Using react-testing-library in Katello

Development

John_Mitsch Katello Apr '20

Hey all,

In the new Content View page for Katello, I introduced and wrote tests using react-testing-library [1]. You can see the PR and tests here [2]. There has been some interest in this library and seeing if it is a good fit for us.

I found it to be very intuitive and I really liked how it can replicate user interaction. This is similiar to cypress.io, which we have also had discussions about introducing, but react-testing-library doesn't require the full-stack environment. A fully provisioned environment is challenging and resource-intensive to recreate in a CI server, especially for Katello.

react-testing-library is a way to get the benefits of integration testing at the PR level without needing more resources to do so. It was very easy to run the new tests along with the others in the already existing `npm test` command, and no CI changes were needed. The library also integrates nicely with jest.

The library author's philosophy is to test your React components as a user would interact with them and not to test implementation details of your code. I feel this is in line with some of the discussions around testing we have had lately.

The major benefits I see are:

- Bug and regression prevention as we test actual user workflows
- Able to test the workflows and UI interaction that we develop with UX professionals
- Worry-free refactoring as our tests are not tied to implementation details, meaning refactoring UI code will not require as much test refactoring.
- Ability to use TDD to recreate bugs with a test and iterate on the fix.

I'm not suggesting we get rid of existing enzyme snapshots, but would like to see `react-testing-library` added to the foreman-js as a development dependency and would encourage its use in new pages instead of enzyme snapshots to test user interaction. I think this will bring us more confidence in our UI code, easier and safer refactoring, and more intuitive testing.

Please let me know what you think!

1 Reply 3

Resurrection of the client-side infrastructure upgrade effort

Use foremanReact directly in plugin tests

created	last reply	replies	views	users	likes	links
Apr '20	May '20	18	593	5	18	3

John_Mitsch Katello Apr '20

Opened up a foreman-js PR here https://github.com/theforeman/foreman-js/pull/134

1 Reply

jeremylenz Katello Apr '20

(Repeating some of our IRC convo, but wanted to repost just so it's documented here.) I have played around with `react-testing-library` before and I really like it. I love the philosophy of not testing implementation details, and focusing on what the user sees. `react-testing-library` tends to make it easy to write tests that subscribe to this philosophy, and harder to write tests that don't. The result is that it guides you toward integration-style tests and away from unit tests. (It's *possible* to write unit tests; it's just harder.) But this may be hard to stomach for Foreman because we seem to focus more on unit tests and snapshot testing in our JS.

The thing with unit tests and snapshots is they are really easy to write and develop repeatable best practices around. It doesn't require much thought to say "you forgot to put a snapshot test here" or "this method needs to be unit-tested," and it makes it really easy to feel that you're doing the right thing if your test coverage number goes up. But the problem with snapshots is that they're really good at telling you if something changes, but make NO assertions that the monitored content is actually any good. And the problem with unit testing is that it tests implementation details rather than the full functionality. So you end up spending a lot of time writing passing tests but your return on investment can be pretty low.

For integration-style tests, I think `react-testing-library` is superior to Enzyme because it guides you toward this philosophy, and doesn't allow access to things like component instances which would make it easier to test implementation details. It forces you to test the actual user experience instead.

For unit tests, `react-testing-library` is possible but more difficult.

In Foreman, our *current* JS testing methodology is

- Use test helpers to generate boilerplate snapshot tests for
  - React components
  - Redux actions
  - Redux reducers
- In most cases, the component snapshot tests are considered to be enough for unit tests. For helper functions, etc. we might write some manual Jest/Enzyme unit tests.
- Write Enzyme integration tests for React pages, one big integration test per page. We also have some test helpers to help with that.

To integrate `react-testing-library`, I would be in favor of the following *new* testing methodology in Foreman:

- Replace Enzyme integration tests with `react-testing-library`. This can be a gradual process, but in the end I think it will be much better. And we don't have to have a hard-and-fast rule of "one integration test per React page," either. We can test specific behaviors focused on the user, like "this table filter works the way I expect" or "the loading spinner appears and disappears at the right time".
- Eliminate snapshots for component unit tests. They give a false sense of security, and no one ever really looks at them before updating.
- I would argue that if our user-focused "integration" tests are good and plentiful enough, very few unit tests should be required. Any that we do need can be written in `react-testing-library`.
- We could probably continue using the helpers and snapshots for Redux and other non-component stuff, simply because it's so easy. But ideally we would write tests that make actual assertions, as recommended in their best practice

Switching to a more user-centric testing philosophy will be difficult because it requires a lot more thought when writing tests. But I think in the end we'll all be better developers and Foreman JS will be much more solid.

