen is some sort of component. A component can be pretty simple - the only thing that's required is a render function which returns some JSX to render.

**Props**

Most components can be customized when they are created, with different parameters. These creation parameters are called props.

For example, one basic React Native component is the Image. When you create an image, you can use a prop named source to control what image it shows.

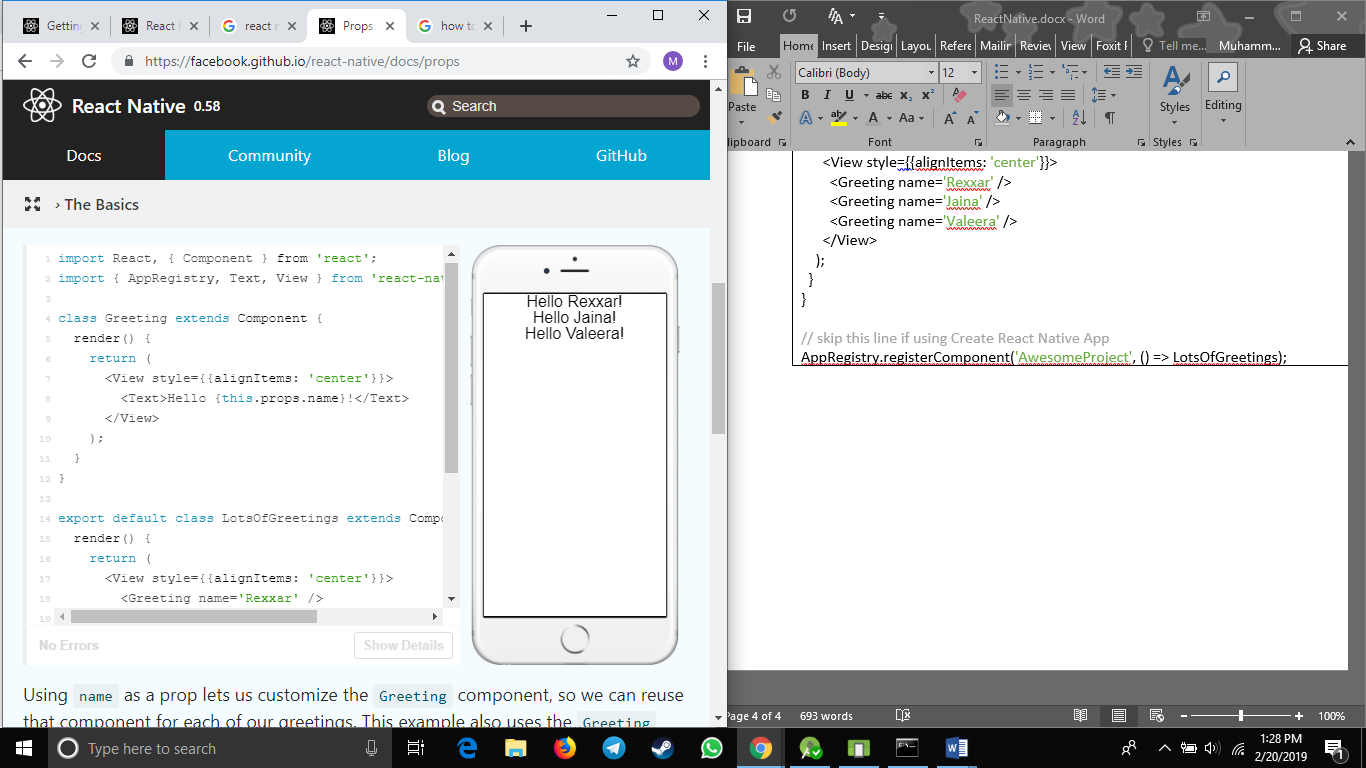
|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, Image } from 'react-native';  export default class Bananas extends Component {  render() {  let pic = {  uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'  };  return (  <Image source={pic} style={{width: 193, height: 110}}/>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => Bananas); |



Notice the braces surrounding {pic} - these embed the variable pic into JSX. You can put any JavaScript expression inside braces in JSX.

