View

The most fundamental component for building a UI, `View` is a container that supports layout with flexbox, style, some touch handling, and accessibility controls. `View` maps directly to the native view equivalent on whatever platform React Native is running on, whether that is a `UIView`, `<div>`, `android.view`, etc.

`View` is designed to be nested inside other views and can have 0 to many children of any type.

This example creates a `View` that wraps two boxes with color and a text component in a row with padding.

View Example

Expo

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

const ViewBoxesWithColorAndText = () => {
  return (
    <View
      style={{
        flexDirection: 'row',
        height: 100,
        padding: 20,
      }}>
      <View style={{backgroundColor: 'blue', flex: 0.3}} />
      <View style={{backgroundColor: 'red', flex: 0.5}} />
      <Text>Hello World!</Text>
    </View>
  );
};

export default ViewBoxesWithColorAndText;
```

Preview My Device iOS Android Web

> `View`s are designed to be used with `StyleSheet` for clarity and performance, although inline styles are also supported.

Synthetic Touch Events

For `View` responder props (e.g., `onResponderMove`), the synthetic touch event passed to them are in form of PressEvent.

Reference

Props

`accessibilityActions`

Accessibility actions allow an assistive technology to programmatically invoke the actions of a component. The `accessibilityActions` property should contain a list of action objects. Each action object should contain the field name and label.

See the Accessibility guide for more information.

TYPE
array

`accessibilityElementsHidden` ◀ iOS

A value indicating whether the accessibility elements contained within this accessibility element are hidden. Default is `false`.

See the Accessibility guide for more information.

TYPE
bool

`accessibilityHint`

An accessibility hint helps users understand what will happen when they perform an action on the accessibility element when that result is not clear from the accessibility label.

TYPE
string

`accessibilityLanguage` iOS

A value indicating which language should be used by the screen reader when the user interacts with the element. It should follow the BCP 47 specification.

See the iOS `accessibilityLanguage` doc for more information.

TYPE
string

`accessibilityIgnoresInvertColors` iOS

A value indicating this view should or should not be inverted when color inversion is turned on. A value of `true` will tell the view to not be inverted even if color inversion is turned on.

See the Accessibility guide for more information.

TYPE
bool

`accessibilityLabel`

Overrides the text that's read by the screen reader when the user interacts with the element. By default, the label is constructed by traversing all the children and accumulating all the `Text` nodes separated by space.

TYPE
string

`accessibilityLiveRegion` Android

Indicates to accessibility services whether the user should be notified when this view changes. Works for Android API >= 19 only. Possible values:

- `'none'` - Accessibility services should not announce changes to this view.
- `'polite'` - Accessibility services should announce changes to this view.
- `'assertive'` - Accessibility services should interrupt ongoing speech to immediately announce changes to this view.

See the Android `View` docs for reference.

TYPE
enum('none', 'polite', 'assertive')

`accessibilityRole`

`accessibilityRole` communicates the purpose of a component to the user of an assistive technology.

`accessibilityRole` can be one of the following:

- `'none'` - Used when the element has no role.
- `'button'` - Used when the element should be treated as a button.
- `'link'` - Used when the element should be treated as a link.
- `'search'` - Used when the text field element should also be treated as a search field.
- `'image'` - Used when the element should be treated as an image. Can be combined with button or link, for example.
- `'keyboardkey'` - Used when the element acts as a keyboard key.
- `'text'` - Used when the element should be treated as static text that cannot change.
- `'adjustable'` - Used when an element can be "adjusted" (e.g. a slider).
- `'imagebutton'` - Used when the element should be treated as a button and is also an image.
- `'header'` - Used when an element acts as a header for a content section (e.g. the title of a navigation bar).
- `'summary'` - Used when an element can be used to provide a quick summary of current conditions in the app when the app first launches.
- `'alert'` - Used when an element contains important text to be presented to the user.
- `'checkbox'` - Used when an element represents a checkbox which can be checked, unchecked, or have mixed checked state.
- `'combobox'` - Used when an element represents a combo box, which allows the user to select among several choices.
- `'menu'` - Used when the component is a menu of choices.
- `'menubar'` - Used when a component is a container of multiple menus.
- `'menuitem'` - Used to represent an item within a menu.
- `'progressbar'` - Used to represent a component which indicates progress of a task.
- `'radio'` - Used to represent a radio button.
- `'radiogroup'` - Used to represent a group of radio buttons.
- `'scrollbar'` - Used to represent a scroll bar.
- `'spinbutton'` - Used to represent a button which opens a list of choices.
- `'switch'` - Used to represent a switch which can be turned on and off.

- `'tab'` - Used to represent a tab.
- `'tablist'` - Used to represent a list of tabs.
- `'timer'` - Used to represent a timer.
- `'toolbar'` - Used to represent a tool bar (a container of action buttons or components).
- `'grid'` - Used with ScrollView, VirtualizedList, FlatList, or SectionList to represent a grid. Adds the in/out of grid announcements to the android GridView.

TYPE
string

`accessibilityState`

Describes the current state of a component to the user of an assistive technology.

See the Accessibility guide for more information.

TYPE
object: `{disabled: bool, selected: bool, checked: bool or 'mixed', busy: bool, expanded: bool}`

`accessibilityValue`

Represents the current value of a component. It can be a textual description of a component's value, or for range-based components, such as sliders and progress bars, it contains range information (minimum, current, and maximum).

See the Accessibility guide for more information.

TYPE
object: `{min: number, max: number, now: number, text: string}`

`accessibilityViewIsModal` iOS

A value indicating whether VoiceOver should ignore the elements within views that are siblings of the receiver. Default is `false`.

See the Accessibility guide for more information.

TYPE
bool

`accessible`

When `true`, indicates that the view is an accessibility element. By default, all the touchable elements are accessible.

`aria-busy`

Indicates an element is being modified and that assistive technologies may want to wait until the changes are complete before informing the user about the update.

TYPE	DEFAULT
boolean	false

`aria-checked`

Indicates the state of a checkable element. This field can either take a boolean or the "mixed" string to represent mixed checkboxes.

TYPE	DEFAULT
boolean, 'mixed'	false

`aria-disabled`

Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable.

TYPE	DEFAULT
boolean	false

`aria-expanded`

Indicates whether an expandable element is currently expanded or collapsed.

TYPE	DEFAULT
boolean	false

`aria-hidden`

Indicates whether the accessibility elements contained within this accessibility element are hidden.

For example, in a window that contains sibling views `A` and `B`, setting `aria-hidden` to `true` on view `B` causes VoiceOver to ignore the elements in the view `B`.

TYPE	DEFAULT
boolean	false

`aria-label`

Defines a string value that labels an interactive element.

TYPE
string

aria-labelledby Android

Identifies the element that labels the element it is applied to. The value of `aria-labelledby` should match the `nativeID` of the related element:

```
<View>
  <Text nativeID="formLabel">Label for Input Field</Text>
  <TextInput aria-label="input" aria-labelledby="formLabel" />
</View>
```

TYPE
string

aria-live Android

Indicates that an element will be updated, and describes the types of updates the user agents, assistive technologies, and user can expect from the live region.

- **off** Accessibility services should not announce changes to this view.
- **polite** Accessibility services should announce changes to this view.
- **assertive** Accessibility services should interrupt ongoing speech to immediately announce changes to this view.

TYPE	DEFAULT
enum(`'assertive'`, `'off'`, `'polite'`)	`'off'`

aria-modal iOS

Boolean value indicating whether VoiceOver should ignore the elements within views that are siblings of the receiver. Has precedence over the `accessibilityViewIsModal` prop.

TYPE	DEFAULT
boolean	false

`aria-selected`

Indicates whether a selectable element is currently selected or not.

TYPE
boolean

`aria-valuemax`

Represents the maximum value for range-based components, such as sliders and progress bars. Has precedence over the `max` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuemin`

Represents the minimum value for range-based components, such as sliders and progress bars. Has precedence over the `min` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuenow`

Represents the current value for range-based components, such as sliders and progress bars. Has precedence over the `now` value in the `accessibilityValue` prop.

TYPE
number

`aria-valuetext`

Represents the textual description of the component. Has precedence over the `text` value in the `accessibilityValue` prop.

TYPE
string

`collapsable` Android

Views that are only used to layout their children or otherwise don't draw anything may be automatically removed from the native hierarchy as an optimization. Set this property to `false` to disable this optimization and ensure that this `View` exists in the native view hierarchy.

TYPE
bool

`focusable` Android

Whether this `View` should be focusable with a non-touch input device, eg. receive focus with a hardware keyboard.

TYPE
boolean

`hitSlop`

This defines how far a touch event can start away from the view. Typical interface guidelines recommend touch targets that are at least 30 - 40 points/density-independent pixels.

For example, if a touchable view has a height of 20 the touchable height can be extended to 40 with `hitSlop={{top: 10, bottom: 10, left: 0, right: 0}}`

> The touch area never extends past the parent view bounds and the Z-index of sibling views always takes precedence if a touch hits two overlapping views.

TYPE
object: `{top: number, left: number, bottom: number, right: number}`

`id`

Used to locate this view from native classes. Has precedence over `nativeID` prop.

> This disables the 'layout-only view removal' optimization for this view!

TYPE
string

`importantForAccessibility` Android

Controls how view is important for accessibility which is if it fires accessibility events and if it is reported to accessibility services that query the screen. Works for Android only.

Possible values:

- `'auto'` - The system determines whether the view is important for accessibility - default (recommended).
- `'yes'` - The view is important for accessibility.
- `'no'` - The view is not important for accessibility.
- `'no-hide-descendants'` - The view is not important for accessibility, nor are any of its descendant views.

See the Android `importantForAccessibility` docs for reference.

TYPE
enum('auto', 'yes', 'no', 'no-hide-descendants')

`nativeID`

Used to locate this view from native classes.

> This disables the 'layout-only view removal' optimization for this view!

TYPE
string

`needsOffscreenAlphaCompositing`

Whether this `View` needs to rendered offscreen and composited with an alpha in order to preserve 100% correct colors and blending behavior. The default (`false`) falls back to drawing the component and its children with an alpha applied to the paint used to draw each element instead of rendering the full component offscreen and compositing it back

with an alpha value. This default may be noticeable and undesired in the case where the `View` you are setting an opacity on has multiple overlapping elements (e.g. multiple overlapping `View`s, or text and a background).

