# Snapshot Testing for Frontends

Test whether any characteristics of your application have changed.

Viduni Wickramarachchi · Follow
Published in Bits and Pieces · 5 min read · Jul 22, 2021




There are many types of software testing. Among these, one of the unique types of testing methods is Snapshot testing.

> Snapshot tests assert that the current output is same as the output before.

The main difference between snapshot testing and functional/unit tests is, snapshot tests never assert the correct behavior of the application functionality but does an output comparison instead.

There are two types of tools that support frontend snapshot testing.

**Tools that take snapshots of serializable data**

Jest is a tool that has built-in support for snapshot testing. Testing React components are highly supported through this. Further, Cypress supports snapshot testing via plugins (@cypress/snapshot)

**Tools that take visual snapshots**

Intern (via visual plugin), Jest-Image-Snapshot plugin, Applitools (for visual AI testing) are examples of tools that take visual snapshots.

In this article, I will explore Snapshot testing with examples of it using Jest. Basic prior knowledge about Jest would be an added benefit to grasp the concepts in this article.

. . .

## How does Snapshot testing work?

As the name suggests, Snapshot testing records take a snap of the system. In Jest, this would be a render tree. Then it compares the recorded snapshot in future executions.

An example Snapshot test in Jest would be as follows.

```
it('renders list with one row', async () => {
   const fetchProductList = jest.fn(() => {
      new Promise(resolve => resolve(data)));
   }

   const wrapper = mount(
      <ProductsComponent fetchProductList={fetchProductList} />
   );

   wrapper.update();
   expect(wrapper).toMatchSnapshot();
});
```

At the first execution, the `toMatchSnapshot()` function saves the snapshot of the particular component to a file. This will have the .snap extension. In subsequent executions, it will compare the component render tree with that snapshot and fail if there are any mismatches.

When this testing mechanism is followed, you don't have to check for each element in the DOM tree to check whether they are rendered correctly.

> With Snapshot testing, the number of lines written for a test will be significantly less.

However, suppose you introduce a new element to your component that will change the DOM tree. In that case, you need to update your snapshot to overwrite the stale snapshot. In Jest, it's pretty straightforward with the following command.

```
jest --updateSnapshot
```

This is much easier than maintaining unit tests to check for elements in the DOM tree if done right. I will discuss more in the coming sections.

. . .

### Benefits of Snapshot testing

Let's have a look at few benefits of snapshot testing for our frontends.

- It introduces a clear pattern for testing component render trees.
- Faster and fewer lines of code than checking for component elements via unit tests —If you write the same in unit tests, you have to look for specific elements and values.

```
// Snapshot
it('renders list with one row', async () => {
    const fetchProductList = jest.fn(() => new Promise(resolve => resolve(data)));
    const wrapper = mount(<ProductsComponent fetchProductList={fetchProductList}/>);
    wrapper.update();

    expect(wrapper).toMatchSnapshot();
});


// Traditional way
it('renders list with one row without snapshot', async () => {
    const fetchProductList = jest.fn(() => new Promise(resolve => resolve(data)));
    const wrapper = mount(<ProductsComponent fetchProductList={fetchProductList}/>);
    wrapper.update();

    expect(wrapper.find('h1').length).toBe(1);
    expect(wrapper.find('h1').text()).toBe('List of customers');
    expect(wrapper.find('button').length).toBe(1);
    expect(wrapper.find('button').text()).toBe('View customer');
    expect(wrapper.find('ul').children().length).toBe(5);
});
```

snapshot-test.js hosted with ♥ by GitHub view raw

- Easier to update when the component changes.
- Easier to view code changes.
- Allows conditional rendering tests.
- Allows checking how components behave once you pass various combinations of props to them — Helps to validate if the passed data is properly reflected in the component.

You can combine it with other testing techniques for the best results while finding the right balance between them.

### Limitations of Snapshot testing

Some argue whether snapshot testing is a curse. Well, like any other technology, this has a few drawbacks to it as well. Let's find out.

- **A large snapshot is of no use** — If your snapshot is over 1000 lines, it's humanly challenging to compare the changes. Therefore, snapshot testing is suitable only for small snapshots. However, there are ways to overcome the creation of large snapshots. One solution to this would be using 'shallow' rendering and testing only the component in question.
- **Multi-language apps may run into issues** — When a translation changes, there is a risk of snapshot tests failing even if the source code hasn't changed. Suppose you are using `react-intl` library for translations. In that case, one solution to this is to mock `react-intl` so that it always returns the ID of translation, not its value.
- **Conflict resolution can be a major hassle** — Resolving merge conflicts in snapshots is a cumbersome task, especially if the snapshot is considerably large.

Apart from the above, I have come across another common issue. At first many developers tend to blindly update the snapshots when they come across a failing test.

> However, blindly updating snapshots should never be done.

It will bypass issues in the component render tree due to source code updates, making the tests useless.

### Summary

Snapshot testing is a handy tool to make sure your UI does not change unexpectedly. However, this should not be considered as a silver bullet for frontend testing. You should know when to use them effectively.

Even though snapshot tests are easier to create and maintain, you should remember that it is not a replacement for the unit or functional tests. Snapshot tests only verify that your application hasn't changed, not that it's working correctly.

