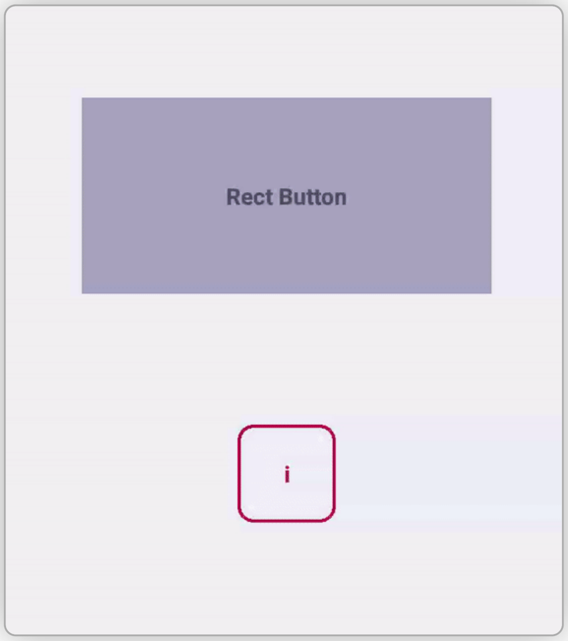
Version: 2.6.0 – 2.12.0

Buttons



Gesture handler library provides native components that can act as buttons. These can be treated as a replacement to `TouchableHighlight` or `TouchableOpacity` from RN core. Gesture handler's buttons recognize touches in native which makes the recognition process deterministic, allows for rendering ripples on Android in highly performant way (`TouchableNativeFeedback` requires that touch event does a roundtrip to JS before we can update ripple effect, which makes ripples lag a bit on older phones), and provides native and platform default interaction for buttons that are placed in a scrollable container (in which case the interaction is slightly delayed to prevent button from highlighting when you fling).

Currently Gesture handler library exposes three components that render native touchable elements under the hood:

- `BaseButton`
- `RectButton`
- `BorderlessButton`

On top of that all the buttons are wrapped with `NativeViewGestureHandler` and therefore allow for all the common gesture handler properties and `NativeViewGestureHandler`'s extra properties to be applied to them.

IMPORTANT: In order to make buttons accessible, you have to wrap your children in a `View` with `accessible` and `accessibilityRole="button"` props. Example:

```
// Not accessible:
const NotAccessibleButton = () => (
  <RectButton onPress={this._onPress}>
    <Text>Foo</Text>
  </RectButton>
);
// Accessible:
const AccessibleButton = () => (
  <RectButton onPress={this._onPress}>
    <View accessible accessibilityRole="button">
      <Text>Bar</Text>
    </View>
  </RectButton>
);
```

It is applicable for both iOS and Android platform. On iOS, you won't be able to even select the button, on Android you won't be able to click it in accessibility mode.

`BaseButton`

Can be used as a base class if you'd like to implement some custom interaction for when the button is pressed.

Below is a list of properties specific to `BaseButton` component:

`onActiveStateChange`

function that gets triggered when button changes from inactive to active and vice versa. It passes active state as a boolean variable as a first parameter for that method.

`onPress`

function that gets triggered when the button gets pressed (analogous to `onPress` in `TouchableHighlight` from RN core).

`onLongPress`

function that gets triggered when the button gets pressed for at least `delayLongPress` milliseconds.

`rippleColor` (Android only)

defines color of native ripple animation used since API level 21.

`exclusive`

defines if more than one button could be pressed simultaneously. By default set `true`.

`delayLongPress`

defines the delay, in milliseconds, after which the `onLongPress` callback gets called. By default set to 600.

`RectButton`

This type of button component should be used when you deal with rectangular elements or blocks of content that can be pressed, for example table rows or buttons with text and icons. This component provides a platform specific interaction, rendering a rectangular ripple on Android or highlighting the background on iOS and on older versions of Android. In addition to the props of `BaseButton`, it accepts the following:

Below is a list of properties specific to `RectButton` component:

`underlayColor`

this is the background color that will be dimmed when button is in active state.

`activeOpacity` (iOS only)

opacity applied to the underlay when button is in active state.