3

tbrisker Release Manager — jeremylenz — Apr '20

> jeremylenz:
>
> Eliminate snapshots for component unit tests. They give a false sense of security, and no one ever really looks at them before updating.

Definitely agree about this point. Snapshots are hard to properly review and don't actually tell you if what was rendered is what you expected to render - just that some html elements did render.
The snapshots also have the added pain that every time some dependency updates its internal dom structure, all snapshots using that component need to be updated for the tests to pass again, even if the functionality of the component did not change, leading to developer churn and de-incentivizing upgrading components.

1 Reply — 2

tbrisker Release Manager — Apr '20

> John_Mitsch:
>
> I'm not suggesting we get rid of existing enzyme snapshots, but would like to see `react-testing-library` added to the foreman-js as a development dependency and would encourage its use in new pages instead of enzyme snapshots to test user interaction. I think this will bring us more confidence in our UI code, easier and safer refactoring, and more intuitive testing.

+1, let's add this to foreman-js and not in specific plugins.

ekohl Installer — tbrisker — Apr '20

Snapshots also make git log really hard to use. Also tools like grep to find something become much less useful. I'd love to see the snapshots disappear.

1 Reply

John_Mitsch Katello — John_Mitsch — Apr '20

Glad to hear there are other who feel the same way about snapshots and are in support of using react-testing library!

> ekohl:
>
> Snapshots also make git log really hard to use. Also tools like grep to find something become much less useful. I'd love to see the snapshots disappear.

I would like to see the same, but I think the first goal is introduce react-testing-library and use for new components to really make sure it does fits our needs. Then we can re-assess and discuss if it would be worth our time to rewrite enzyme snapshot tests with react-testing-library.

The next steps would be to get this merged to get it into foreman-js:

> John_Mitsch:
>
> Opened up a foreman-js PR here https://github.com/theforeman/foreman-js/pull/134

And then upgrade the foreman-js packages (which needs to be done for patternfly 4 anyways): https://github.com/theforeman/foreman/pull/7590

I also added more tests to my PR and was able to integrate with Redux and React-router with a wrapping method. With this, it feels like using the actual component, the only part that is mocked is external API calls. I even already found an issue in the PR and was able to quickly write a test to reproduce and use TDD to fix, so its already proving to be very useful 😃

I also like that you test from the larger parent components rather than every single component, which I can see useful for breaking up pages into smaller components and ensuring that there are no regressions. I could see this being useful with a page like Katello's Subscription page, which should be refactored into smaller components, but with enzyme shallow rendered snapshots there is not much confidence that the refactoring is regression-proof since most tests will be rewritten or re-recorded. React-testing-library allows you to refactor the child components and as long as what is shown on screen still is the same, the tests will pass and give some confidence in the refactoring's stability with little to no modification of the tests. (Just to give a real-world example)

Ron_Lavi — Apr '20

Didn't had the chance to use it yet, though I read this nice post that compares `react-testing-library` versus Enzyme from https://blog.logrocket.com/comparing-react-testing-libraries/

`react-testing-library` versus Enzyme

Among the most important considerations when writing tests for a component are your `util` functions. They may force you to write a cleaner and truer way of testing or lead you to write your tests incorrectly in terms of exported APIs.

	`react-testing-library`	Enzyme
Github	10.1K ★ 530 forks	18.3K ★ 2K forks
Shallow render	❌	✅
Full-DOM rendering	❌	✅
Fire events	✅	✅
Exported APIs	★★★★★ (easy to use)	★★★★ (complex)
Easy to adapt project	✅	❌ (requires setup/adapters)
Testing without implementation detail	✅	❌

"When writing tests for your components, don't get too bogged down in the implementation details. Remember, try to think about it from the user's perspective. This will help you produce better test suites, which will help you feel more confident about your tests."

"For most use cases, I prefer `react-testing-library` , primarily because its exported APIs do not allow you to use a component's internal API, which forces you to write better tests. In addition, there is zero configuration required."

"Enzyme, on the other hand, lets you use a component's internal API, which can include life cycle

methods or state."

"I've used both enzyme and `react-testing-libraries` in many projects. However, I've often found that `react-testing-library` makes things easier."

I agree that sometime the snapshots updates are annoying,
but since we are using a lot of libraries and while we don't have visual testing,
this is our way to see updates, and to know where to look if something broke.
IMO the snapshot is like a stump that shows how the component looked when it was created and tested.