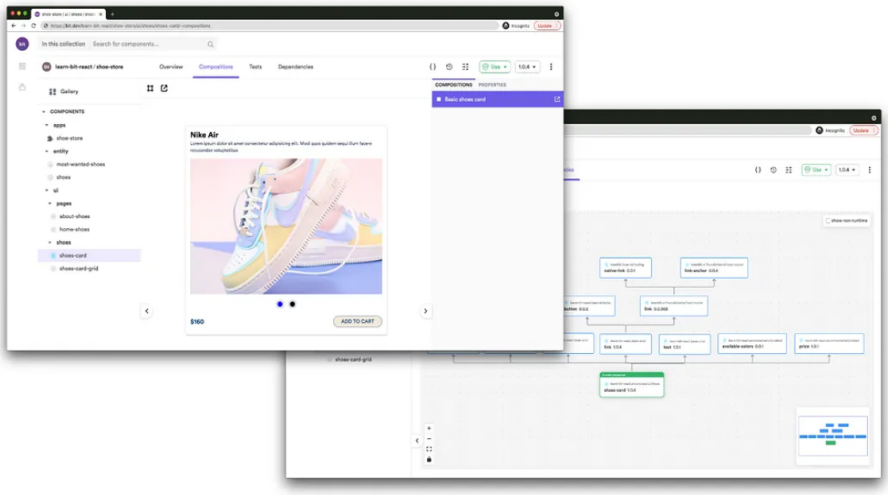
Let me know your thoughts once you use snapshot testing in your applications too. Thanks for reading!

. . .

### Build with independent components, for speed and scale

Instead of building monolithic apps, build independent components first and compose them into features and applications. It makes development faster and helps teams build more consistent and scalable applications.

OSS Tools like Bit offer a great developer experience for building independent components and composing applications. Many teams start by building their Design Systems or Micro Frontends, through independent components.
Give it a try →




An independently source-controlled and shared "card" component (on the right -> its dependency graph, auto-generated by Bit)



## Learn More

blog.bitsrc.io

**Testing React Components with Jest and Enzyme- In Depth**

Everything you should know about testing React components using Jest & Enzyme. With examples.

blog.bitsrc.io

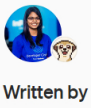
**End to End Testing React Apps With Cypress**

How to run End-To-End testing on React apps with Cypress.

blog.bitsrc.io

Frontend Testing | Testing | Frontend Development | JavaScript | React

278



## Written by Viduni Wickramarachchi

908 Followers · Writer for Bits and Pieces

Follow

Software Engineer | Tech Enthusiast | Co-Founder — 22UponTwo

### More from Viduni Wickramarachchi and Bits and Pieces


Viduni Wickramarachchi in Bits and Pieces

**The BFF Pattern (Backend for Frontend): An Introduction**

Get to know the benefits of using BFF pattern in practice

8 min read · Feb 23, 2021

2.8K 18


Anto Semeraro in Bits and Pieces

**JavaScript Optimization Techniques for Faster Website...**

Master JavaScript optimization to enhance website performance: minimize file sizes,...

16 min read · Apr 14

515 3

Fernando Doglio in Bits and Pieces

**Node.js Just Released Version 20! WTH?!**

Let's see what's new with Node v20!

8 min read · Apr 24

507 4

Viduni Wickramarachchi in Bits and Pieces

**Github Actions or Jenkins? Making the Right Choice for You**

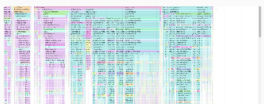
GitHub Actions and Jenkins both get the job done. Let's find out whether it's worth...

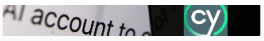
6 min read · Dec 9, 2020

701 6

See all from Viduni Wickramarachchi | See all from Bits and Pieces

## Recommended from Medium




Miktad Öztürk

### ChatGPT + Cypress Tests

In this article, I would like to tell you how I made the Cypress tests by asking ChatGPT.

2 min read · Jan 18

60 2

Steven Lemon in Bits and Pieces

### Why Is My Jest Suite So Slow?

The simple mistake undermining Jest's performance

12 min read · Jan 11

228 3

## Lists

**Stories to Help You Grow as a Software Developer**

19 stories · 14 saves

**Staff Picks**

289 stories · 55 saves

Shuvo Habib

### Monorepo: Lerna Project Setup—React and Node.js Application...

In the last few years, Monorepo has gained considerable adoption. People are discussin...

6 min read · Feb 2

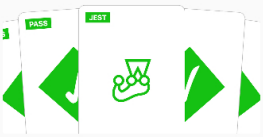
1

Steven Lemon in JavaScript in Plain English

### Optimizing Jest for Faster CI Performance with GitHub Actions

7 min read · Jan 1

9 2




Razvan L in Dev Genius

### Write Unit Tests with Jest in Node.js

Jest is a clean and concise JavaScript testing framework developed and maintained by...

4 min read · Jan 5

56


Rebai Ahmed in Level Up Coding

### Write CRUD E2E tests with Angular and Cypress

Introduction:

4 min read · Nov 17, 2022

197

See more recommendations