React Native Cheat Sheet

by Bochrak via cheatography.com/200241/cs/42327/

Development environment Set-up

Expo	☞ *For beginners* ✔ Simplifies the setup process ✔ Provides OTA updates ✖ Does not allow you to add custom native code ✖ Expo apps tend to have larger sizes
React Native CLI	☞ *For Experienced Developers* ✔ Supports integrating custom native modules ✔ Potentially better performance for complex applications ✖ Requires Xcode or Android Studio to get started. ✖ No OTA updates.

Creating an app

Initialize a new project	`npx create -ex po-app my-app`

Start development server	`cd my-app`
	`npm start`

Running app

Android	Use the Expo Go app to scan the QR code from your terminal to open your project.
iPhone	Use the built-in QR code scanner of the default iOS Camera app.

☞ *Connect to the same wireless network as your computer.*

Metro

↪ When you run your app, the Expo CLI starts Metro Bundler. It's a JavaScript bundler that takes all your JavaScript files and assets, bundles them, and transforms them using Babel. This process converts the code into a format that can be executed by the platform running the app (iOS or Android).

Expo

Expo	A set of tools and services to make development with React Native easier.
Expo SDK	A modular set of packages that provides access that provides access to native APIs, like Camera or Notifications.
Expo CLI	A command-line tool that is the primary interface between a developer and other Expo tools.
Expo Go	An open-source sandbox app you can download on your phone to view your app in development.

Expo (cont)

Expo Snack	A web-based playground where you can write React Native snippets and run them in the browser.
Expo Tunnel	For establishing a connection that allows devices to access the app even if they're not on the same wireless network. `npx expo start --tunnel`

Finding Libraries

🗁 React Native Directory is a searchable database of libraries built specifically for React Native.

StyleSheet

^ An abstraction similar to CSS StyleSheets.
^ Declare styles in a structured and optimized manner.
^ You can use an array of styles to combine multiple style objects- the last style in the array has precedence, or mix predefined styles with inline styles.
^ All of the core components accept a **prop** named **style**.

```
import React from 'react';
import {Style Sheet, Text, View} from 'react-native'
;
const App = () => (
<View style={styles.container}>
 <Text style= {[s tyl es.b as eText, styles.boldText
]}>
    This is bold and black text
 </Text>
 <Text style= {[s tyl es.b as eText, { color: 'blue'
 }]}>
     This is blue and normal weight text
  </Text>
</View>
);
const styles = StyleS hee t.c reate({
 container: { flex: 1,
              padding: 24,
              backgr oun dColor: '#eaeaea' },
 baseText: { fontSize: 16,
                    color: 'black' },
        boldText: { fontWe ight: 'bold' }
  }
);
export default App;
```

UseColorScheme Hook

^ Provides and subscribes to color scheme updates from the appearance module in react native.
^ It returns the current color scheme preference of the user's device.
^ Supported color schemes: "light", "dark", null.

```
import React from 'react';
import
 {Text, StyleS heet, useCol orS cheme, View}
from 'react-native';
const App = () => {
const colorS cheme = useColorScheme();
return (
<View style={styles.container}>
  <Te xt> use Col orS che me(): {colorScheme}</Tex
t>
</View>
    );
};
const styles = StyleSheet.create({
     container: {
       flex: 1,
       alignI tems: 'center',
       justif yCo ntent: 'cente r'}});
export default App;
```

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

useWindowDimensions Hook

- Used to get the dimensions of the device window.
- It returns an object containing the window's width and height.
- Useful for creating responsive designs and layouts that adapt to different screen sizes.

```
import React from 'react';
import
{View, StyleS heet, Text, useWindowDimensions}
from 'react-native';
const App = () => {
const {height,width,scale,fontScale}=useWindowDimensions();
return (
<View style={styles.container}>
  <Te xt> Window Dimension Data</Text>
  <Te xt> Height: {height}</Text>
  <Te xt> Width: {width}</Text>
  <Te xt>Font scale: {fontScale}</Text>
  <Te xt> Pixel ratio: {scale}</Text>
</View>
    );
};
const styles = StyleSheet.create(
     container: {
       flex: 1,
       justif yCo ntent: 'center',
       alignI tems: 'center'},
      });
export default App;
```

Button

- A basic button component that should render on any platform.
- Supports a minimal level of customization.

```
import React from 'react';
import { View, Button } from 'react -na tive';
const Exampl eButton = () => {
const handle Press = () => {conso le.l og ('B utton pressed');};
return (
<View>
<Button title= " Click Me" onPres s={ han dle Press} color= " #84
 158 4"/>
</View>
    );
};
export default Exampl eBu tton;
```

⚠ **Required props: title** and **onPress**

Pressable

Pressable (cont)

⚠ **props:**

- **onPressIn:** method called when a press is activated.
- **onPressOut:** method called when the press gesture is deactivated.
- **onLongPress:** method called when user leaves their finger longer than 500 milliseconds before removing it, customize this time period using the delayLongPress prop.
- **pressed:** state that refers to a boolean value provided to the style and children functions of Pressable, to check if component is being pressed.
- **hitSlop:** prop to increase the area where touch gestures are recognized. (extended interactive area "HitRect").
- **pressRetentionOffset:** prop to specify the area in which the touch can move while maintaining the press's active state. ("PressRect").