Your own components can also use props. This lets you make a single component that is used in many different places in your app, with slightly different properties in each place. Just refer to this.props in your render function. Here's an example:

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, Text, View } from 'react-native';  class Greeting extends Component {  render() {  return (  <View style={{alignItems: 'center'}}>  <Text>Hello {this.props.name}!</Text>  </View>  );  }  }  export default class LotsOfGreetings extends Component {  render() {  return (  <View style={{alignItems: 'center'}}>  <Greeting name='Rexxar' />  <Greeting name='Jaina' />  <Greeting name='Valeera' />  </View>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => LotsOfGreetings); |



Using name as a prop lets us customize the Greeting component, so we can reuse that component for each of our greetings. This example also uses the Greeting component in JSX, just like the built-in components. The power to do this is what makes React so cool - if you find yourself wishing that you had a different set of UI primitives to work with, you just invent new ones.

The other new thing going on here is the View component. A View is useful as a container for other components, to help control style and layout.

With props and the basic Text, Image, and View components, you can build a wide variety of static screens. To learn how to make your app change over time, you need to learn about State.

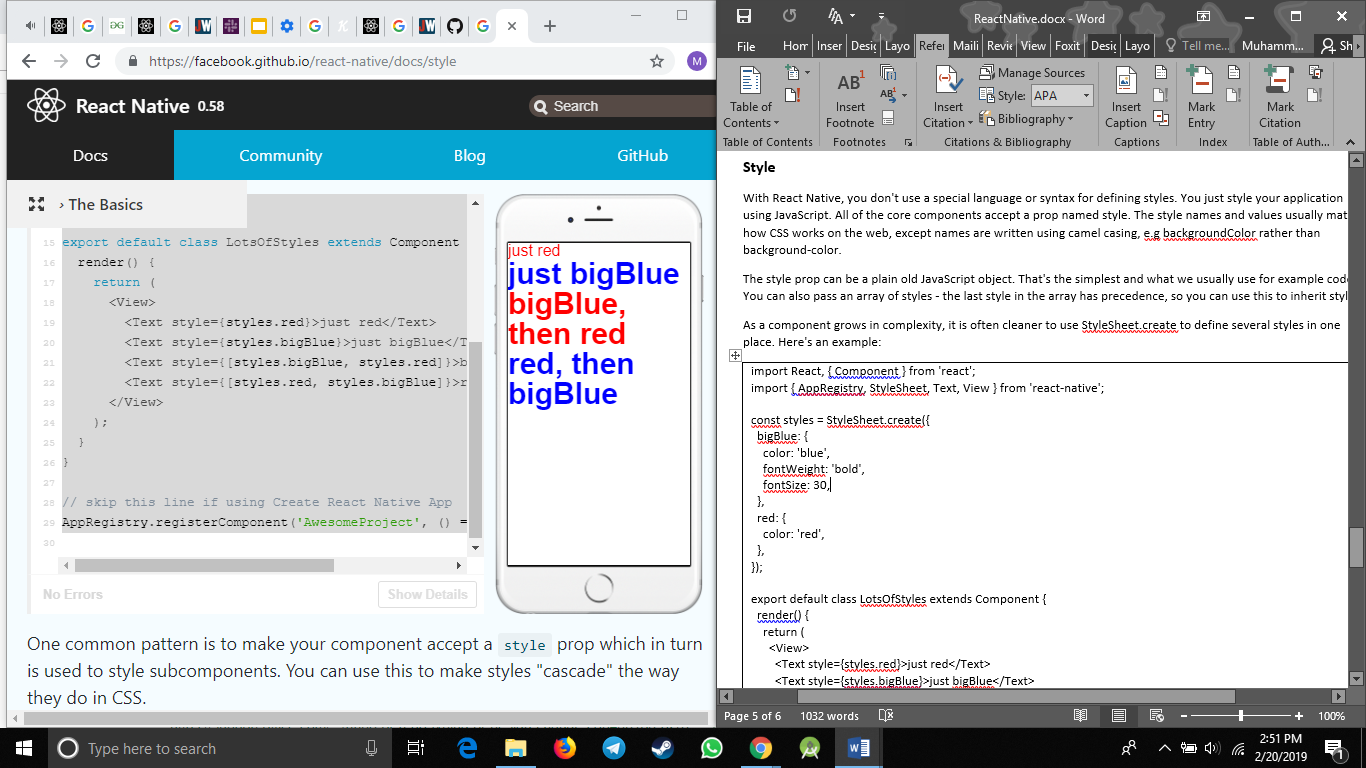
**Style**

With React Native, you don't use a special language or syntax for defining styles. You just style your application using JavaScript. All of the core components accept a prop named style. The style names and values usually match how CSS works on the web, except names are written using camel casing, e.g backgroundColor rather than background-color.

The style prop can be a plain old JavaScript object. That's the simplest and what we usually use for example code. You can also pass an array of styles - the last style in the array has precedence, so you can use this to inherit styles.

