Version: 6.x

Hello React Navigation

In a web browser, you can link to different pages using an anchor (`<a>`) tag. When the user clicks on a link, the URL is pushed to the browser history stack. When the user presses the back button, the browser pops the item from the top of the history stack, so the active page is now the previously visited page. React Native doesn't have a built-in idea of a global history stack like a web browser does -- this is where React Navigation enters the story.

React Navigation's native stack navigator provides a way for your app to transition between screens and manage navigation history. If your app uses only one stack navigator then it is conceptually similar to how a web browser handles navigation state - your app pushes and pops items from the navigation stack as users interact with it, and this results in the user seeing different screens. A key difference between how this works in a web browser and in React Navigation is that React Navigation's native stack navigator provides the gestures and animations that you would expect on Android and iOS when navigating between routes in the stack.

Let's start by demonstrating the most common navigator, `createNativeStackNavigator`.

Installing the native stack navigator library

The libraries we've installed so far are the building blocks and shared foundations for navigators, and each navigator in React Navigation lives in its own library. To use the native stack navigator, we need to install `@react-navigation/native-stack` :

npm Yarn

```
npm install @react-navigation/native-stack
```

> 💡 `@react-navigation/native-stack` depends on `react-native-screens` and the other libraries that we installed in Getting started. If you haven't installed those yet, head over to that page and follow the installation instructions.

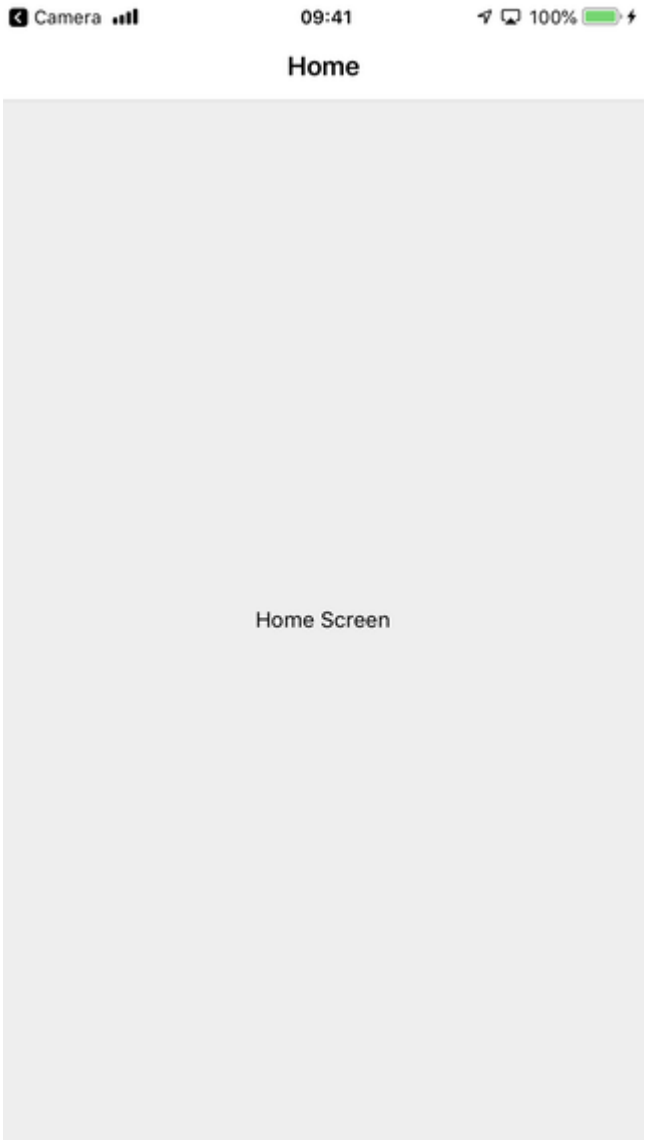
Creating a native stack navigator

`createNativeStackNavigator` is a function that returns an object containing 2 properties: `Screen` and `Navigator`. Both of them are React components used for configuring the navigator. The `Navigator` should contain `Screen` elements as its children to define the configuration for routes.

`NavigationContainer` is a component which manages our navigation tree and contains the navigation state. This component must wrap all navigators structure. Usually, we'd render this component at the root of our app, which is usually the component exported from `App.js`.

```
// In App.js in a new project

import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Try this example on Snack



If you run this code, you will see a screen with an empty navigation bar and a grey content area containing your `HomeScreen` component (shown above). The styles you see for the navigation bar and the content area are the default configuration for a stack navigator, we'll learn how to configure those later.

> The casing of the route name doesn't matter -- you can use lowercase `home` or capitalized `Home`, it's up to you. We prefer capitalizing our route names.

Configuring the navigator

All of the route configuration is specified as props to our navigator. We haven't passed any props to our navigator, so it just uses the default configuration.

Let's add a second screen to our native stack navigator and configure the `Home` screen to render first:

```
function DetailsScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

Now our stack has two *routes*, a `Home` route and a `Details` route. A route can be specified by using the `Screen` component. The `Screen` component accepts a `name` prop which corresponds to the name of the route we will use to navigate, and a `component` prop which corresponds to the component it'll render.

Here, the `Home` route corresponds to the `HomeScreen` component, and the `Details` route corresponds to the `DetailsScreen` component. The initial route for the stack is the `Home` route. Try changing it to `Details` and reload the app (React Native's Fast Refresh won't update changes from `initialRouteName`, as you might expect), notice that you will now see the `Details` screen. Then change it back to `Home` and reload once more.

> Note: The `component` prop accepts component, not a render function. Don't pass an inline function (e.g. `component={() => <HomeScreen />}`), or your component will unmount and remount losing all state when the parent component re-renders. See Passing additional props for alternatives.

Specifying options

Each screen in the navigator can specify some options for the navigator, such as the title to render in the header. These options can be passed in the `options` prop for each screen component:

```
<Stack.Screen
  name="Home"
  component={HomeScreen}
  options={{ title: 'Overview' }}
/>
```

Try this example on Snack

Sometimes we will want to specify the same options for all of the screens in the navigator. For that, we can pass a `screenOptions` prop to the navigator.

Passing additional props

Sometimes we might want to pass additional props to a screen. We can do that with 2 approaches:

1. Use React context and wrap the navigator with a context provider to pass data to the screens (recommended).

2. Use a render callback for the screen instead of specifying a `component` prop:

   ```
   <Stack.Screen name="Home">
     {(props) => <HomeScreen {...props} extraData={someData} />}
   </Stack.Screen>
   ```

> Note: By default, React Navigation applies optimizations to screen components to prevent unnecessary renders. Using a render callback removes those optimizations. So if you use a render callback, you'll need to ensure that you use `React.memo` or `React.PureComponent` for your screen components to avoid performance issues.

What's next?

The natural question at this point is: "how do I go from the `Home` route to the `Details` route?". That is covered in the next section.

Summary

- React Native doesn't have a built-in API for navigation like a web browser does. React Navigation provides this for you, along with the iOS and Android gestures and animations to transition between screens.
- `Stack.Navigator` is a component that takes route configuration as its children with additional props for configuration and renders our content.
- Each `Stack.Screen` component takes a `name` prop which refers to the name of the route and `component` prop which specifies the component to render for the route. These are the 2 required props.
- To specify what the initial route in a stack is, provide an `initialRouteName` as the prop for the navigator.
- To specify screen-specific options, we can pass an `options` prop to `Stack.Screen`, and for common options, we can pass `screenOptions` to `Stack.Navigator`

Edit this page