Version: 6.x

Nesting navigators

Nesting navigators means rendering a navigator inside a screen of another navigator, for example:

```
function Home() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Feed" component={Feed} />
      <Tab.Screen name="Messages" component={Messages} />
    </Tab.Navigator>
  );
}

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={Home}
          options={{ headerShown: false }}
        />
        <Stack.Screen name="Profile" component={Profile} />
        <Stack.Screen name="Settings" component={Settings} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

In the above example, the `Home` component contains a tab navigator. The `Home` component is also used for the `Home` screen in your stack navigator inside the `App` component. So here, a tab navigator is nested inside a stack navigator:

- `Stack.Navigator`
  - `Home` (`Tab.Navigator`)

        - `Feed` (`Screen`)
        - `Messages` (`Screen`)
    - `Profile` (`Screen`)
    - `Settings` (`Screen`)

Nesting navigators work very much like nesting regular components. To achieve the behavior you want, it's often necessary to nest multiple navigators.

How nesting navigators affects the behaviour

When nesting navigators, there are some things to keep in mind:

Each navigator keeps its own navigation history

For example, when you press the back button when inside a screen in a nested stack navigator, it'll go back to the previous screen inside the nested stack even if there's another navigator as the parent.

Each navigator has its own options

For example, specifying a `title` option in a screen nested in a child navigator won't affect the title shown in a parent navigator.

If you want to achieve this behavior, see the guide for screen options with nested navigators. this could be useful if you are rendering a tab navigator inside a stack navigator and want to show the title of the active screen inside the tab navigator in the header of the stack navigator.

Each screen in a navigator has its own params

For example, any `params` passed to a screen in a nested navigator are in the `route` prop of that screen and aren't accessible from a screen in a parent or child navigator.

If you need to access params of the parent screen from a child screen, you can use React Context to expose params to children.

Navigation actions are handled by current navigator and bubble up if couldn't be handled

For example, if you're calling `navigation.goBack()` in a nested screen, it'll only go back in the parent navigator if you're already on the first screen of the navigator. Other actions such as `navigate` work similarly, i.e. navigation will happen in the nested navigator and if the nested navigator couldn't handle it, then the parent navigator will try to handle it. In the above example, when calling `navigate('Messages')`, inside `Feed` screen, the nested tab navigator will handle it, but if you call `navigate('Settings')`, the parent stack navigator will handle it.

Navigator specific methods are available in the navigators nested inside

For example, if you have a stack inside a drawer navigator, the drawer's `openDrawer`, `closeDrawer`, `toggleDrawer` methods etc. will also be available on the `navigation` prop in the screen's inside the stack navigator. But say you have a stack navigator as the parent of the drawer, then the screens inside the stack navigator won't have access to these methods, because they aren't nested inside the drawer.

Similarly, if you have a tab navigator inside stack navigator, the screens in the tab navigator will get the `push` and `replace` methods for stack in their `navigation` prop.

If you need to dispatch actions to the nested child navigators from a parent, you can use `navigation.dispatch`:

```
navigation.dispatch(DrawerActions.toggleDrawer());
```

Nested navigators don't receive parent's events

For example, if you have a stack navigator nested inside a tab navigator, the screens in the stack navigator won't receive the events emitted by the parent tab navigator such as (`tabPress`) when using `navigation.addListener`.

To receive events from parent navigator, you can explicitly listen to parent's events with `navigation.getParent`:

```
const unsubscribe = navigation
  .getParent('MyTabs')
  .addListener('tabPress', (e) => {
```

```
    // Do something
  });
```

Try this example on Snack

Here `'MyTabs'` refers to the value you pass in the `id` prop of the parent `Tab.Navigator` whose event you want to listen to.

Parent navigator's UI is rendered on top of child navigator

For example, when you nest a stack navigator inside a drawer navigator, you'll see that the drawer appears above the stack navigator's header. However, if you nest the drawer navigator inside a stack, the drawer will appear below the header of the stack. This is an important point to consider when deciding how to nest your navigators.

In your app, you will probably use these patterns depending on the behavior you want:

- Tab navigator nested inside the initial screen of stack navigator - New screens cover the tab bar when you push them.
- Drawer navigator nested inside the initial screen of stack navigator with the initial screen's stack header hidden - The drawer can only be opened from the first screen of the stack.
- Stack navigators nested inside each screen of drawer navigator - The drawer appears over the header from the stack.
- Stack navigators nested inside each screen of tab navigator - The tab bar is always visible. Usually pressing the tab again also pops the stack to top.

Navigating to a screen in a nested navigator

Consider the following example:

```
function Root() {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={Home} />
      <Drawer.Screen name="Profile" component={Profile} />
      <Stack.Screen name="Settings" component={Settings} />
    </Drawer.Navigator>
  );
```

```
}

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Root"
          component={Root}
          options={{ headerShown: false }}
        />
        <Stack.Screen name="Feed" component={Feed} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

Here, you might want to navigate to the `Root` screen from your `Feed` component:

```
navigation.navigate('Root');
```

It works, and the initial screen inside the `Root` component is shown, which is `Home`. But sometimes you may want to control the screen that should be shown upon navigation. To achieve it, you can pass the name of the screen in params:

```
navigation.navigate('Root', { screen: 'Profile' });
```

Now, the `Profile` screen will be rendered instead of `Home` upon navigation.