As a component grows in complexity, it is often cleaner to use StyleSheet.create to define several styles in one place. Here's an example:

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, 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}>just red</Text>  <Text style={styles.bigBlue}>just bigBlue</Text>  <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>  <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>  </View>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => LotsOfStyles); |



One common pattern is to make your component accept a style prop which in turn is used to style subcomponents. You can use this to make styles "cascade" the way they do in CSS.

There are a lot more ways to customize text style. Check out the Text component reference for a complete list.

Now you can make your text beautiful. The next step in becoming a style master is to learn how to control component size.

**Height and Width**

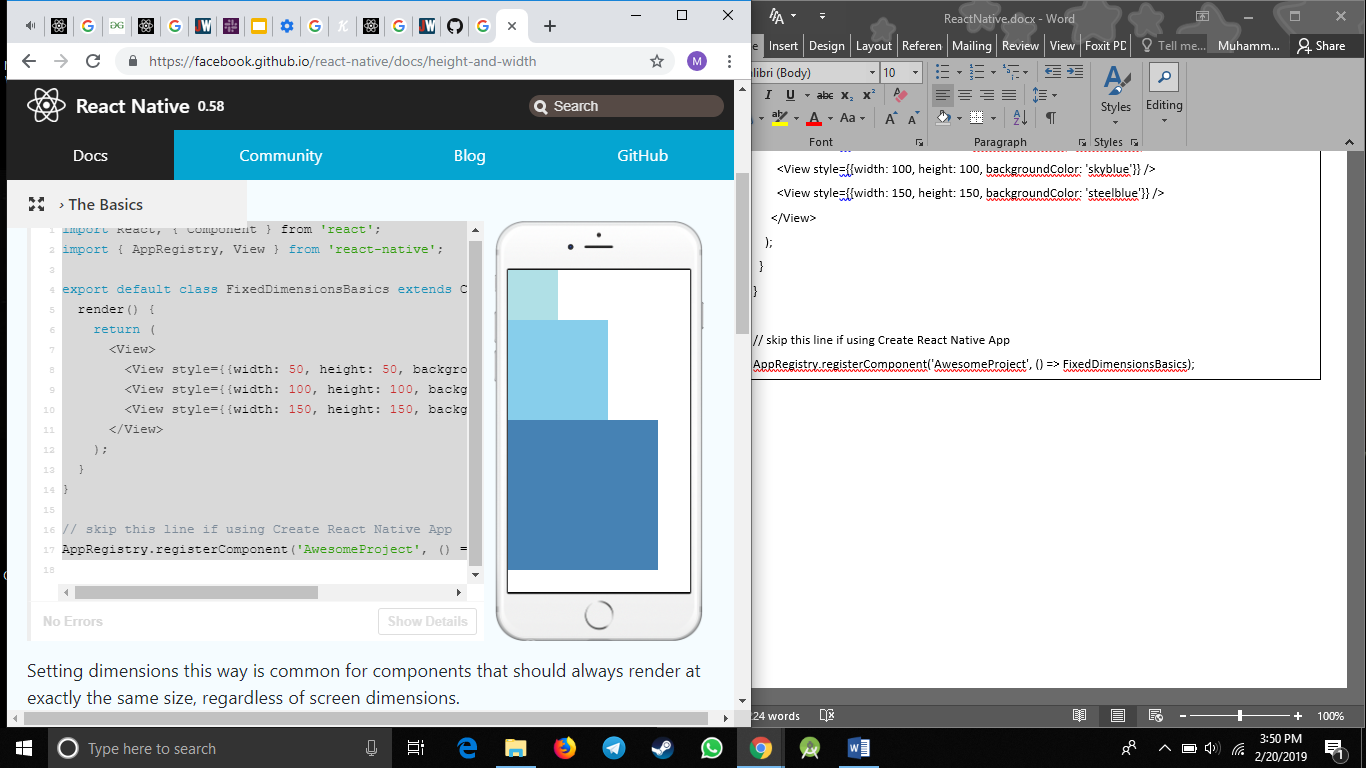
A component's height and width determine its size on the screen.

Fixed Dimensions

The simplest way to set the dimensions of a component is by adding a fixed width and height to style. All

dimensions in React Native are unitless, and represent density-independent pixels.