I like the macro testing approach of `react-testing-library` and the fact that, as for now, they provide better utils to tests React hooks than Enzyme.

I wonder how well does it work for integration test with Redux.

3 Replies

ekohl Installer → Ron_Lavi Apr '20

> Ron_Lavi:
> but since we are using a lot of libraries and while we don't have visual testing,

It does require manual mental parsing of HTML and CSS. Who does that?

1 Reply

John_Mitsch Katello → Ron_Lavi Apr '20

> Ron_Lavi:
> I agree that sometime the snapshots updates are annoying,
> but since we are using a lot of libraries and while we don't have visual testing,
> this is our way to see updates, and to know where to look if something broke.
> IMO the snapshot is like a stump that shows how the component looked when it was created and tested.

I can see us using enzyme snapshots for small shared library components, such as a dropdown or button. But I think anything more complex than that (such as a table), react-testing-library would be better. And r-t-lib is the best fit for full pages. You are also getting the additional benefit of testing these small library components in the context of a larger page.

I think this would be a good compromise to have the libraries co-exist for now and we can create documentation with the expected guidelines.

After some time with r-t-lib, we could open up the discussion of replacing all enzyme tests with r-t-lib, but for now I think its too early and likely don't have the bandwidth to do so.

> Ron_Lavi:
> I wonder how well does it work for integration test with Redux.

I was able to use a wrapper to test the component with a Redux store and use the actual Redux actions in the Content View PR: see `webpack/test-utils/react-testing-lib-wrapper.js`

It works well, you don't test Redux directly, rather test the component and it's functionality while mocking the API calls that Redux makes. This reduces the amount of mocking we do and tests the whole connected page rather than the individual pieces.

1 Reply 1

tbrisker Release Manager → Ron_Lavi Apr '20

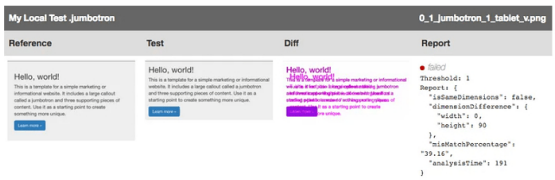
> Ron_Lavi:
> I agree that sometime the snapshots updates are annoying,
> but since we are using a lot of libraries and while we don't have visual testing,
> this is our way to see updates, and to know where to look if something broke.
> IMO the snapshot is like a stump that shows how the component looked when it was created and tested.

I think the question here is do we care if some button's internal markup changed, or if we care that pressing on it leads to the expected outcome in the page or larger component incorporating it. Do you have some examples where snapshots helped us identify a functionality change that needed us to change our code rather than a markup one?

1 Reply 2

Ron_Lavi → ekohl Apr '20

there are some libraries that uses css regression testing,
checkout this article: https://css-tricks.com/automating-css-regression-testing/, we might want to add such tests.



Ron_Lavi → John_Mitsch Apr '20

Looks interesting, I didn't see actions, reducers or integration test yet in the PR, this is where most of the logic exist, so it would be nice to see if there are helpers we can create to test those,
IMO since it's also using jest they can co-exist with Enzyme, the React community definitely recommend it so I guess we should find a good convention for it, and see how it makes things easier to test and if it cover all of our needs.

1 Reply 1

Ron_Lavi ↱ tbrisker Apr '20

I agree that the other tests are more important, but it is a good practice to keep track also on the React components and HTML markup, and in my opinion it is not that painful to have those tests and maintain existing component once in a few months or once a year.

That was about the markup snapshots,
though there are a lot of actions snapshots and integration snapshots which are more important, they enable you to actually see the process that several actions did and their impact, in this way it is very easy to debug when a test fails.

John_Mitsch Katello ↱ Ron_Lavi Apr '20

> Ron_Lavi:
>
> Looks interesting, I didn't see actions, reducers or integration test yet in the PR, this is where most of the logic exist, so it would be nice to see if there are helpers we can create to test those,
> IMO since it's also using jest they can co-exist with Enzyme, the React community definitely recommend it so I guess we should find a good convention for it, and see how it makes things easier to test and if it cover all of our needs.

Testing the actions and reducers directly is not the approach taken by react-testing-library. I would rather assure that the correct API call is made and the DOM updates as expected after clicking a button, rather than testing implementation details. I think adding enzyme snapshot tests for redux actions and reducers will bring a lot of the negatives about snapshot testing and make things less flexible, negating a lot of the benefits of using r-t-lib. For instance, if I want to refactor the redux actions and reducers, but ensure the page still functions as expected, a snapshot test won't do that for me, and I have the added work of refactoring and re-recording all the snapshot tests. Are there any examples of these snapshots preventing regressions?

