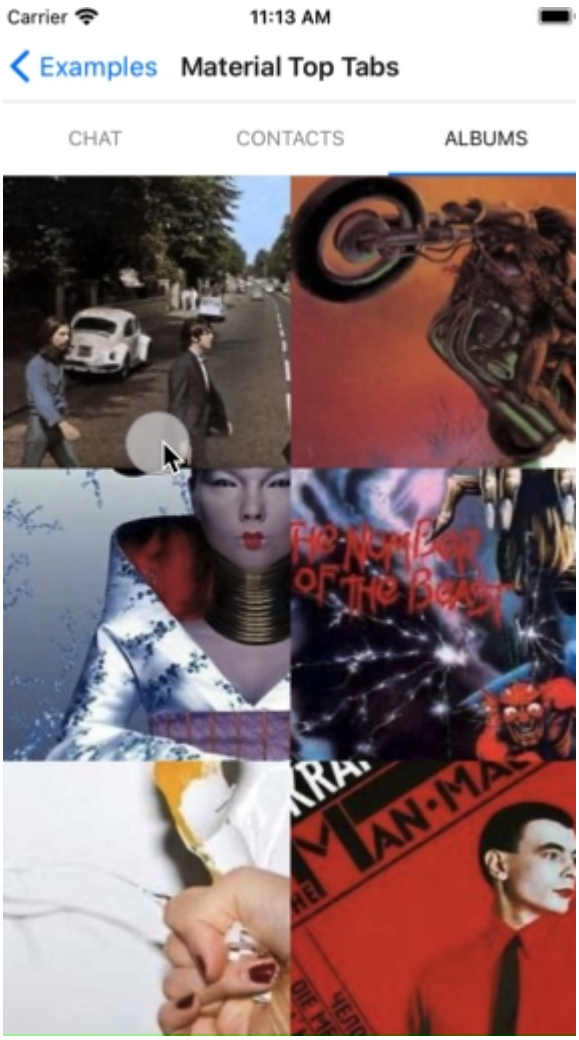
Version: 6.x

Material Top Tabs Navigator

A material-design themed tab bar on the top of the screen that lets you switch between different routes by tapping the tabs or swiping horizontally. Transitions are animated by default. Screen components for each route are mounted immediately.



This wraps `react-native-tab-view`. If you want to use the tab view without React Navigation integration, use the library directly instead.

Installation

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

npm Yarn

```
npm install @react-navigation/material-top-tabs react-native-tab-view
```

Then, you need to install `react-native-pager-view` which is required by the navigator.

If you have a Expo managed project, in your project directory, run:

```
npx expo install react-native-pager-view
```

If you have a bare React Native project, in your project directory, run:

npm Yarn

```
npm install react-native-pager-view
```

If you're on a Mac and developing for iOS, you also need to install the pods (via Cocoapods) to complete the linking.

```
npx pod-install ios
```

API Definition

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

```
import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';

const Tab = createMaterialTopTabNavigator();

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

`tabBarPosition`

Position of the tab bar in the tab view. Possible values are `'top'` and `'bottom'`. Defaults to `'top'`.

`keyboardDismissMode`

String indicating whether the keyboard gets dismissed in response to a drag gesture. Possible values are:

- `'auto'` (default): the keyboard is dismissed when the index changes.
- `'on-drag'`: the keyboard is dismissed when a drag begins.
- `'none'`: drags do not dismiss the keyboard.

`initialLayout`

Object containing the initial height and width of the screens. Passing this will improve the initial rendering performance. For most apps, this is a good default:

```
{
  width: Dimensions.get('window').width;
}
```

`sceneContainerStyle`

Style to apply to the view wrapping each screen. You can pass this to override some default styles such as overflow clipping.

`style`

Style to apply to the tab view container.

`tabBar`

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

Example:

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

function MyTabBar({ state, descriptors, navigation, position }) {
  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,
          });
        };

        const inputRange = state.routes.map((_, i) => i);
        const opacity = position.interpolate({
          inputRange,
          outputRange: inputRange.map(i => (i === index ? 1 : 0)),
```

```
        });

        return (
          <TouchableOpacity
            accessibilityRole="button"
            accessibilityState={isFocused ? { selected: true } : {}}
            accessibilityLabel={options.tabBarAccessibilityLabel}
            testID={options.tabBarTestID}
            onPress={onPress}
            onLongPress={onLongPress}
            style={{ flex: 1 }}
          >
            <Animated.Text style={{ opacity }}>
              {label}
            </Animated.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:

Example:

```
<Tab.Navigator
  screenOptions={{
    tabBarLabelStyle: { fontSize: 12 },
    tabBarItemStyle: { width: 100 },
    tabBarStyle: { backgroundColor: 'powderblue' },
  }}
>
  {/* ... */}
</Tab.Navigator>
```

Try this example on Snack

`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` option.

`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.

`tabBarAllowFontScaling`

Whether label font should scale to respect Text Size accessibility settings.

`tabBarShowLabel`

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

`tabBarIcon`

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

`tabBarShowIcon`

Whether the tab icon should be visible. Defaults to `false`.

`tabBarBadge`

Function that returns a React element to use as a badge for the tab.

`tabBarIndicator`

Function that returns a React element as the tab bar indicator.

`tabBarIndicatorStyle`

Style object for the tab bar indicator.

`tabBarIndicatorContainerStyle`

Style object for the view containing the tab bar indicator.

`tabBarTestID`

ID to locate this tab button in tests.

`tabBarActiveTintColor`

Color for the icon and label in the active tab.

`tabBarInactiveTintColor`

Color for the icon and label in the inactive tabs.

`tabBarGap`

Spacing between the tab items in the tab bar.

Example:

```
<Tab.Navigator
  //...
  screenOptions={{
    tabBarGap: 10,
  }}
>
</Tab.Navigator>
```

`tabBarAndroidRipple`

Allows to customize the android ripple effect.

Example:

```
<Tab.Navigator
  //...
  screenOptions={{
    tabBarAndroidRipple: { borderless: false },
  }}
>
</Tab.Navigator>
```

`tabBarPressColor`

Color for material ripple.

Only supported on Android.

`tabBarPressOpacity`

Opacity for pressed tab.

Only supported on iOS.

`tabBarBounces`

Boolean indicating whether the tab bar bounces when overscrolling.

`tabBarScrollEnabled`

Boolean indicating whether to make the tab bar scrollable.

If you set this to `true`, you should also specify a width in `tabBarItemStyle` to improve the performance of initial render.

`tabBarIconStyle`

Style object for the tab icon container.

`tabBarLabelStyle`

Style object for the tab label.

`tabBarItemStyle`

Style object for the individual tab items.

`tabBarContentContainerStyle`

Style object for the view containing the tab items.

`tabBarStyle`

Style object for the tab bar.

`swipeEnabled`

Boolean indicating whether to enable swipe gestures. Swipe gestures are enabled by default. Passing `false` will disable swipe gestures, but the user can still switch tabs by pressing the tab bar.

`lazy`

Whether this screen should be lazily rendered. When this is set to `true`, the screen will be rendered as it comes into the viewport. By default all screens are rendered to provide a smoother swipe experience. But you might want to defer the rendering of screens out of the viewport until the user sees them. To enable lazy rendering for this screen, set `lazy` to `true`.

When you enable `lazy`, the lazy loaded screens will usually take some time to render when they come into the viewport. You can use the `lazyPlaceholder` prop to customize what the user sees during this short period.

`lazyPreloadDistance`

When `lazy` is enabled, you can specify how many adjacent screens should be preloaded in advance with this prop. This value defaults to `0` which means lazy pages are loaded as they come into the viewport.

`lazyPlaceholder`

Function that returns a React element to render if this screen hasn't been rendered yet. The `lazy` option also needs to be enabled for this to work.

This view is usually only shown for a split second. Keep it lightweight.

By default, this renders `null`.

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

`tabLongPress`

This event is fired when the user presses the tab button for the current screen in the tab bar for an extended period.

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 pass to the destination route.

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

Try this example on Snack

Example

```
import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';

const Tab = createMaterialTopTabNavigator();

function MyTabs() {
  return (
    <Tab.Navigator
      initialRouteName="Feed"
      screenOptions={{
        tabBarActiveTintColor: '#e91e63',
        tabBarLabelStyle: { fontSize: 12 },
        tabBarStyle: { backgroundColor: 'powderblue' },
      }}
    >
      <Tab.Screen
        name="Feed"
        component={Feed}
        options={{ tabBarLabel: 'Home' }}
      />
      <Tab.Screen
        name="Notifications"
        component={Notifications}
        options={{ tabBarLabel: 'Updates' }}
      />
      <Tab.Screen
        name="Profile"
        component={Profile}
        options={{ tabBarLabel: 'Profile' }}
```

```
      />
    </Tab.Navigator>
  );
}
```

Try this example on Snack

Edit this page