Mark's Dev Blog (https://blog.isquaredsoftware.com/)

Random musings on React, Redux, and more, by Redux maintainer Mark "acemarke" Erikson

Blogged Answers: The Evolution of Redux Testing Approaches

Posted on Jun 22, 2021

#javascript (https://blog.isquaredsoftware.com//tags/javascript) #react (https://blog.isquaredsoftware.com//tags/react) #redux (https://blog.isquaredsoftware.com//tags/redux)

()This is a post in the **Blogged Answers** (/series/blogged-answers) series.

Thoughts on how Redux testing has evolved from 'isolation' to 'integration'

Introduction

I was just asked a question about how testing works for Redux applications:

> 1. *For a beginner front-end developer, what would you expect them to know about Redux testing? (techniques and patterns specifically)*
> 2. *What - if any - tools are industry-standard for Redux testing? I've seen some tutorials using redux-mock-store, fetchMock, redux-testkit, redux-action-assertions, etc.*

I ended up writing an extended response about the history of testing Redux apps, the two major styles of testing that I've seen, and how those approaches

have evolved over time - clearly worthy of being reposted publicly.

Testing Redux

What should a beginning dev know about testing Redux?

For #1, I wouldn't necessarily expect a beginner to know much testing at all, really :) I actually just spent the last couple weeks helping guide a pair of interns through writing their first React component tests with React Testing Library, and it was a bit eye-opening when I had to explain so many different moving pieces: Jest, RTL, `describe`, `it`, `jest.fn()`, `render()`, `expect()`, and even just "write multiple tests that feed in different inputs, and check to see what the output is". It's not an easy or natural thing, and I'd kinda forgotten that.

All that said: the bare basic bit of testing I'd expect a beginner to be able to at least try or hopefully understand is testing reducers, because those are the simplest possible things to test: pure functions, whose entire logic depends on `(state, action)`, and it's just `expect(actual).toEqual(expected)` in some way

What's standard for Redux testing?

For #2: this has varied a lot, and I think it's changed some over the last couple years.

The real question here is about how you test your Redux code. Do you test pieces in isolation? reducers, selectors, thunks, sagas, etc. Or do you test your Redux logic integrated into the rest of the app?

Our docs have always taught the "isolation" approach, and that does especially make sense for reducers and selectors. The "integration" approach was in a minority.

But, RTL and Kent C Dodds have drastically changed the mindset and approach for testing in the React ecosystem. The patterns I see now are about "integration"-style tests - large chunks of code, working together, as they'd be used in a real app.

These two approaches lead to very different styles of writing tests.

"Isolation"-style tests

With the "isolation" approach, again, writing tests for reducers and selectors is trivial - pure functions.

Testing thunks and components, on the other hand, got a lot more complicated.

`redux-mock-store`'s primary purpose, as I understand it, is to collect a list of dispatched actions for later assertions: "yes, we did in fact dispatch `todoAdded` with this text", etc. That makes sense when you're testing things in isolation.

Testing thunks has always been really tricky for that approach, though. You have to configure the mock store with the thunk middleware. And what about any async calls? Thunks tend to directly import AJAX libs or client API layers - Angular-style DI has never been a big practice in the React ecosystem. So, that makes it harder to test thunks that make async calls. For that matter, thunks can dispatch *other* thunks too. So, testing dispatched actions is understandable here given those limitations.

The thunk middleware *does* have an "extra argument" that can be defined at middleware setup time, and that has been used for injecting some service layer for API calls that can be swapped with a mock version in tests.

This also gets very tricky when looking at components, especially when using the `connect` API with Enzyme. One of the reasons for the popularity of the "container/presentational" pattern is that it made it very easy to test presentational components. They're props-only, no logic other than formatting, no dependencies on any external APIs or behavior.

That concept is still a valid thing to do, but the React ecosystem has strongly moved away from "containers" thanks to hooks. I talked about this in my post Thoughts on React Hooks, Redux, and Separation of Concerns (/2019/07/blogged-answers-thoughts-on-hooks/), and my talk ReactBoston 2019: Hooks, HOCs, and Tradeoffs (/2019/09/presentation-hooks-hocs-tradeoffs/).

