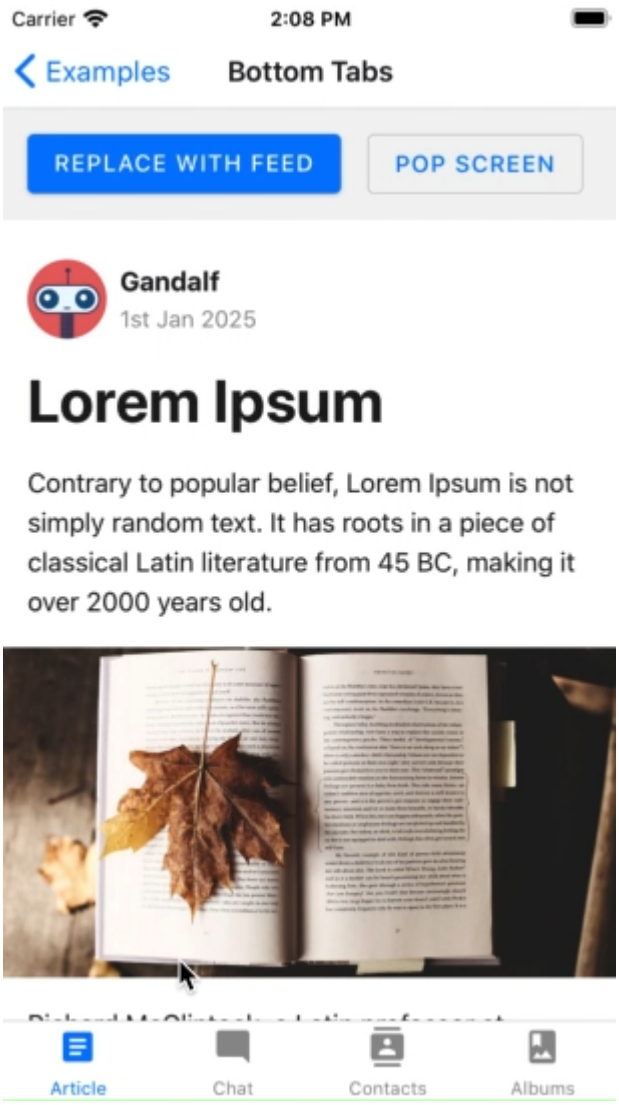
Version: 6.x

Bottom Tabs Navigator

A simple tab bar on the bottom of the screen that lets you switch between different routes. Routes are lazily initialized -- their screen components are not mounted until they are first focused.



Installation

To use this navigator, ensure that you have `@react-navigation/native` and its dependencies (follow this guide), then install `@react-navigation/bottom-tabs`:

npm Yarn

```
npm install @react-navigation/bottom-tabs
```

API Definition

To use this tab navigator, import it from `@react-navigation/bottom-tabs`:

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Tab = createBottomTabNavigator();

function MyTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Home" component={HomeScreen} />
      <Tab.Screen name="Settings" component={SettingsScreen} />
    </Tab.Navigator>
  );
}
```

Try this example on Snack

> For a complete usage guide please visit Tab Navigation

Props

The `Tab.Navigator` component accepts following props:

`id`

Optional unique ID for the navigator. This can be used with `navigation.getParent` to refer to this navigator in a child navigator.

`initialRouteName`

The name of the route to render on first load of the navigator.

`screenOptions`

Default options to use for the screens in the navigator.

`backBehavior`

This controls what happens when `goBack` is called in the navigator. This includes pressing the device's back button or back gesture on Android.

It supports the following values:

- `firstRoute` - return to the first screen defined in the navigator (default)
- `initialRoute` - return to initial screen passed in `initialRouteName` prop, if not passed, defaults to the first screen
- `order` - return to screen defined before the focused screen
- `history` - return to last visited screen in the navigator; if the same screen is visited multiple times, the older entries are dropped from the history
- `none` - do not handle back button

`detachInactiveScreens`

Boolean used to indicate whether inactive screens should be detached from the view hierarchy to save memory. This enables integration with react-native-screens. Defaults to `true`.

`sceneContainerStyle`

Style object for the component wrapping the screen content.

`tabBar`

Function that returns a React element to display as the tab bar.

Example:

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

function MyTabBar({ state, descriptors, navigation }) {
  return (
    <View style={{ flexDirection: 'row' }}>
```

```
        {state.routes.map((route, index) => {
          const { options } = descriptors[route.key];
          const label =
            options.tabBarLabel !== undefined
              ? options.tabBarLabel
              : options.title !== undefined
              ? options.title
              : route.name;

          const isFocused = state.index === index;

          const onPress = () => {
            const event = navigation.emit({
              type: 'tabPress',
              target: route.key,
              canPreventDefault: true,
            });

            if (!isFocused && !event.defaultPrevented) {
              // The `merge: true` option makes sure that the params inside the
tab screen are preserved
              navigation.navigate({ name: route.name, merge: true });
            }
          };

          const onLongPress = () => {
            navigation.emit({
              type: 'tabLongPress',
              target: route.key,
            });
          };

          return (
            <TouchableOpacity
              accessibilityRole="button"
              accessibilityState={isFocused ? { selected: true } : {}}
              accessibilityLabel={options.tabBarAccessibilityLabel}
              testID={options.tabBarTestID}
              onPress={onPress}
              onLongPress={onLongPress}
              style={{ flex: 1 }}
            >
              <Text style={{ color: isFocused ? '#673ab7' : '#222' }}>
                {label}
```

```
            </Text>
          </TouchableOpacity>
        );
      })}
    </View>
  );
}

// ...

<Tab.Navigator tabBar={props => <MyTabBar {...props} />}>
  {...}
</Tab.Navigator>
```

