Version: 6.x

Integrating with MobX State Tree

> TODO: This guide is incomplete. Please help improve this by sending pull requests.

This guide explores possible way to use React Navigation in a React Native project that uses MobX State Tree(MST) for state management. The guide is accompanied by a sample app. Parts of the guide may be relevant also for users of MobX but please be aware of the fact that MobX does not come with a built-in solution for (de)serializing its state.

> Please note that in this guide, Mobx State Tree is not used to manage the navigation state itself - just the navigation params!

Overview

Our goal with this guide is to use MST with React Navigation and achieve optimal developer experience. In the scope of this guide, this means allowing us to do a full JS reload and be brought back to the state before the reload happened.

We will do this by persisting the navigation state using the React Navigation's built-in mechanism. We also need to persist the app state and navigation params - that way, when you're working on a screen in your app and do a full JS reload, you will be brought back to the same screen, with the same data in it.

Guide

First, start by creating initial navigation structure and React components. When you're done with that, continue with modelling your state in MST. If you want to learn more about this, check out the egghead.io course.

At this point, you're probably wondering how to connect your MST objects with the components. The answer is in the mobx-react package that contains React bindings for MobX (they also work for MST). You will likely be using the `Provider` component and the `inject` and `observer` functions.

Use `Provider` to wrap what you return from your root component's render method:

```
<Provider myObject={this.myObject}>
  <NavigationContainer>{/* Screen configuration */}</NavigationContainer>
</Provider>
```

this will allow you to access `myObject` from any React component in the application through the `inject` function which can be quite useful.

Use `observer` function to wrap all components that render observable data. This will make sure the components re-render once the data they render changes.

Navigation params

Screens in your application often depend on params. React Navigation allows you to send params from one screen to another. These params are stored in the navigation state. However, in order to persist the navigation state, it needs to be serializable. This requirement does not play well with MST, because the MST objects are complex objects and React Navigation doesn't know how to (de)serialize them. In this guide, we will work around this by storing the navigation params ourselves.

This means that rather than sending the params from one screen to another (eg. with `props.navigation.navigate('MyScreen', { complexMSTObject })`) we will store the params to a navigation store, then navigate without sending any params, and on the target screen, we'll pick the params up from the navigation store.

To give an example, the navigation store may look similar to this:

```
import { types, onSnapshot, getRoot } from 'mobx-state-tree';
import { Product } from '../models/Product';
import { User } from '../models/User';

export const NavigationStore = types
  .model('NavigationStore', {
    productDetailScreenParams: types.map(
      types.model('ProductDetailScreenParams', {
        product: types.optional(types.safeReference(Product)),
      })
    ),
    userProfileScreenParams: types.model('UserProfileScreenParams', {
      user: types.maybe(types.safeReference(User)),
    }),
  })
  .actions(self => ({
    ...
  }));
```

Note that `userProfileScreenParams` is a simple model with a `user` entry, while `productDetailScreenParams` is a map of `ProductDetailScreenParams` model. The reason we chose this shape of data is that we only have a single user profile screen in our app which reads its params from `userProfileScreenParams`. `productDetailScreenParams` is a map because the app can have several product screens on a stack. Each screen points to a `Product` instance saved in the map. The keys into the map are the React Navigation keys: think of the `key` as of an identifier of the route.

Your navigation store may also be just one map where for each screen (regardless if it is a product or user profile screen), we store its navigation params. This is the approach taken in the sample app.

Summary

- you can use React Navigation with MobX State Tree in a React Native app
- use the `Provider` component and the `inject` and `observer` functions to wire up MobX or MST with React
- it's possible to persist the entire application state and restore it upon JS reload

Edit this page