Conclusion

17



Manujith Pallewatte

17

# Using React Router with Redux

Manujith Pallewatte

Aug 28, 2020 • 9 Min read • 7,618 Views

Aug 28, 2020 • 9 Min read • 7,618 Views

Web Development Front End Web Dev... Client-side Framew... React



## Introduction

Redux and React Router are two of the most used React libraries. Redux is used for app state management, while React Router is used for routing. In a majority of apps, it's necessary to communicate between the app state and the router. Usually, this is in response to an initial user action.

Although both libraries are widely used, the integration between the two is not trivial. There are varying opinions on the matter, and each approach is aimed at solving a specific set of issues. In this

guide, we will explore two such approaches that enable us to seamlessly integrate Redux with React Router.

17

the scope of this guide, let's assume the implementation of an e-commerce stop app with three sample pages:

1. Product submission page: A user can fill a form to submit a product
2. Product listing page: All submitted products are listed here
3. Product view page: On selecting a product from the listing, the user is navigated here to view the individual product detail

We will not discuss the implementation in detail but will highlight a few facts relating to the implementation of Redux and React Router in the following code.

## Keeping Redux and React Router Separate

Given the above example, you can first explore how the two libraries can work together without being tightly coupled. Since both libraries are already installed and configured, the next step is to observe a couple of practical scenarios where you would need to create communication between the two.

**1. Changing routes in response to state change**

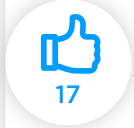
17

One of the most common Redux-React Router interactions is to perform a route change after a certain app state is changed. For example, consider a form submission. You would ideally keep the form state locally (directly in the component state or using a library such as Formik). Once the form submission is fired, you will dispatch a thunk action with the form data. Assume that the form submission is to create a user in your app and on successful submission you need to route the user back to a user table. Similarly, on a failed submission you need to keep the user at the same place (form UI) and show error feedback.

The first challenge is to detect the form submission completion and its state. With the unidirectional data flow architecture of Redux, this is not straightforward. The key problem is the inability of the Redux store to access the router state. Even if you somehow externally access the router from a reducer or an action, this violates the condition of a pure function by introducing a side effect.

So the only possible way is to propagate the state change to a component and let the component trigger the router change. For this, you can keep a form submission state in the Redux store and update the variable on API call response of the form. The following code shows an extraction of code that uses the above method.

Conclusion







jsx

```jsx
// actions.js

const initState = {
    isSubmitted: false
}

const formSlice = createSlice({
    name: "form",
    initialState: initState,
    reducers: {
        setIsFormSubmitted(state, { action, payload }){
            state.isSubmitted = payload
        },
        // ...
    }
});

export const formActions = formSlice.actions;
export const formReducer = formSlice.reducer;

export function submitForm(formData){
    return async (dispatch, getState) => {
        try{
            const response = await api.submitForm(formData);
            dispatch(formActions.setIsFormSubmitted(true));
        }catch{
            console.error("Error submitting the form");
        }
    }
}

// ...

// ProductSubmissionPage.jsx
```

17

```
import React, { useState } from 'react';
import { useSelector } from 'react-redux';
import { useHistory } from 'react-router-dom';
import { useDispatch } from 'react-redux/lib/hooks/useDispatch';
import * as apiActions from './state';

export function ProductSubmissionPage(){
    const [formState, setFormState] = useState({});
    const isSubmitted = useSelector(state => state.form.isSubmitted);
    const history = useHistory();
    const dispatch = useDispatch();

    useState(() => {
        if(isSubmitted){
            history.push("/products");
        }
    }, [isSubmitted]);

    const submitForm = () => {
        dispatch(apiActions.submitForm(formState));
    }

    return (
        <div>
            ...
        </div>
    )
}
```

You can see the use of the `useHistory` hook to gain access to the history instance of the router. In pre-hooks versions of React Router, you had to pass any component that required access to the

Conclusion

17

- Introduction
- Keeping Redux and Re
- Integrating Redux and
- Pros and Cons
- Conclusion
- Top ^

router state through a `withRouter` HoC (higher-order component). With the introduction of the `useHistory` hook, the process is now ch simpler.

17

Although the above works as expected, it adds unnecessary complications. You are forced to keep a form state variable in your app state just to facilitate the routing of the app. So a better way to handle the situation is to concentrate the entire form submission flow into the component itself. With this, you can now implement the entire form control flow at the same point and trigger router changes easily. The code below demonstrates this idea.

jsx

```jsx
// ProductSubmission.jsx
    // ...

    const submitForm = () => {
        try{
            const response = await api.submitForm(formState);
            if(response.message === "OK"){
                history.push("/products");
            }else{
                console.error("Error submitting the form");
            }
        }catch{
            console.error("Error submitting the form");
        }
    }

    // ...
```

17

### 2. Changing state in response to a route change

other instance is making changes to the state based on the route ansitions. For example, consider an e-commerce store app with a product listing. A user would select a product from the listing and be redirected to a link similar to `http://ecommerce.app/products/123456` where `123456` is the product ID. In this instance, the app state should be updated to reflect that the current active product ID is `123456`.

Just as in the earlier scenario, the best way to resolve the above is through a rendered component that has access both to the app state and the router state. Using the `useEffect` and `useParams` hooks, you can listen to changes in the path parameter of the URL and update the app state accordingly.

jsx

```jsx
// ProductViewPage.jsx

import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import { useDispatch } from 'react-redux/lib/hooks/useDispatch';
import { productActions } from './state';

export function ProductViewPage(){
    const { productId } = useParams();
    const dispatch = useDispatch();

```

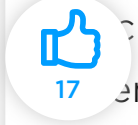
```
        useEffect(() => {
            dispatch(productActions.setActiveProduct(productId));
        }, [productId]);

        return (
            <div>
                ...
            </div>
        )
    }
```

## Integrating Redux and React Router

If you are starting the project from scratch, another option available to you is to integrate the React-Router state with the Redux store. You can use the `connected-react-router` library (formerly known as `react-router-redux`). Their Github Repo details the steps for the integration. Once the setup is complete, you can now access the router state directly within Redux as well as dispatch actions to modify the router state within Redux actions.

## Pros and Cons




While both above options are perfectly legal methods to integrate Redux with React Router, there are a few inherent pros and cons of ch method, and the correct choice will greatly depend on your enario.

Considering the ease of use, the second approach of integrating the router state with the app state is quite obviously the better choice. It prevents the need to have a component mediating between the router and the state.

But if you consider the readability of the code, you can see that the integrated approach slowly starts to hide critical implementation details away from the component where the actual action occurs. For example, earlier you described the entire life cycle from extracting the path parameters of the URL to navigation after form submission within the same component code. This is much more verbose and declarative than having a part of the implementation hidden inside the reducer and action codes in a different code file.

## Conclusion

In this guide, we explored two approaches of using React Router with Redux. Although both approaches are valid, each has its pros

and cons. I would recommend the first approach for a reasonably sized project given that it presents the key flow of events at the ne place versus being distributed among several files.

17

Conclusion

17