Try this example on Snack

This example will render a basic tab bar with labels.

Note that you **cannot** use the `useNavigation` hook inside the `tabBar` since `useNavigation` is only available inside screens. You get a `navigation` prop for your `tabBar` which you can use instead:

```
function MyTabBar({ navigation }) {
  return (
    <Button
      title="Go somewhere"
      onPress={() => {
        // Navigate using the `navigation` prop that you received
        navigation.navigate('SomeScreen');
      }}
    />
  );
}
```

Options

The following options can be used to configure the screens in the navigator. These can be specified under `screenOptions` prop of `Tab.navigator` or `options` prop of `Tab.Screen`.

`title`

Generic title that can be used as a fallback for `headerTitle` and `tabBarLabel`.

`tabBarLabel`

Title string of a tab displayed in the tab bar or a function that given `{ focused: boolean, color: string }` returns a React.Node, to display in tab bar. When undefined, scene `title` is used. To hide, see `tabBarShowLabel`.

`tabBarShowLabel`

Whether the tab label should be visible. Defaults to `true`.

`tabBarLabelPosition`

Whether the label is shown below the icon or beside the icon.

- `below-icon`: the label is shown below the icon (typical for iPhones)
- `beside-icon` the label is shown next to the icon (typical for iPad)

By default, the position is chosen automatically based on device width.

`tabBarLabelStyle`

Style object for the tab label.

`tabBarIcon`

Function that given `{ focused: boolean, color: string, size: number }` returns a React.Node, to display in the tab bar.

`tabBarIconStyle`

Style object for the tab icon.

`tabBarBadge`

Text to show in a badge on the tab icon. Accepts a `string` or a `number`.

`tabBarBadgeStyle`

Style for the badge on the tab icon. You can specify a background color or text color here.

`tabBarAccessibilityLabel`

Accessibility label for the tab button. This is read by the screen reader when the user taps the tab. It's recommended to set this if you don't have a label for the tab.

`tabBarTestID`

ID to locate this tab button in tests.

`tabBarButton`

Function which returns a React element to render as the tab bar button. It wraps the icon and label. Renders `Pressable` by default.

You can specify a custom implementation here:

```
tabBarButton: (props) => <TouchableOpacity {...props} />;
```

`tabBarActiveTintColor`

Color for the icon and label in the active tab.

`tabBarInactiveTintColor`

Color for the icon and label in the inactive tabs.

`tabBarActiveBackgroundColor`

Background color for the active tab.

`tabBarInactiveBackgroundColor`

Background color for the inactive tabs.

`tabBarHideOnKeyboard`

Whether the tab bar is hidden when the keyboard opens. Defaults to `false`.

`tabBarItemStyle`

Style object for the tab item container.

`tabBarStyle`

Style object for the tab bar. You can configure styles such as background color here.

To show your screen under the tab bar, you can set the `position` style to absolute:

```
<Tab.Navigator
  screenOptions={{
    tabBarStyle: { position: 'absolute' },
  }}
>
```

You also might need to add a bottom margin to your content if you have a absolutely positioned tab bar. React Navigation won't do it automatically.

To get the height of the bottom tab bar, you can use `BottomTabBarHeightContext` with React's Context API or `useBottomTabBarHeight`:

```
import { BottomTabBarHeightContext } from '@react-navigation/bottom-tabs';

// ...

<BottomTabBarHeightContext.Consumer>
  {tabBarHeight => (
    /* render something */
  )}
</BottomTabBarHeightContext.Consumer>
```

or

```
import { useBottomTabBarHeight } from '@react-navigation/bottom-tabs';

// ...

const tabBarHeight = useBottomTabBarHeight();
```

`tabBarBackground`

Function which returns a React Element to use as background for the tab bar. You could render an image, a gradient, blur view etc.:

```
import { BlurView } from 'expo-blur';

// ...

<Tab.Navigator
  screenOptions={{
    tabBarStyle: { position: 'absolute' },
    tabBarBackground: () => (
      <BlurView tint="light" intensity={100} style={StyleSheet.absoluteFill} />
    ),
  }}
>
```

When using `BlurView`, make sure to set `position: 'absolute'` in `tabBarStyle` as well. You'd also need to use `useBottomTabBarHeight()` to add a bottom padding to your content.