`connect` does have an option to pass a Redux store instance directly as a prop named `store`. That works okay if you've only got one level of connected

component being tested, but if you're doing an Enzyme `mount()`, which is a full render of the component tree, and there are *other* connected components in that subtree, they don't get access to the store correctly. So, you occasionally saw people creating real Redux stores and using a `<Provider>` in their component tests, but it was definitely a rarer thing

"Integration"-style tests

Well, that has flipped around over the last couple years with the arrival of RTL and hooks. Granted, I don't spend a lot of time reading other people's tests, and a lot of what I'm about to describe is my *own* experience working on some apps and working with other members of the Redux team. But, what I'm seeing is that "integration" style tests:

- always create a real Redux store in a test and wrap the component under test in a `<Provider>`. Typically there's a customized `render()` function that wraps the RTL `render` method, accepts a store as an option or creates one internally if none was passed in, and automatically does the provider wrapping.
- either fill the store in with fake data on creation, or dispatch some actions to load it up
- don't care about what actions were dispatched - what matters is "I click the 'Add Todo' button, and another item shows up in the list"
- mock async requests at the `fetch/xhr` level using tools like `msw`, `miragejs`, `jest-mock-fetch`, or similar. that way, none of the thunk logic has to change in a test - the thunk still tries to make a "real" async request, it just gets intercepted

Testing Sagas

I know that a number of people have chosen to use sagas specifically because they *don't* actually make real async requests in the saga function - it's just a stream of descriptions that the saga middleware is supposed to execute. in theory, that makes them much more testable.

having said that, I've seen discussions showing some annoyance with testing sagas, because you can effectively end up testing internal implementation details: "first it yields X, then it yields Y", etc.

I believe https://github.com/jfairbank/redux-saga-test-plan (https://github.com/jfairbank/redux-saga-test-plan) picked up some traction as a potentially better way to test sagas, but I haven't tried to do any of that myself

Recommendations

I can say that we did update the Redux docs "Testing" page to show more of an "integration"-type approach for testing connected components recently, although I think we ought to make that a bit more clear.

The couple projects I've been on in the last couple years haven't had tons of tests, but the integration-style approach seems to work out very well for us.

The new RTK Query APIs in Redux Toolkit are *entirely* "integration"-style tests, using MSW for all the API calls, and that's worked out fantastically well.

So. if you're going to teach anything, I'd say:

- teach how to do basic tests for pure functions (reducers, selectors)
- teach integration tests for everything working together (`<Provider>` + store wrapped around component, clicking a button does whatever real Redux logic, API calls are mocked out so app code doesn't have to change, assert UI is updated appropriately)

()This is a post in the **Blogged Answers** (/series/blogged-answers) series. Other posts in this series:

- Aug 08, 2023 - **Blogged Answers: My Experience Modernizing Packages to ESM** (https://blog.isquaredsoftware.com/2023/08/esm-modernization-lessons/)
- Jul 06, 2022 - **Blogged Answers: How I Estimate NPM Package Market Share (and how Redux usage compares to other libraries)** (https://blog.isquaredsoftware.com/2022/07/npm-package-market-share-estimates/)
- Jun 22, 2021 - **Blogged Answers: The Evolution of Redux Testing Approaches**
- Jan 18, 2021 - **Blogged Answers: Why React Context is Not a "State**

Management" Tool (and Why It Doesn't Replace Redux) (https://blog.isquaredsoftware.com/2021/01/context-redux-differences/)