`BorderlessButton`

This type of button component should be used with simple icon-only or text-only buttons. The interaction will be different depending on platform: on Android a borderless ripple will be rendered (it means that the ripple will animate into a circle that can span outside of the view bounds), whereas on iOS the button will be dimmed (similar to how `TouchableOpacity` works). In addition to the props of `BaseButton`, it accepts the following:

Below is a list of properties specific to `BorderlessButton` component:

`borderless` (Android only)

set this to `false` if you want the ripple animation to render only within view bounds.

`activeOpacity` (iOS only)

opacity applied to the button when it is in an active state.

Design patterns

Components listed here were not designed to behave and look in the same way on both platforms but rather to be used for handling similar behaviour on iOS and Android taking into consideration their's design concepts.

If you wish to get specific information about platforms design patterns, visit official Apple docs and Material.io guideline, which widely describe how to implement coherent design.

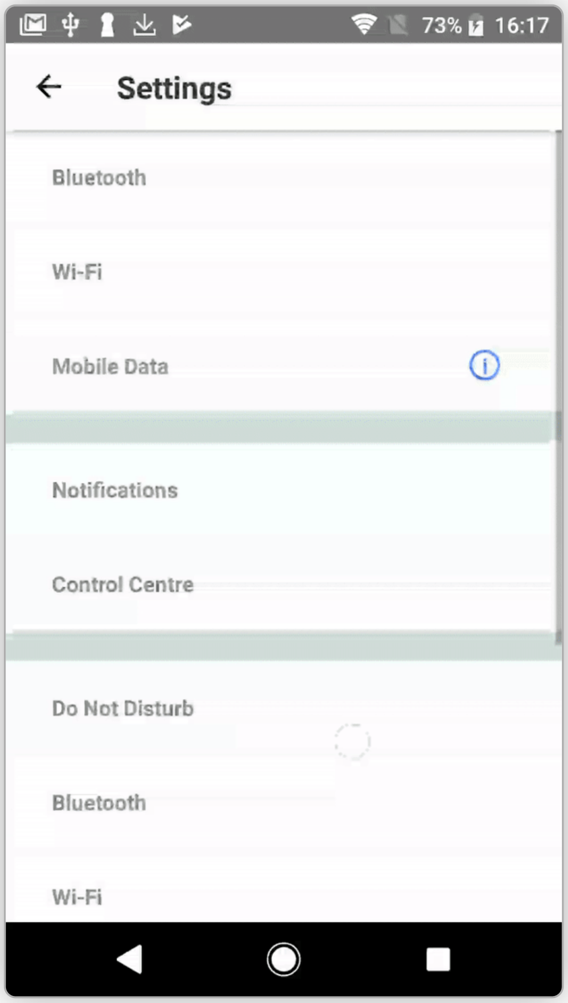
This library allows to use native components with native feedback in adequate situations.

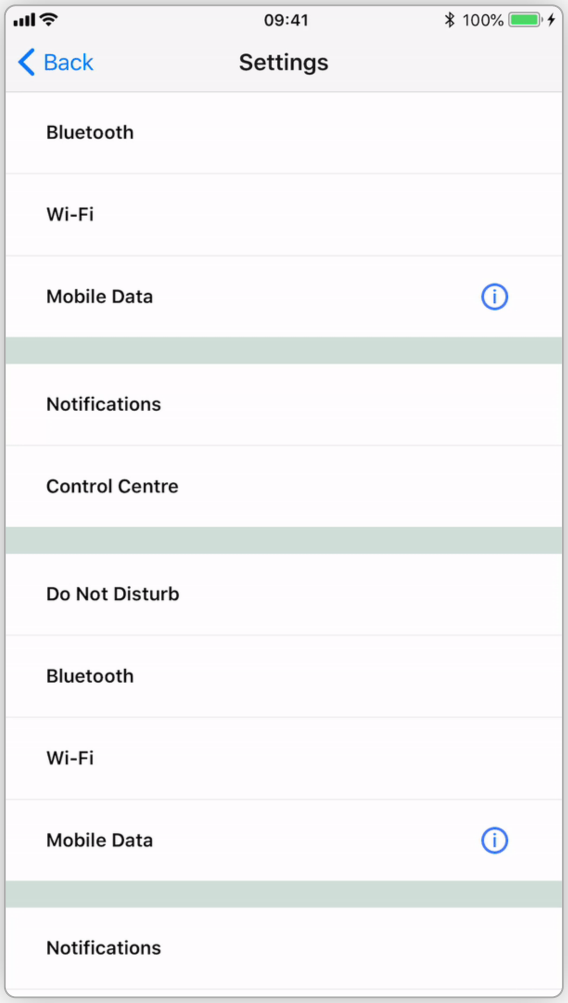
If you do not wish to implement custom design approach, `RectButton` and `BorderlessButton` seem to be absolutely enough and there's no need to use anything else. In all the remaining cases you can always rely on `BaseButton` which is a superclass for the other button classes and can be used as a generic `Touchable` replacement that can be customized to your needs.