Navigation

React Navigation

- React Native does not come with built-in navigation capabilities.
- → *React Navigation* is the most popular third-party library.
- Enable developers to implement various navigation patterns.
- Provides a set of navigators, such as stack, tab, and drawer navigators.

Stack Navigator

- Used for users press interactions.
- Detects various stages of press interactions on any of its child components.
- Highly customizable and flexible way to handle touch-based input.
- Inherits all the styles of the View component.

```
import React from 'react';
import { Pressable, Text } from 'react-native';
const Exampl ePr essable = () => {
return (
<Pressable onPres s={() => consol e.l og( 'Pr ess ed!')}
   style={({ pressed }) =>  [
    {backg rou ndC olor: pressed ? 'light sky blue' : 'light gra
y'},
    {padding: 10, alignI tems: 'center' }]}
    hitSlop={{top: 10, bottom: 10, left: 10, right: 10 }}
    pressRetentionOffset={{top: 20, bottom: 20, left: 20}} >
        <Te xt> Press Me</Text>
  </P res sab le>
 );
};
export default Exampl ePr ess able;
```

- Allows transition between screens where each new screen is placed on top of a stack.

- **NavigationContainer:** Component container for your app's navigation structure.
  - Manages the navigation tree and contains the navigation state.
  - Should be only used once in your app at the root.

- **createNativeStackNavigator:** Function that returns an object containing two properties.
  - **Navigator:** Takes Screen elements as its children to define the configuration for routes.
    initialRouteName: prop for the Navigator specify what the initial route in a stack is.
    screenOptions: prop to Navigator to specify common options.
  - **Screen**: Component takes 2 required props name and component.
    name: prop which refers to the name of the route.
    component: prop which specifies the component to render for the route.
    options: prop to Screen to specify screen-specific options.

- **navigation and route props:** are automatically provided to each screen component by the navigator.
  navigation: prop is available to all screen components and allows you to control navigation actions.
  route: prop contains information about the route, including parameters passed to that screen.
  You can read the params through **route.params** inside a screen.
  Params should contain the minimal data required to show a screen.

Stack Navigator (cont)

```
import * as React from 'react';
import { View, Text, Button } from 'react-native';
import { create Sta ckN avi gator } from '@react-navigation/stack';
import { Naviga tio nCo ntainer } from '@react-navigation/native';
const HomeScreen = ({ navigation }) => {
return (
<View style={{ flex: 1, alignI tems: 'center', justif yCo ntent: 'center' }}>
 <Te xt>Home Screen</Text>
  <Button title= "Go to Detail s"
          onPres s={() => naviga tio n.n avi gat e(' Details', { someParam: 'Hello!' })} />
</View>
);
};
const Detail sScreen = ({ route }) => {
return (
<View style={{ flex: 1, alignI tems: 'center', justif yCo ntent: 'center' }}>
 <Te xt> Details Screen</Text>
 <Te xt> Par ameter: {route.params.someParam}</Text>
</View>
);
};
const Stack = createStackNavigator();
function App() {
return (
<NavigationContainer>
  <St ack.Na vigator  initialRouteName="Home"
             screenOptions={{ header Style: {backg rou ndC olor: '#f451 1e'}}} >
    <St ack.Screen name="H ome " compon ent ={H ome Screen} options={{ title: 'My Home' }} />
    <St ack.Screen name="D eta ils " compon ent ={D eta ils Screen} options={{ title: 'Detail Vie
w' }} />
   </S tac k.N avi gat or>
</NavigationContainer>
);
}
export default App;
```

Navigation actions:

navigation.navigate('RouteName'): Pushes a new route to the native stack navigator if it's not already in the stack.

❗ If you navigate to a route that is not defined in the navigator, it will print an error in the development mode and will not show errors in production mode.

navigation.push('RouteName'): Used to navigate to a screen in the stack navigator, adding a new route to the navigation regardless of the existing navigation history.

navigation.goBack(): Is used to programmatically go back to the previous screen.

navigation.popToTop(): Used to go back to the first screen in the stack.

