

**React Native**

**Basic Core Components**

### [View](https://reactnative.dev/docs/view)

[The most fundamental component for building a UI.](https://reactnative.dev/docs/view)

### [Text](https://reactnative.dev/docs/text)

[A component for displaying text.](https://reactnative.dev/docs/text)

### [Image](https://reactnative.dev/docs/image)

[A component for displaying images.](https://reactnative.dev/docs/image)

### [TextInput](https://reactnative.dev/docs/textinput)

[A component for inputting text into the app via a keyboard.](https://reactnative.dev/docs/textinput)

### [ScrollView](https://reactnative.dev/docs/scrollview)

[Provides a scrolling container that can host multiple components and views.](https://reactnative.dev/docs/scrollview)

### [StyleSheet](https://reactnative.dev/docs/stylesheet)

[Provides an abstraction layer similar to CSS stylesheets.](https://reactnative.dev/docs/stylesheet)

**Text Input**

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

import React, { useState } from 'react';

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

const **PizzaTranslator** = () => {

const [**text**, **setText**] = useState('');

return (

<View style={{ padding: 10 }}>

<TextInput

style={{ height: 40 }}

placeholder="Type here to translate!"

onChangeText={text => setText(text)}

defaultValue={text}

/>

<Text style={{ padding: 10, fontSize: 42 }}>

{text.split(' ').map((word) => word && '🍕').join(' ')}

</Text>

</View>

);

}

export default PizzaTranslator;

**ScrollView**

The ScrollView is a generic scrolling container that can contain multiple components and views. The scrollable items need not be homogeneous, and you can scroll both vertically and horizontally (by setting the horizontal property).

export default App = () => (

<ScrollView>

<Text style={{ fontSize: 96 }}>Scroll me plz</Text>

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

<Text style={{ fontSize: 96 }}>If you like</Text>

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

<Image source={logo} />

</ScrollView>

);

**List Views**

The FlatList component requires two props: data and renderItem. data is the source of information for the list. renderItem takes one item from the source and returns a formatted component to render.

import React from 'react';

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

const **styles** = StyleSheet.create({

container: {

flex: 1,

paddingTop: 22

},

item: {

padding: 10,

fontSize: 18,

height: 44,

},

});

const **FlatListBasics** = () => {

return (

<View style={styles.container}>

<FlatList

data={[

{ key: 'Devin' },

{ key: 'Dan' },

{ key: 'Dominic' },

{ key: 'Jackson' },

{ key: 'James' },

{ key: 'Joel' },

{ key: 'John' },

{ key: 'Jillian' },

{ key: 'Jimmy' },

{ key: 'Julie' },

]}

renderItem={({ item }) => <Text style={styles.item}>{item.key}</Text>}

/>

</View>

);

}

export default FlatListBasics;

**Platform Specific Code**

When building a cross-platform app, you'll want to re-use as much code as possible. Scenarios may arise where it makes sense for the code to be different, for example you may want to implement separate visual components for Android and iOS.

React Native provides two ways to organize your code and separate it by platform:

* Using the **Platform module**.
* Using **platform-specific file extensions**.

Certain components may have properties that work on one platform only. All of these props are annotated with @platform and have a small badge next to them on the website.

**Platform module**

React Native provides a module that detects the platform in which the app is running. You can use the detection logic to implement platform-specific code. Use this option when only small parts of a component are platform-specific.

import { Platform, StyleSheet } from 'react-native';

const **styles** = StyleSheet.create({

height: Platform.**OS** === 'ios' ? 200 : 100

});

<https://reactnative.dev/docs/platform-specific-code>

**Design**

**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.



## **Static Image Resources**

React Native provides a unified way of managing images and other media assets in your Android and iOS apps. To add a static image to your app, place it somewhere in your source code tree and reference it like this.

<Image source={require('./my-icon.png')} />

**Network Images**

Many of the images you will display in your app will not be available at compile time, or you will want to load some dynamically to keep the binary size down. Unlike with static resources, you will need to manually specify the dimensions of your image. It's highly recommended that you use https as well in order to satisfy App Transport Security requirements on iOS.

*// GOOD*

<Image source={{uri: 'https://reactjs.org/logo-og.png'}}

style={{width: 400, height: 400}} />

*// BAD*

<Image source={{uri: 'https://reactjs.org/logo-og.png'}} />

**Background Image via Nesting**

A common feature request from developers familiar with the web is background-image. To handle this use case, you can use the <ImageBackground> component, which has the same props as <Image>, and add whatever children to it you would like to layer on top of it.

return (

<ImageBackground source={...} style={{width: '100%', height: '100%'}}>

<Text>Inside</Text>

</ImageBackground>

);

# **Handling Touches**

onPress

<https://reactnative.dev/docs/handling-touches#touchables> // handling react native componentes

**Navigating Between Screens**

Mobile apps are rarely made up of a single screen. Managing the presentation of, and transition between, multiple screens is typically handled by what is known as a navigator.

This guide covers the various navigation components available in React Native. If you are getting started with navigation, you will probably want to use React Navigation. React Navigation provides a straightforward navigation solution, with the ability to present common stack navigation and tabbed navigation patterns on both Android and iOS.

import \* as React from 'react';

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

import { createStackNavigator } from '@react-navigation/stack';

const **Stack** = createStackNavigator();

const **MyStack** = () => {

return (

<NavigationContainer>

<Stack.Navigator>

<Stack.Screen

name="Home"

component={HomeScreen}

options={{ title: 'Welcome' }}

/>

<Stack.Screen name="Profile" component={ProfileScreen} />

</Stack.Navigator>

</NavigationContainer>

);

};

const HomeScreen = ({ navigation }) => {

return (

<Button

title="Go to Jane's profile"

onPress={() =>

navigation.navigate('Profile', { name: 'Jane' })

}

/>

);

};

const ProfileScreen = () => {

return <Text>This is Jane's profile</Text>;

Animated REACT NATIVE

<https://reactnative.dev/docs/animations>