Rendering offscreen to preserve correct alpha behavior is extremely expensive and hard to debug for non-native developers, which is why it is not turned on by default. If you do need to enable this property for an animation, consider combining it with renderToHardwareTextureAndroid if the view **contents** are static (i.e. it doesn't need to be redrawn each frame). If that property is enabled, this View will be rendered off-screen once, saved in a hardware texture, and then composited onto the screen with an alpha each frame without having to switch rendering targets on the GPU.

TYPE
bool

`nextFocusDown` Android

Designates the next view to receive focus when the user navigates down. See the Android documentation.

TYPE
number

`nextFocusForward` Android

Designates the next view to receive focus when the user navigates forward. See the Android documentation.

TYPE
number

`nextFocusLeft` Android

Designates the next view to receive focus when the user navigates left. See the Android documentation.

TYPE
number

`nextFocusRight` Android

Designates the next view to receive focus when the user navigates right. See the Android documentation.

TYPE
number

`nextFocusUp` Android

Designates the next view to receive focus when the user navigates up. See the Android documentation.

TYPE
number

`onAccessibilityAction`

Invoked when the user performs the accessibility actions. The only argument to this function is an event containing the name of the action to perform.

See the Accessibility guide for more information.

TYPE
function

`onAccessibilityEscape` iOS

When `accessible` is `true`, the system will invoke this function when the user performs the escape gesture.

TYPE
function

`onAccessibilityTap`

When `accessible` is true, the system will try to invoke this function when the user performs accessibility tap gesture.

TYPE
function

`onLayout`

Invoked on mount and on layout changes.

This event is fired immediately once the layout has been calculated, but the new layout may not yet be reflected on the screen at the time the event is received, especially if a layout animation is in progress.

TYPE
`({nativeEvent: LayoutEvent}) => void`

`onMagicTap` iOS

When `accessible` is `true`, the system will invoke this function when the user performs the magic tap gesture.

TYPE
function

`onMoveShouldSetResponder`

Does this view want to "claim" touch responsiveness? This is called for every touch move on the `View` when it is not the responder.

TYPE
`({nativeEvent: PressEvent}) => boolean`

`onMoveShouldSetResponderCapture`

If a parent `View` wants to prevent a child `View` from becoming responder on a move, it should have this handler which returns `true`.

TYPE
`({nativeEvent: PressEvent}) => boolean`

`onResponderGrant`

The View is now responding for touch events. This is the time to highlight and show the user what is happening.

On Android, return true from this callback to prevent any other native components from becoming responder until this responder terminates.

TYPE
({nativeEvent: PressEvent}) => void | boolean

onResponderMove

The user is moving their finger.

TYPE
({nativeEvent: PressEvent}) => void

onResponderReject

Another responder is already active and will not release it to that `View` asking to be the responder.

TYPE
({nativeEvent: PressEvent}) => void

onResponderRelease

Fired at the end of the touch.

TYPE
({nativeEvent: PressEvent}) => void

onResponderTerminate

The responder has been taken from the `View`. Might be taken by other views after a call to `onResponderTerminationRequest`, or might be taken by the OS without asking (e.g.,

happens with control center/ notification center on iOS)

TYPE
`({nativeEvent: PressEvent}) => void`

`onResponderTerminationRequest`

Some other `View` wants to become responder and is asking this `View` to release its responder. Returning `true` allows its release.

TYPE
`({nativeEvent: PressEvent}) => void`

`onStartShouldSetResponder`

Does this view want to become responder on the start of a touch?

TYPE
`({nativeEvent: PressEvent}) => boolean`

`onStartShouldSetResponderCapture`

If a parent `View` wants to prevent a child `View` from becoming responder on a touch start, it should have this handler which returns `true`.

TYPE
`({nativeEvent: PressEvent}) => boolean`

`pointerEvents`

Controls whether the `View` can be the target of touch events.

- `'auto'`: The View can be the target of touch events.
- `'none'`: The View is never the target of touch events.
- `'box-none'`: The View is never the target of touch events but its subviews can be. It behaves like if the view had the following classes in CSS:

```
.box-none {
     pointer-events: none;
}
.box-none * {
     pointer-events: auto;
}
```

- `'box-only'`: The view can be the target of touch events but its subviews cannot be. It behaves like if the view had the following classes in CSS:

```
.box-only {
     pointer-events: auto;
}
.box-only * {
     pointer-events: none;
}
```

TYPE
enum('box-none', 'none', 'box-only', 'auto')

`removeClippedSubviews`

This is a reserved performance property exposed by `RCTView` and is useful for scrolling content when there are many subviews, most of which are offscreen. For this property to be effective, it must be applied to a view that contains many subviews that extend outside its bound. The subviews must also have `overflow: hidden`, as should the containing view (or one of its superviews).

TYPE
bool

`renderToHardwareTextureAndroid` Android

Whether this `View` should render itself (and all of its children) into a single hardware texture on the GPU.

On Android, this is useful for animations and interactions that only modify opacity, rotation, translation, and/or scale: in those cases, the view doesn't have to be redrawn and display lists don't need to be re-executed. The texture can be re-used and re-composited with different parameters. The downside is that this can use up limited video memory, so this prop should be set back to false at the end of the interaction/animation.

TYPE
bool

`role`

`role` communicates the purpose of a component to the user of an assistive technology. Has precedence over the `accessibilityRole` prop.

TYPE
Role

`shouldRasterizeIOS` iOS

Whether this `View` should be rendered as a bitmap before compositing.

On iOS, this is useful for animations and interactions that do not modify this component's dimensions nor its children; for example, when translating the position of a static view,

rasterization allows the renderer to reuse a cached bitmap of a static view and quickly composite it during each frame.

Rasterization incurs an off-screen drawing pass and the bitmap consumes memory. Test and measure when using this property.

TYPE
bool

`style`

TYPE
View Style

`tabIndex` Android

Whether this `View` should be focusable with a non-touch input device, eg. receive focus with a hardware keyboard. Supports the following values:

- `0` - View is focusable
- `-1` - View is not focusable

TYPE
enum(0, -1)

`testID`

Used to locate this view in end-to-end tests.

> This disables the 'layout-only view removal' optimization for this view!

TYPE
string

Is this page useful? 

Edit this page

*Last updated on **Aug 17, 2023***