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, 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>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => FixedDimensionsBasics); |



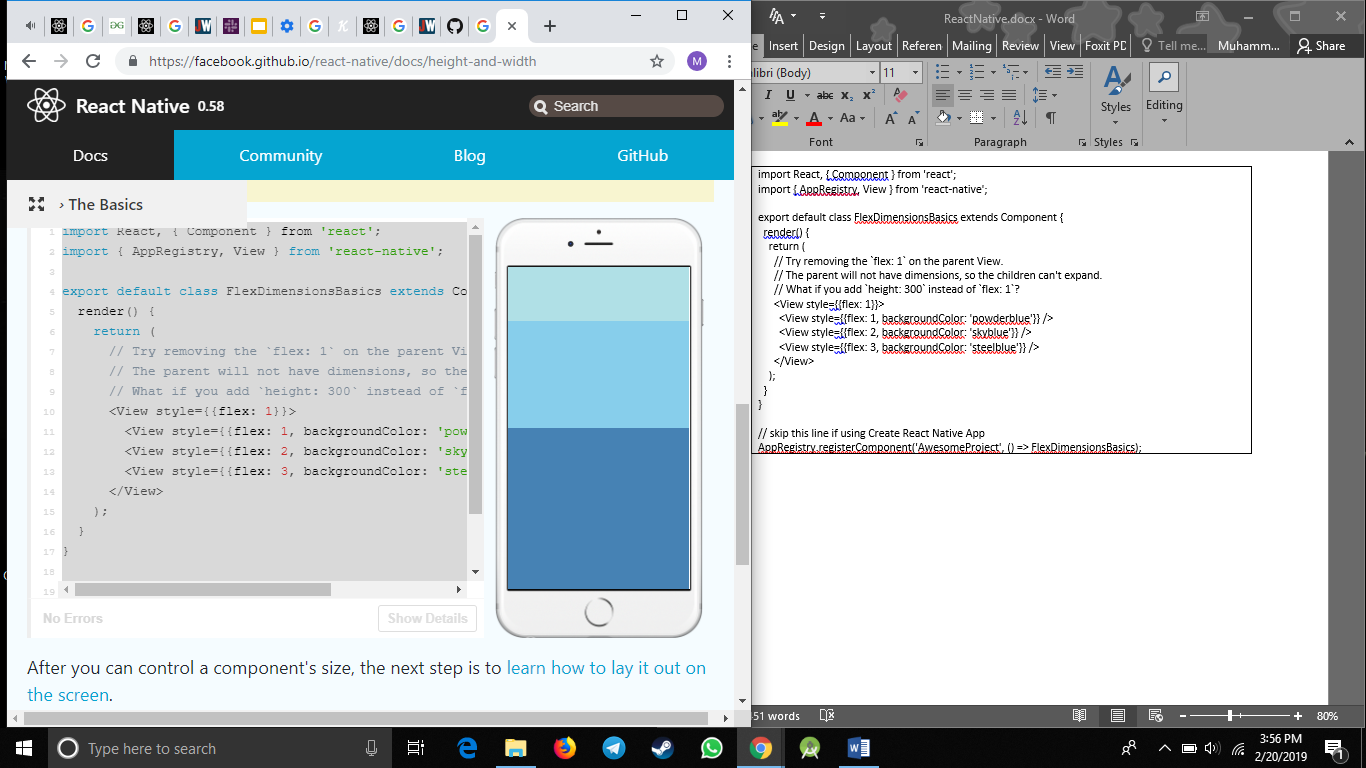
Setting dimensions this way is common for components that should always render at exactly the same size, regardless of screen dimensions.

**Flex Dimensions**

Use flex in a component's style to have the component expand and shrink dynamically based on available space. Normally you will use flex: 1, which tells a component to fill all available space, shared evenly amongst other components with the same parent. The larger the flex given, the higher the ratio of space a component will take compared to its siblings.

A component can only expand to fill available space if its parent has dimensions greater than 0. If a parent does not have either a fixed width and height or flex, the parent will have dimensions of 0 and the flex children will not be visible.

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, View } from 'react-native';  export default class FlexDimensionsBasics extends Component {  render() {  return (  // Try removing the `flex: 1` on the parent View.  // The parent will not have dimensions, so the children can't expand.  // What if you add `height: 300` instead of `flex: 1`?  <View style={{flex: 1}}>  <View style={{flex: 1, backgroundColor: 'powderblue'}} />  <View style={{flex: 2, backgroundColor: 'skyblue'}} />  <View style={{flex: 3, backgroundColor: 'steelblue'}} />  </View>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => FlexDimensionsBasics); |



After you can control a component's size, the next step is to learn how to lay it out on the screen.