Below we list some of the common usecases for button components to be used along with the type of button that should be used according to the platform specific design guidelines.

Lists and action buttons

If you have a list with clickable items or have an action button that need to display as a separate UI block (vs being inlined in a text) you should use `RectButton`. It changes opacity on click and additionally supports a ripple effect on Android.

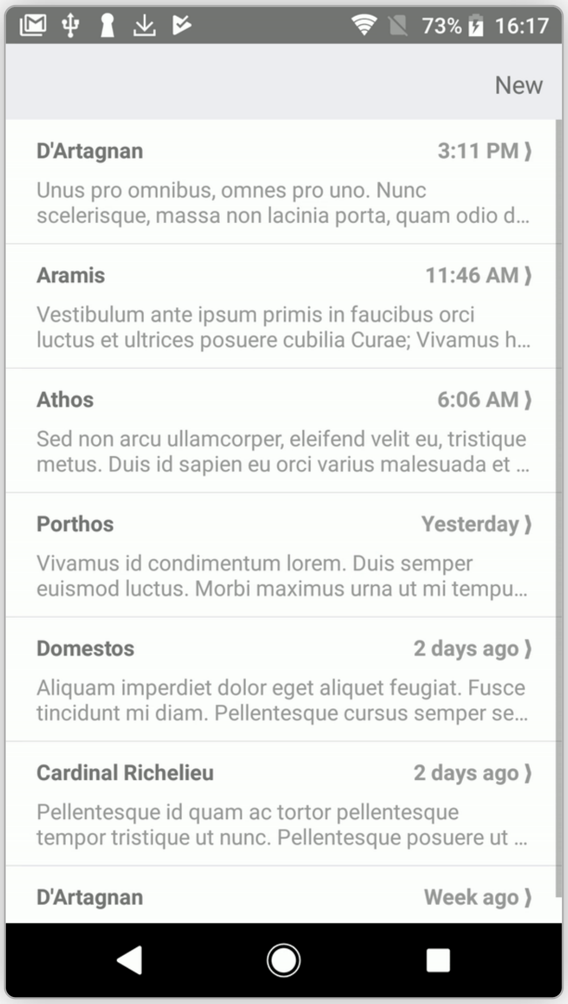




To determine emphasis of button it's vital to use fill color or leave it transparent especially on Android. For medium emphasis you may consider outlined buttons which are used for lower impact than fill buttons.

Icon or text only buttons

Use `BorderlessButton` for simple icon-only or text-only buttons. The interaction will be different depending on platform: on Android a borderless ripple will be rendered, whereas on iOS the button will be dimmed. It should be used if you wish to handle non-crucial actions and supportive behaviour.



D'Artagnan 3:11 PM ❯
Unus pro omnibus, omnes pro uno. Nunc scelerisque, massa non lacinia porta, quam odio d...

Aramis 11:46 AM ❯
Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Vivamus h...