- Jun 21, 2020 - **Blogged Answers: React Components, Reusability, and Abstraction** (https://blog.isquaredsoftware.com/2020/06/blogged-answers-react-components-reusability-and-abstraction/)
- May 17, 2020 - **Blogged Answers: A (Mostly) Complete Guide to React Rendering Behavior** (https://blog.isquaredsoftware.com/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior/)
- May 12, 2020 - **Blogged Answers: Why I Write** (https://blog.isquaredsoftware.com/2020/05/blogged-answers-why-i-write/)
- Feb 22, 2020 - **Blogged Answers: Why Redux Toolkit Uses Thunks for Async Logic** (https://blog.isquaredsoftware.com/2020/02/blogged-answers-why-redux-toolkit-uses-thunks-for-async-logic/)
- Feb 22, 2020 - **Blogged Answers: Coder vs Tech Lead - Balancing Roles** (https://blog.isquaredsoftware.com/2020/02/blogged-answers-coder-vs-tech-lead---balancing-roles/)
- Jan 19, 2020 - **Blogged Answers: React, Redux, and Context Behavior** (https://blog.isquaredsoftware.com/2020/01/blogged-answers-react-redux-and-context-behavior/)
- Jan 01, 2020 - **Blogged Answers: Years in Review, 2018-2019** (https://blog.isquaredsoftware.com/2020/01/blogged-answers-years-in-review-2018-2019/)
- Jan 01, 2020 - **Blogged Answers: Reasons to Use Thunks** (https://blog.isquaredsoftware.com/2020/01/blogged-answers-reasons-to-use-thunks/)
- Jan 01, 2020 - **Blogged Answers: A Comparison of Redux Batching Techniques** (https://blog.isquaredsoftware.com/2020/01/blogged-answers-redux-batching-techniques/)
- Nov 26, 2019 - **Blogged Answers: Learning and Using TypeScript as an App Dev and a Library Maintainer** (https://blog.isquaredsoftware.com/2019/11/blogged-answers-learning-and-using-typescript/)
- Jul 10, 2019 - **Blogged Answers: Thoughts on React Hooks, Redux, and Separation of Concerns** (https://blog.isquaredsoftware.com/2019/07/blogged-answers-thoughts-on-hooks/)
- Jan 19, 2019 - **Blogged Answers: Debugging Tips** (https://

blog.isquaredsoftware.com/2019/01/blogged-answers-debugging-tips/)

- Mar 29, 2018 - **Blogged Answers: Redux - Not Dead Yet!** (https://blog.isquaredsoftware.com/2018/03/redux-not-dead-yet/)
- Dec 18, 2017 - **Blogged Answers: Resources for Learning Redux** (https://blog.isquaredsoftware.com/2017/12/blogged-answers-learn-redux/)
- Dec 18, 2017 - **Blogged Answers: Resources for Learning React** (https://blog.isquaredsoftware.com/2017/12/blogged-answers-learn-react/)
- Aug 02, 2017 - **Blogged Answers: Webpack HMR vs React-Hot-Loader** (https://blog.isquaredsoftware.com/2017/08/blogged-answers-webpack-hmr-vs-rhl/)
- Sep 14, 2016 - **How I Got Here: My Journey Into the World of Redux and Open Source** (https://blog.isquaredsoftware.com/2016/09/how-i-got-here-my-journey-into-the-world-of-redux-and-open-source/)

← Older (https://blog.isquaredsoftware.com/2021/05/learn-modern-redux-livestream/)

Mark Erikson

Collector of interesting links, answerer of questions

Newer → (https://blog.isquaredsoftware.com/2021/12/codebase-conversion-mean-react-next-ts/)

Recent Posts

esm-modernization-lessons/)

Presentations: Debugging JavaScript (/2023/06/presentations-debugging-javascript/)

Top Tags (All Tags) (/tags/)

redux (/tags/redux)	60
javascript (/tags/javascript)	56
react (/tags/react)	47
greatest-hits (/tags/greatest-hits)	30
presentation (/tags/presentation)	27

Greatest Hits

Greatest Hits: The Most Popular and Most Useful Posts I've Written (/2020/08/greatest-hits/)

Redux - Not Dead Yet! (/2018/03/redux-not-dead-yet/)

Why React Context is Not a "State Management" Tool (and Doesn't Replace Redux) (/2021/01/context-redux-differences/)

A (Mostly) Complete Guide to React Rendering Behavior (/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior/)

Presentations: Modern Redux with Redux Toolkit (/2022/06/presentations-modern-redux-rtk/)

When (and when not) to reach for Redux (https://changelog.com/posts/when-and-when-not-to-reach-for-redux)

The Tao of Redux, Part 1 - Implementation and Intent (/2017/05/idiomatic-redux-tao-of-redux-part-1/)

Series