Jest snapshots: useless or harmful?

November 09, 2020

Seems like Jest's "snapshot testing" is a (surprisingly) popular choice among people who write React Native apps. Let's dive into whether that's a bad idea, or a terrible idea.

What is it?

First of all, let's define what we're talking about. Snapshot testing has been a testing technique where you save a screenshot of a part of your GUI to be able to later compare it to a newer screenshot of the same thing to see how much it changed. Jest does the same thing, but instead of screenshots it just makes a copy of an internal representation of a component.

The old approach was not great. It was error-prone, half-manual, but it had a big benefit that not only it created a library that could be easily browsed to see how your app looks like at any time, but also the screenshots were relatively easy to compare (being visual files).

However, Jest's approach of taking snapshots of the underlying code representation takes away this benefit. It might make sense for web apps and HTML, but I can't really say – I don't write web code anymore. What I can say is that for mobile, it's not a good solution.

How useful are they?

The idea of automated software testing is based on defining behaviour and verifying it. And here's the biggest problem with Jest's snapshot approach – there's no place where you can define what you expect.

In behaviour driven tests you'd explain what the component is supposed to do and then verify if it actually does that. A snapshot just tells you what the component looked like before and what it looks like now. The decision whether you've fixed a bug, or introduced one, is entirely on you. That's in my opinion the biggest problem with this approach: there's no context that tells us why something changed.

In practise, on bigger PRs, you might be confonted with a big list of snapshot changed and a question from Jest: should those snapshots be updated? When you just want to finish your PR, there's a strong temptation to answer: "sure, why not". However, if you wanted to check them, there's no context there to guide you. You need to put on your detective hat and figure out for each of them what changed, why and whether that's a good thing.

But... they're easy to write!

It's a common argument to hear that the main advantage of this type of testing is that it's much easier to test this way. That's absolutely true, they're fairly easy to create. However, that cryteria is only relevant if you're comparing two approaches with similar outcomes.

Of course it takes more time to read through your component, decide what its behaviour should be, define those rules and write down a test that verifies them. But there's also infinitely more value in it.

The only benefit I can see in creating tons of snapshot tests over the app is for teams that have an artificial KPI on code coverage. It does make it easy to get to 95% coverage. However, the problem here is that code coverage is just not a useful metric of a test suite.

But... it's still better than nothing!

Well, not necessarily.

Imagine a big project when a new developer joins. Any change they make results in a cascade of snapshots to decide on whether they should be updated or not. That's not something that (being new on the team) they can confidently decide themselves and it's not easy to dump this on your colleagues neither.

That can lead to discouragement from making changes, lack of enthusiasm and ultimately worse codebase (because refactors are painful) and less productive team.

At least in case of the alternative where you have no tests you don't get the false sense of confidence (coming form 95% coverage) and you can clearly see that you're lacking them. Then it's easier to make a decision to add them responsibly.

Case for snapshot tests

To be fair, I've recently talked to a developer about an actually useful test suite that takes advantage of Jest's snapshots. It seems there is a scenario where they do make sense.

The idea is that once you've divided your components into presentational and business logic, you should still have some way of testing the presentational ones in an easy way. Usually it doesn't make much sense to write normal component tests for them, because there's usually no behaviour to test (as they're purely visual). However, we could implement a test suite for them in a following way:

- We create snapshots of all of them to be able to track when they change. This gives us information what changed when we adjusted our code. For example, when we changed a low-level component (atom) that's used in more complex, higher-level components (molecules / organisms) we'll see a list of components that were affected.
- We maintain a thorough storybook library that we can easily go through to see whether all of the affected components still look correct.

Of course, that's a partly-manual process, but it's made easier thanks to a library of Jest snapshots.

Want more?

If you liked this post, why don't you subscribe for more content? If you're as old-school as we are, you can just grab the RSS feed of this blog. If not, why don't you subscribe to our newsletter to stay in touch (fill in the form just below). We'll let you know from time to time when something interesting comes out.

Alternatively, if audio's more your thing why don't you subscribe to our podcast! We're still figuring out what it's going to be, but already quite a few episodes are waiting for you to check them out.

PS. We're hiring!

If you've read all the way till the end, it's probably a safe bet that you'd enjoy discussing topics like this one on an everyday basis. Why don't you apply to one of our open positions? We'd love to have you on our team!

WRITTEN BY
Wojciech Ogrodowczyk
Software developer

Subscribe to our newsletter

Stay in touch with us! If you subscribe, we will keep you up to date with our new adventures, learnings and insights.

If emails are not your cup of coffee, you can use RSS feed instead.

Email address

Name

I want to know more

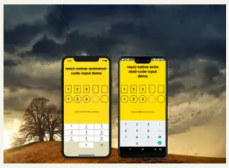
Clicking "I want to know more" you consent to processing your data by Brains & Beards sp. z o.o. for marketing purposes, including sending emails.

More Brains and Beards stories

Apple Silicon - Nerd's Christmas 🎄 was on 10.11.2020

Patryk Peszko
Co Founder

November 11, 2020



React Native Animated Code Input

Natalia Majkowska-Stewart
React and React Native developer

June 10, 2020

Contact

Brains & Beards sp. z o.o.
VAT ID: PL9721298520
smile@brainsandbeards.com

Sitemap

About us
Team
Case studies
Blog
Careers
Contact us

Services

Native iOS and Android
Cross-platform apps
Staff augmentation

If you're a bit of a technical nerd (like us!) you can keep on reading. This site has been built in React, using GatsbyJS. The data layer is managed through GraphQL. The whole thing is hosted using Netlify. All of those products are great and we're so happy to be able to use them ❤️💖💞