**Layout with Flexbox**

A component can specify the layout of its children using the flexbox algorithm. Flexbox is designed to provide a consistent layout on different screen sizes.

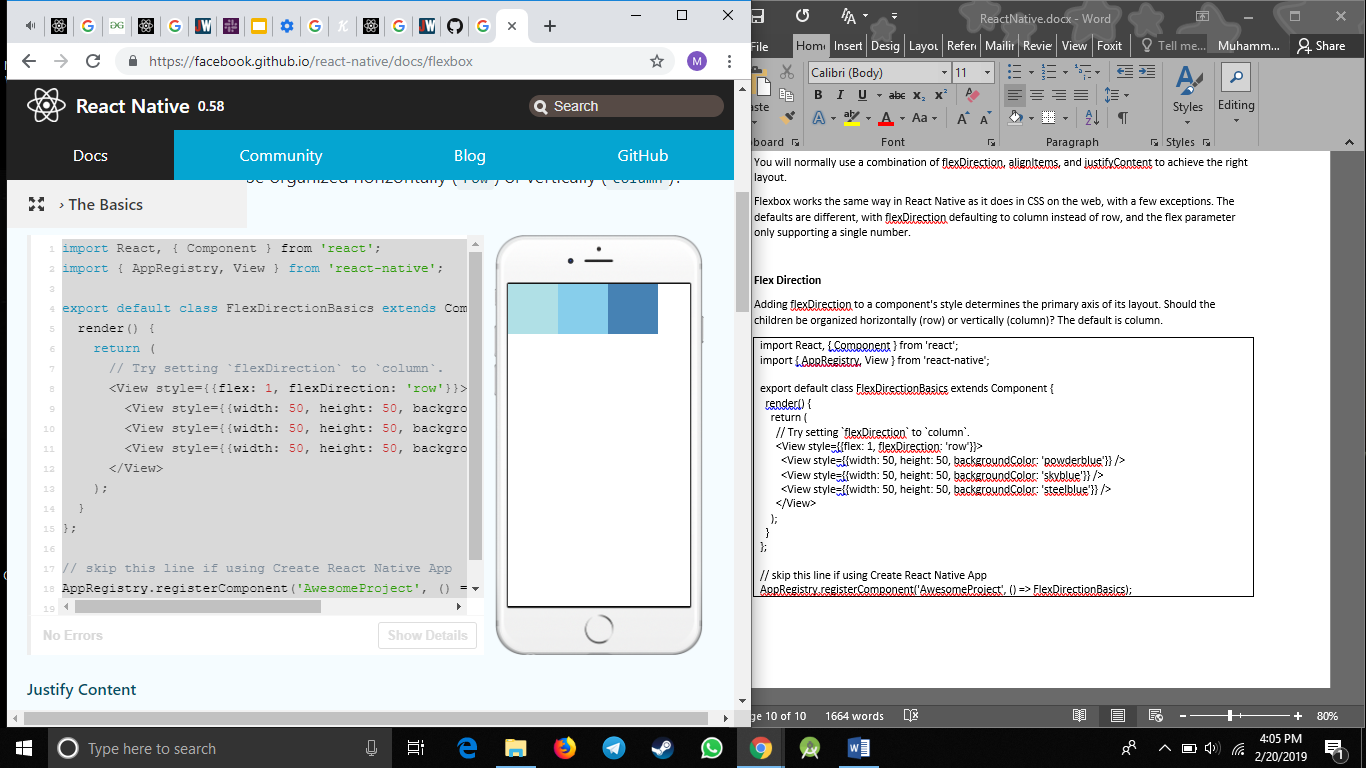
You will normally use a combination of flexDirection, alignItems, and justifyContent to achieve the right layout.

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.

**Flex Direction**

Adding flexDirection to a component's style determines the primary axis of its layout. Should the children be organized horizontally (row) or vertically (column)? The default is column.