Drawer Navigator

Drawer Navigator (cont)

```
import * as React from 'react';
import { View, Text} from 'react-native';
import { Naviga tio nCo ntainer } from '@react-naviga
import { create Dra wer Nav igator } from '@react-nav
function Custom Dra wer Con tent() {
return (
<DrawerContentScrollView {...props}>
    <Te xt> Welcome </Text>
    <Dr awe rIt emList {...props} />
    <Dr awe rItem label= " Hel p" onPres s={() => ale
</DrawerContentScrollView>
 );
}
function HomeSc reen() {   // ... }
function Notifi cat ion sSc reen() {   // ... }
const Drawer = createDrawerNavigator();
function App() {
return (
<NavigationContainer>
  <Dr awe r.N avi gator  initialRouteName="Home"
                   screenOptions={{drawerPosition:
              drawer Con ten t={ props => <Cu sto mD
     <Dr awe r.S creen name="H ome " compon ent ={H
     <Dr awe r.S creen name="N oti fic ati ons " comp
wer.Navigator>
</NavigationContainer>
    );
}
export default App;
```

The following are also available:

navigation. jumpTo('RouteName'): go to a specific screen in the drav

navigation. openDrawer: open the drawer.

navigation.closeDrawer: close the drawer.

navigation.toggleDrawer: toggle the state, ie. switch from closed to c

Tab Navigator

^ Renders a navigation drawer on the side of the screen which can be opened and closed via gestures.

^ You cannot use the useNavigation hook inside the drawerContent since useNavigation is only available inside screens. You get a navigation prop for your drawerContent which you can use instead.

^ **drawerPosition:** prop typically set in the screenOptions to specify the position of the drawer, such as left or right.

^ **drawerContent:** prop in the Drawer Navigator that allows you to provide a custom component for the drawer's content.

^ **CustomDrawerContent:** refer to a user-defined React component that is passed to the drawerContent prop.

^ **DrawerItem:** in a custom drawer allows for more flexibility and customization compared to defining routes directly in the navigator.

^ Common style of navigation.

^ Can be tabs on the bottom of the screen or on the top below the hea

^ **Bottom tab navigation:** A simple tab bar on the bottom of the screen different routes.

^ Routes are lazily initialized -- their screen components are not mour

^ You cannot use the useNavigation hook inside the tabBar since use screens. You get a **navigation prop** for your tabBar which you can use i

```
import React from 'react';
import { View, Text } from 'react-native';
import { Naviga tio nCo ntainer } from '@react-naviga
import { create Bot tom Tab Nav igator } from '@react
;
import Ionicons from 'react -na tiv e-v ect or- ico
const HomeScreen = () => {
return(
<View>
  <Te xt>Home Screen</Text>
</View>
  )
};
const Settin gsS creen = () => {
return(
  <View>
  <Te xt> Set tings Screen</Text></View>
  )
}
const Tab = createBottomTabNavigator();
function App() {
return (
<NavigationContainer>
 <Ta b.N avi gator  screen Opt ion s={({ route }) =>
    tabBar Icon: ({ focused, color, size }) => {
     let iconName;
     if (route.name === 'Home') {
     iconName = focused ? 'ios-home' : 'ios-home-outl
     } else if (route.name === 'Setti ngs') {
     iconName = focused ? 'ios-s ett ings' : 'ios-s e
     return <Io nicons name={ ico nName} size={ size
)} >
   <Ta b.S creen name="H ome " compon ent ={H ome Sc
   <Ta b.S creen name="S ett ing s" compon ent ={S et
 </Tab.Navigator>
</NavigationContainer>
    );
}
export default App;
```

Tab Navigator (cont)

The following are also available:

navigation.jumpTo('RouteName'): is a method that directly switches to a specified screen within the tab navigator.

View

- A container that supports layout with flexbox, style, some touch handling, and accessibility controls.
- Like a <div> in HTML.
- Designed to be nested inside other views and can have 0 to many children of any type.

```
import React from 'react';
import { View, Text } from 'react -na tive';
const Exampl eView = () => {
return (
<View style={{ flex: 1, justif yCo ntent: 'center', alignI tems: 'center' }
}>
   <Te xt> Hello from View!< /Te xt>
</View>
   );
 };
export default Exampl eView;
```

Text

ScrollView (cont)

```
import React from 'react';
import { Scroll View, Text, View } from 'react-native
const Exampl eSc rol lView = () => {
return (
<ScrollView indica tor Sty le= {"wh ite "}
            style={{ flex: 1 }}
            horizo nta l={ tru e}> {/ horizontal scr
<Te xt>It em 1</ Tex t> {/ Repeat more components f

</ScrollView>
   );
 };
export default Exampl eSc rol lView;
```

Performance Issues with Large Lists: Slow rendering times for large lists.