This may look very different from the way navigation used to work with nested screens previously. The difference is that in the previous versions, all configuration was static, so React Navigation could statically find the list of all the navigators and their screens by recursing into nested configurations. But with dynamic configuration, React Navigation doesn't know which screens are available and where until the navigator containing the screen renders. Normally, a screen doesn't render its contents until you navigate to it, so the configuration of navigators which haven't rendered is not

yet available. This makes it necessary to specify the hierarchy you're navigating to. This is also why you should have as little nesting of navigators as possible to keep your code simpler.

Passing params to a screen in a nested navigator

You can also pass params by specifying a `params` key:

```
navigation.navigate('Root', {
  screen: 'Profile',
  params: { user: 'jane' },
});
```

Try this example on Snack

If the navigator was already rendered, navigating to another screen will push a new screen in case of stack navigator.

You can follow similar approach for deeply nested screens. Note that the second argument to `navigate` here is just `params`, so you can do something like:

```
navigation.navigate('Root', {
  screen: 'Settings',
  params: {
    screen: 'Sound',
    params: {
      screen: 'Media',
    },
  },
});
```

In the above case, you're navigating to the `Media` screen, which is in a navigator nested inside the `Sound` screen, which is in a navigator nested inside the `Settings` screen.

Rendering initial route defined in the navigator

By default, when you navigate a screen in the nested navigator, the specified screen is used as the initial screen and the initial route prop on the navigator is ignored. This behaviour is different from the React Navigation 4.

If you need to render the initial route specified in the navigator, you can disable the behaviour of using the specified screen as the initial screen by setting `initial: false`:

```
navigation.navigate('Root', {
  screen: 'Settings',
  initial: false,
});
```

This affects what happens when pressing the back button. When there's an initial screen, the back button will take the user there.

Nesting multiple navigators

It's sometimes useful to nest multiple navigators such as stack, drawer or tabs.

When nesting multiple stack, drawer or bottom tab navigator, headers from both child and parent navigators would be shown. However, usually it's more desirable to show the header in the child navigator and hide the header in the screen of the parent navigator.

To achieve this, you can hide the header in the screen containing the navigator using the `headerShown: false` option.

For example:

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

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
```

```
          component={Home}
          options={{ headerShown: false }}
        />
        <Stack.Screen name="EditPost" component={EditPost} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

In these examples, we have used a bottom tab navigator directly nested inside another stack navigator, but the same principle applies when there are other navigators in the middle, for example: stack navigator inside a tab navigator which is inside another stack navigator, stack navigator inside drawer navigator etc.

If you don't want headers in any of the navigators, you can specify `headerShown: false` in all of the navigators:

```
function Home() {
  return (
    <Tab.Navigator screenOptions={{ headerShown: false }}>
      <Tab.Screen name="Profile" component={Profile} />
      <Tab.Screen name="Settings" component={Settings} />
    </Tab.Navigator>
  );
}

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator screenOptions={{ headerShown: false }}>
        <Stack.Screen name="Home" component={Home} />
        <Stack.Screen name="EditPost" component={EditPost} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Best practices when nesting

We recommend to reduce nesting navigators to minimal. Try to achieve the behavior you want with as little nesting as possible. Nesting has many downsides:

- It results in deeply nested view hierarchy which can cause memory and performance issues in lower end devices
- Nesting same type of navigators (e.g. tabs inside tabs, drawer inside drawer etc.) might lead to a confusing UX
- With excessive nesting, code becomes difficult to follow when navigating to nested screens, configuring deep link etc.

Think of nesting navigators as a way to achieve the UI you want rather than a way to organize your code. If you want to create separate group of screens for organization, instead of using separate navigators, you can use the `Group` component.

```
<Stack.Navigator>
  {isLoggedIn ? (
    // Screens for logged in users
    <Stack.Group>
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="Profile" component={Profile} />
    </Stack.Group>
  ) : (
    // Auth screens
    <Stack.Group screenOptions={{ headerShown: false }}>
      <Stack.Screen name="SignIn" component={SignIn} />
      <Stack.Screen name="SignUp" component={SignUp} />
    </Stack.Group>
  )}
  {/* Common modal screens */}
  <Stack.Group screenOptions={{ presentation: 'modal' }}>
    <Stack.Screen name="Help" component={Help} />
    <Stack.Screen name="Invite" component={Invite} />
  </Stack.Group>
</Stack.Navigator>
```

Try this example on Snack

Edit this page