`lazy`

Whether this screens should render the first time it's accessed. Defaults to `true`. Set it to `false` if you want to render the screen on initial render.

`unmountOnBlur`

Whether this screen should be unmounted when navigating away from it. Unmounting a screen resets any local state in the screen as well as state of nested navigators in the screen. Defaults to `false`.

Normally, we don't recommend enabling this prop as users don't expect their navigation history to be lost when switching tabs. If you enable this prop, please consider if this will actually provide a better experience for the user.

`freezeOnBlur`

Boolean indicating whether to prevent inactive screens from re-rendering. Defaults to `false`. Defaults to `true` when `enableFreeze()` from `react-native-screens` package is run at the top of

the application.

Requires `react-native-screens` version >=3.16.0.

Only supported on iOS and Android.

Header related options

You can find the list of header related options here. These options can be specified under `screenOptions` prop of `Tab.navigator` or `options` prop of `Tab.Screen`. You don't have to be using `@react-navigation/elements` directly to use these options, they are just documented in that page.

In addition to those, the following options are also supported in bottom tabs:

`header`

Custom header to use instead of the default header.

This accepts a function that returns a React Element to display as a header. The function receives an object containing the following properties as the argument:

- `navigation` - The navigation object for the current screen.
- `route` - The route object for the current screen.
- `options` - The options for the current screen
- `layout` - Dimensions of the screen, contains `height` and `width` properties.

Example:

```
import { getHeaderTitle } from '@react-navigation/elements';

// ..

header: ({ navigation, route, options }) => {
  const title = getHeaderTitle(options, route.name);

  return <MyHeader title={title} style={options.headerStyle} />;
};
```

To set a custom header for all the screens in the navigator, you can specify this option in the `screenOptions` prop of the navigator.

Specify a `height` in `headerStyle`

If your custom header's height differs from the default header height, then you might notice glitches due to measurement being async. Explicitly specifying the height will avoid such glitches.

Example:

```
headerStyle: {
  height: 80, // Specify the height of your custom header
};
```

Note that this style is not applied to the header by default since you control the styling of your custom header. If you also want to apply this style to your header, use `options.headerStyle` from the props.

`headerShown`

Whether to show or hide the header for the screen. The header is shown by default. Setting this to `false` hides the header.

Events

The navigator can emit events on certain actions. Supported events are:

`tabPress`

This event is fired when the user presses the tab button for the current screen in the tab bar. By default a tab press does several things:

- If the tab is not focused, tab press will focus that tab
- If the tab is already focused:
  - If the screen for the tab renders a scroll view, you can use `useScrollToTop` to scroll it to top
  - If the screen for the tab renders a stack navigator, a `popToTop` action is performed on the stack

To prevent the default behavior, you can call `event.preventDefault`:

```
React.useEffect(() => {
  const unsubscribe = navigation.addListener('tabPress', (e) => {
    // Prevent default behavior
    e.preventDefault();

    // Do something manually
    // ...
  });

  return unsubscribe;
}, [navigation]);
```

Try this example on Snack

If you have a custom tab bar, make sure to emit this event.

`tabLongPress`

This event is fired when the user presses the tab button for the current screen in the tab bar for an extended period. If you have a custom tab bar, make sure to emit this event.

Example:

```
React.useEffect(() => {
  const unsubscribe = navigation.addListener('tabLongPress', (e) => {
    // Do something
  });

  return unsubscribe;
}, [navigation]);
```

Helpers

The tab navigator adds the following methods to the navigation prop:

`jumpTo`

Navigates to an existing screen in the tab navigator. The method accepts following arguments:

- `name` - *string* - Name of the route to jump to.
- `params` - *object* - Screen params to use for the destination route.

```
navigation.jumpTo('Profile', { owner: 'Michaś' });
```

Try this example on Snack

Example

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import MaterialCommunityIcons from 'react-native-vector-
icons/MaterialCommunityIcons';

const Tab = createBottomTabNavigator();

function MyTabs() {
  return (
    <Tab.Navigator
      initialRouteName="Feed"
      screenOptions={{
        tabBarActiveTintColor: '#e91e63',
      }}
    >
      <Tab.Screen
        name="Feed"
        component={Feed}
        options={{
          tabBarLabel: 'Home',
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="home" color={color} size={size} />
          ),
        }}
      />
      <Tab.Screen
        name="Notifications"
        component={Notifications}
        options={{
          tabBarLabel: 'Updates',
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="bell" color={color} size={size} />
```

```
          ),
          tabBarBadge: 3,
        }}
      />
      <Tab.Screen
        name="Profile"
        component={Profile}
        options={{
          tabBarLabel: 'Profile',
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="account" color={color} size={size} />
          ),
        }}
      />
    </Tab.Navigator>
  );
}
```

Try this example on Snack

Edit this page