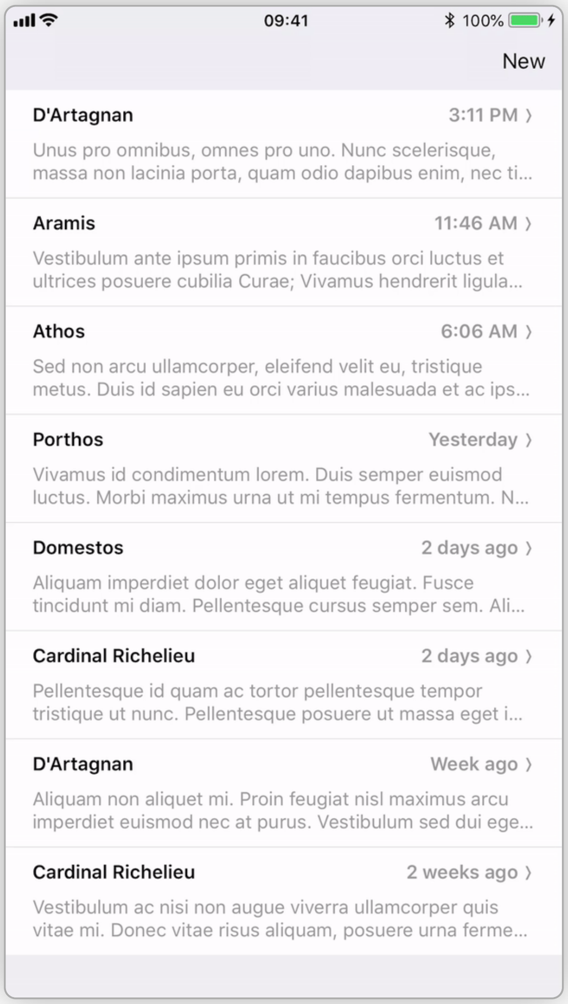
Athos 6:06 AM ❯
Sed non arcu ullamcorper, eleifend velit eu, tristique metus. Duis id sapien eu orci varius malesuada et ...

Porthos Yesterday ❯
Vivamus id condimentum lorem. Duis semper euismod luctus. Morbi maximus urna ut mi tempu...

Domestos 2 days ago ❯
Aliquam imperdiet dolor eget aliquet feugiat. Fusce tincidunt mi diam. Pellentesque cursus semper se...

Cardinal Richelieu 2 days ago ❯
Pellentesque id quam ac tortor pellentesque tempor tristique ut nunc. Pellentesque posuere ut ...

D'Artagnan Week ago ❯



D'Artagnan 3:11 PM ❯
Unus pro omnibus, omnes pro uno. Nunc scelerisque, massa non lacinia porta, quam odio dapibus enim, nec ti...

Aramis 11:46 AM ❯
Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Vivamus hendrerit ligula...

Athos 6:06 AM ❯
Sed non arcu ullamcorper, eleifend velit eu, tristique metus. Duis id sapien eu orci varius malesuada et ac ips...

Porthos Yesterday ❯
Vivamus id condimentum lorem. Duis semper euismod luctus. Morbi maximus urna ut mi tempus fermentum. N...

Domestos 2 days ago ❯
Aliquam imperdiet dolor eget aliquet feugiat. Fusce tincidunt mi diam. Pellentesque cursus semper sem. Ali...

Cardinal Richelieu 2 days ago ❯
Pellentesque id quam ac tortor pellentesque tempor tristique ut nunc. Pellentesque posuere ut massa eget i...

D'Artagnan Week ago ❯
Aliquam non aliquet mi. Proin feugiat nisl maximus arcu imperdiet euismod nec at purus. Vestibulum sed dui ege...

Cardinal Richelieu 2 weeks ago ❯
Vestibulum ac nisi non augue viverra ullamcorper quis vitae mi. Donec vitae risus aliquam, posuere urna ferme...

`PureNativeButton`

Use a `PureNativeButton` for accessing the native Component used for build more complex buttons listed above. It's normally is not recommended to use, but it might be useful if we want to wrap it using Animated or Reanimated.

```
import {
  createNativeWrapper,
  PureNativeButton,
} from 'react-native-gesture-handler';
import Animated from 'react-native-reanimated';
const { event, Value, createAnimatedComponent } = Animated;

const AnimatedRawButton = createNativeWrapper(
  createAnimatedComponent(PureNativeButton),
  {
    shouldCancelWhenOutside: false,
    shouldActivateOnStart: false,
  }
);

export default class App extends React.Component {
  constructor(props) {
    super(props);
    const state = new Value();
    this._onGestureEvent = event([
      {
        nativeEvent: { state },
      },
    ]);
  }

  render() {
    return <AnimatedRawButton onHandlerStateChange={this._onGestureEvent} />;
  }
}
```