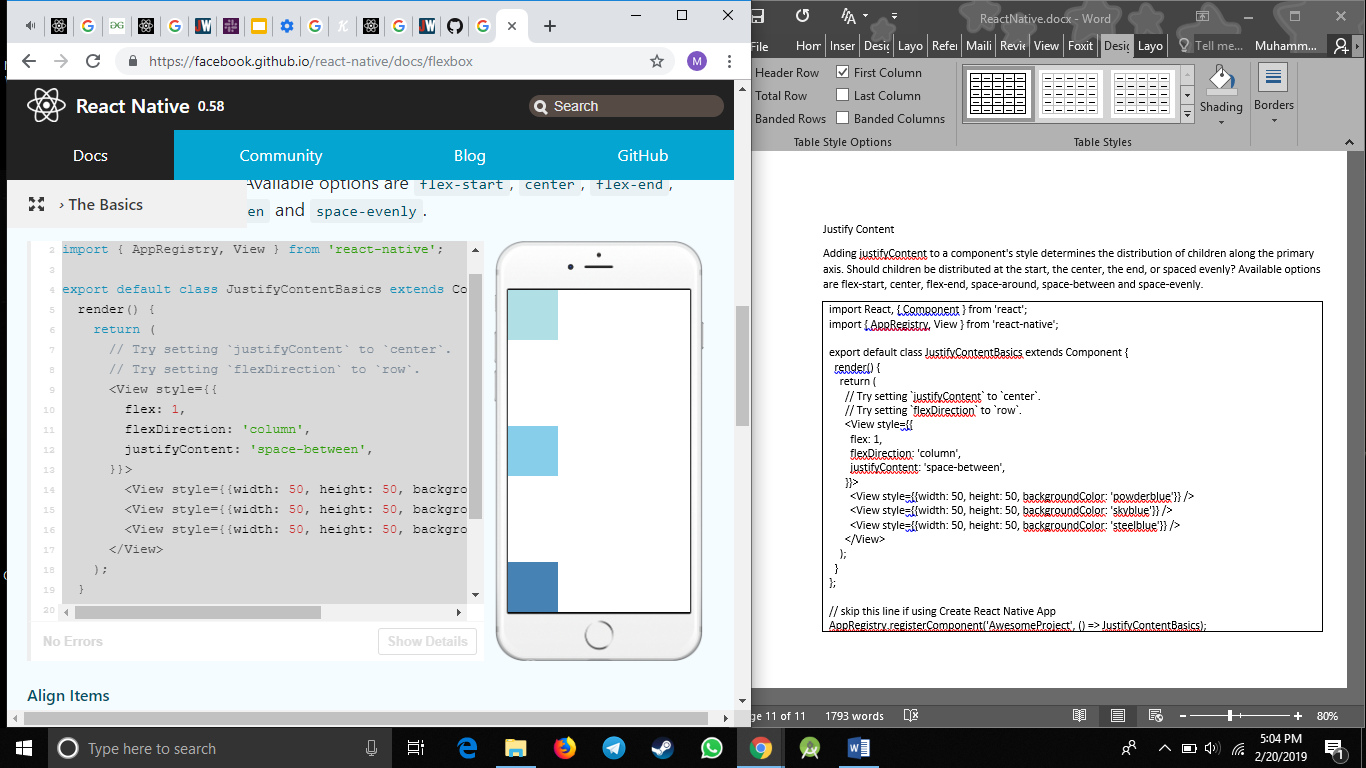
|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, View } from 'react-native';  export default class FlexDirectionBasics extends Component {  render() {  return (  // Try setting `flexDirection` to `column`.  <View style={{flex: 1, flexDirection: 'row'}}>  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  </View>  );  }  };  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => FlexDirectionBasics); |



**Justify Content**

Adding justifyContent to a component's style determines the distribution of children along the primary axis. Should children be distributed at the start, the center, the end, or spaced evenly? Available options are flex-start, center, flex-end, space-around, space-between and space-evenly.

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, View } from 'react-native';  export default class JustifyContentBasics extends Component {  render() {  return (  // Try setting `justifyContent` to `center`.  // Try setting `flexDirection` to `row`.  <View style={{  flex: 1,  flexDirection: 'column',  justifyContent: 'space-between',  }}>  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />  </View>  );  }  };  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => JustifyContentBasics); |