There may be use cases where it makes more sense to use enzyme snapshots for redux, but for the reasons above I don't plan on adding them to the updated Content View pages, instead making sure there is comprehensive testing based on actual usage of the application.

1

Ron_Lavi ↱ John_Mitsch Apr '20

I played with it more today, and It looks pretty cool and easy to use,
my conclusion is that we should keep tests for **Selectors, Reducers & Actions** as it is now,
because it is really straight forward and doesn't need anything special.

as for the **integration tests**, I tend to agree that taking the snapshot there can be really messy if you are not sure 100% what you are trying to check... and we want to make it easier for more people to write those tests - so I think we can use r-t-lib for integration tests.

about the **component tests**, I feel that for other libraries we should take a small snapshot (less than 20 lines) just to be safe, but for our own component that we build, we could definitely use r-t-lib.

thanks `@John_Mitsch` !

2 Replies 1

Ron_Lavi Apr '20

By the way, it seem that it is possible to take a snapshot with r-t-lib with jest:

```
import React from 'react';
import { render, cleanup } from "react-testing-library";
import "jest-dom/extend-expect";
import App from './App';

afterEach(cleanup);

it("matches snapshot", () => {
  const { asFragment } = render(<App />);
  expect(asFragment()).toMatchSnapshot();
});
```

1

jeremylenz Katello ↱ Ron_Lavi Apr '20

+1 to this idea. I hadn't thought about using snapshots *only* for third-party components, but this would be quite useful to know if something changes in a JS library that we're using.

For Redux selectors/actions etc., I'm okay with snapshots because it's so easy, but I still think it's not ideal. Snapshots say only "assert that this hasn't changed since the last time I ran it", when really, the assertions should be "this action creator returns this action," "this reducer returns this new state," etc.

14 days later

John_Mitsch Katello ↱ Ron_Lavi May '20

Since this was brought up again in a UI discussion off-thread I would like to share my thoughts on the react-testing library and our current testing.

I agree with others that we should not update old tests and instead focus on our approach to the new tests. If we have the time to update old tests later, then great! But there are bigger fish to fry than updating old tests 🐟 It sounds like we are all in agreement there 🙂

> Ron_Lavi:
>
> I played with it more today, and It looks pretty cool and easy to use,
> my conclusion is that we should keep tests for **Selectors, Reducers & Actions** as it is now,
> because it is really straight forward and doesn't need anything special.

I don't see using enzyme tests for redux actions being in line with moving to react-testing-library for new pages. I still don't see the benefit of snapshots, in fact I just updated redux tests in https://github.com/theforeman/foreman/pull/7638 1 and the snapshots all passed, but my manual smoke tests revealed that a regression in the functionality. It was because of a bad import (which is now fixed) and the regression was found performing a very basic workflow for that feature.

The point of `react-testing-library` is not to test the implementation, rather test the actual

functionality of your application. If we aren't catching actual regressions in our application, then I think we should re-evaluate our testing approach. The priority shouldn't be an assurance that our implementation is working as we expect, it should be that our application is working as expected.

I've been able to do this with `react-testing-library`. In https://github.com/Katello/katello/pull/8644, I've been able to use a fully connected (to redux and react-router) component and only mock network calls. So far, I've been testing things from the top-level page, which has been really nice for refactoring, as I can split the child components up into smaller components or refactor large pieces of logic and not touch the tests. Which means the tests passing give me a lot of confidence that I haven't broken anything during a refactoring or update, and tests have easily caught the times I did break things. So far this has proven to be useful in just the PR stage, and I can see how useful it will be during development.

I also think the mental overhead of using different testing frameworks within the same large component or page is not worth it, especially for those who aren't as familiar with the React ecosystem.

I think we should still evaluate react-testing-library as it's used in the new Content View page and elsewhere, but **I am proposing that we continue to use only r-t-lib for all new pages and functionality and not continue with enzyme snapshots**. If we come across a specific area that we feel it falls short, we can evaluate that and discuss further.

3

Reply

Suggested Topics

Topic	Replies	Views	Activity
Plugin Maintainers Action Required - Request to extract strings for Foreman 3.4 Development	0	50	Jul '22
GitHub organization account cleanup Development	0	79	Sep '22
2fa enable for foreman webui Support	0	38	9d
Release team status 2022-06-29 Releases	0	97	Jun '22
Foreman release team meeting 2023-04-05 Releases	0	45	Apr 6

Want to read more? Browse other topics in Development **or view latest topics.**