Memory Consumption: Consume a significant amount of memory with large lists or complex item views.

FlatList

- Used to efficiently render long lists.
- Offers features like pull-to-refresh, infinite scrolling, and easy item se
- **Lazy rendering:** renders items only when they appear on the screen user scrolls away from them.
- Internal state is not preserved when content scrolls out of the render
- Inherits the props of the ScrollView component.

```
import React from 'react';
import { FlatList, Text, View } from 'react-native';
const Exampl eFl atList = () => {
const data = [{ id: '1', name: 'Item 1' }, { id: '2',
return (
    <Fl atList data={data}
       render Ite m={({ item }) => <Text>{item.name}
              keyExt rac tor ={item => item.id} />
 );
};
export default Exampl eFl atList;
```

Two required props:

data: accepts a plain array that contains the list of items to display.

renderItem: a function that goes over each item in the array and ren

keyExtractor: It instructs the list to use the id of each item as React keys property.

SectionList

A component for displaying text.
Supports nesting, styling, and touch handling.
Everything inside it is no longer using the Flexbox layout but using text layout.
Elements inside it are no longer rectangles, but wrap at the end of the line.

```
import React from 'react';
import { Text } from 'react-native';
const Exampl eText = () => {
return (
<Text style={{ fontSize: 18, color: 'blue' }}>
     Hello, this is a Text component!
</Text>
            );
          };
export default Exampl eText;
```

You must wrap all the text nodes inside of a <Text> component

Will raise exception

```
<Vi ew> Some text </V iew>
```

Correct

```
<View>
<Text> Some text </Text>
</View>
```

Text container: Text will be inline if the space allow it, otherwise, text will flow as if it was one.

```
<Text>
<Text>First part and </Text>
<Text>second part</Text>
</Text>
```

First part and second part

View container:
Each text is its own block, otherwise, the text will flow in its own block.

```
<View>
<Text>First part and </Text>
<Text>second part</Text>
</View>
```

First part and
second part

ScrollView

Creates a scrollable area when content exceeds screen's physical limits.
Can contain multiple components and views.
Can be scrolled vertically or horizontally.
Must have a bounded height in order to work.
Renders all its react child components at once.

Used for rendering large lists with section headers.
Uses **lazy rendering** to achieve faster rendering.
Inherits the props of the ScrollView component.
Internal state is not preserved when content scrolls out of the render
Provides support for section headers and section separators.

```
import React from 'react';
import { Sectio nList, Text, View } from 'react-nativ
const Exampl eSe cti onList = () => {
const sections = [ { title: 'Section 1', data: ['Iter
                   { title: 'Section 2', data: ['Iter
return (
<SectionList
       sectio ns= {se ctions}
       render Ite m={({ item }) => <Te xt> {it em} <,
       render Sec tio nHe ade r={({ section }) => <Te
} </T ext >}
       keyExt rac tor ={( item, index) => item + inde
);
};
export default Exampl eSe cti onList;
```

SectionList (cont)

Two required props:
sections : accepts the array that contains the list of items to display, akin to the data prop in FlatList.
renderItem: method which acts as the default renderer for every item in each section.
renderSectionHeader: prop, render each section's header.

TextInput

Used for inputting text into the app via a keyboard.

```
import React, { useState } from 'react';
import { TextInput } from 'react-native';
const Exampl eTe xtInput = () => {
const [input Value, setInp utV alue] = useState('');
return (
<TextInput value= {in put Value}
    onChan geT ext ={text => setInp utV alu e(t ext)}
           placeh old er= " Enter text here"
    style={{ height: 40, border Width: 1, margin: 10 }} /
>
 );
};
export default Exampl eTe xtI nput;
```

Image

Used for displaying different types of images,
network images, static resources, temporary local images,
and images from local disk, such as the camera roll.
You can also add style to an image.

```
import React from 'react';
import { Image } from 'react-native';
const Exampl eImage = () => {
return (
  <>
{/ Remote Image /}
<Image source={{ uri: 'https :// exa mpl e.c om/ ima ge.j
pg' }}
       style={{ width: 200, height: 200 }}
       resize Mod e="c ont ain " />
{/ Local Image  /}
<Image source={require('./path-to-your-local-image.png')}
       style={{ width: 200, height: 200 }}
       resize Mod e="c ove r" />
   </>
   );
};
export default Exampl eImage;
```

resizeMode :
'cover': Scales image to fill the container, maintaining its aspect ratio.
'contain': Scales image to fit inside the container, maintain the image's aspect ratio ensuring the entire image is visible.
'stretch': Stretches image to fill the container, possibly distorting the aspect ratio.
'center': Centers image in the container without scaling. 'repeat': Repeats the image to cover the container.

For network and data images, you must specify the dimensions of the image.