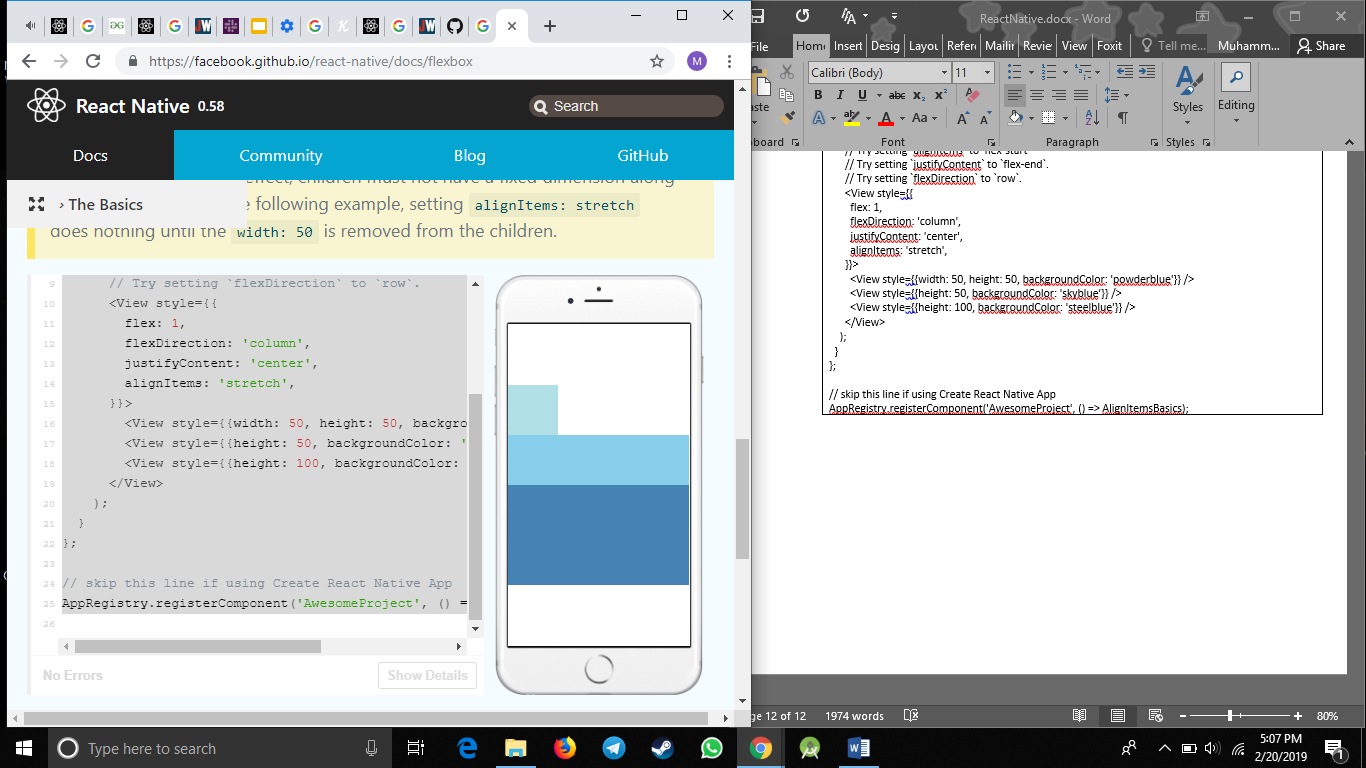


**Align Items**

Adding alignItems to a component's style determines the alignment of children along the secondary axis (if the primary axis is row, then the secondary is column, and vice versa). Should children be aligned at the start, the center, the end, or stretched to fill? Available options are 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 example, setting alignItems: stretch does nothing until the width: 50 is removed from the children.

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, View } from 'react-native';  export default class AlignItemsBasics extends Component {  render() {  return (  // Try setting `alignItems` to 'flex-start'  // Try setting `justifyContent` to `flex-end`.  // Try setting `flexDirection` to `row`.  <View style={{  flex: 1,  flexDirection: 'column',  justifyContent: 'center',  alignItems: 'stretch',  }}>  <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />  <View style={{height: 50, backgroundColor: 'skyblue'}} />  <View style={{height: 100, backgroundColor: 'steelblue'}} />  </View>  );  }  };  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => AlignItemsBasics); |

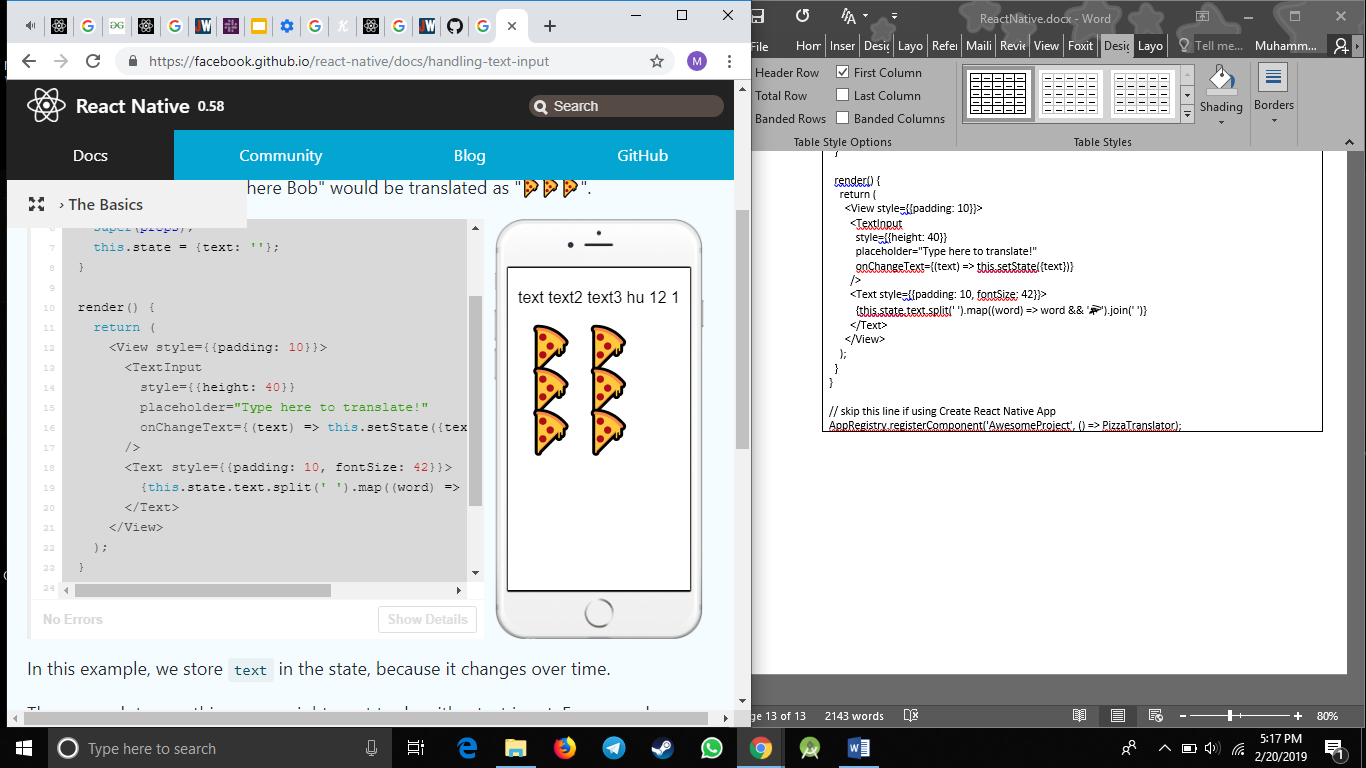


**Handling Text Input**

TextInput is a basic component that allows the user to enter text. It has an onChangeText prop that takes a function to be called every time the text changed, and an onSubmitEditing prop that takes a function to be called when the text is submitted.

For example, let's say that as the user types, you're translating their words into a different language. In this new language, every single word is written the same way: 🍕. So the sentence "Hello there Bob" would be translated as "🍕🍕🍕".

|  |
| --- |
| import React, { Component } from 'react';  import { AppRegistry, 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="Type here to translate!"  onChangeText={(text) => this.setState({text})}  />  <Text style={{padding: 10, fontSize: 42}}>  {this.state.text.split(' ').map((word) => word && '🍕').join(' ')}  </Text>  </View>  );  }  }  // skip this line if using Create React Native App  AppRegistry.registerComponent('AwesomeProject', () => PizzaTranslator); |



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

There are a lot more things you might want to do with a text input. For example, you could validate the text inside while the user types. For more detailed examples, see the React docs on controlled components, or the reference docs for TextInput.

Text input is one of the ways the user interacts with the app. Next, let's look at another type of input and learn how to handle touches.

**Handling Touches**

Users interact with mobile apps mainly through touch. They can use a combination of gestures, such as tapping on a button, scrolling a list, or zooming on a map. React Native provides components to handle all sorts of common gestures, as well as a comprehensive gesture responder system to allow for more advanced gesture recognition, but the one component you will most likely be interested in is the basic Button.

**Displaying a basic button**

Button provides a basic button component that is rendered nicely on all platforms. The minimal example to display a button looks like this:

|  |
| --- |
| <Button  onPress={() => {  Alert.alert('You tapped the button!');  }}  title="Press Me"  /> |

This will render a blue label on iOS, and a blue rounded rectangle with white text on Android. Pressing the button will call the "onPress" function, which in this case displays an alert popup. If you like, you can specify a "color" prop to change the color of your button.



Go ahead and play around with the Button component using the example below. You can select which platform your app is previewed in by clicking on the toggle in the bottom right, then click on "Tap to Play" to preview the app.