Dominik Kriese & Andre Wruszczak 09. April 2021 4 min

When Snapshot Testing will make you snap

In modern frontend development, there is a rise of component based frameworks and libraries such as React and Angular. And with them came new ways of testing Web-UI-Code. The most well known example is snapshot testing. It grew within the React community and offers a lot of benefits.

However, today we want to focus on the things that can go wrong. So let us first have a brief look at what the method is. Afterwards we will see how it could undermine your testing efforts.

What is snapshot testing

Snapshot testing is a term used to describe approval testing for the frontend community. Or is it the other way round? Anyways: The basic idea is to take pieces of plain text and compare them to a stored baseline.

Frontend developers love this technique. And for good reason. With little to no effort, we can write tests, that do a multitude of things in regards to verifying our DOM and data structures. A snapshot test could look like this, with the resulting snapshot below.

```
it('should style a button', () => {
    const callback = jest.fn();
    const button = mount(<Button label={'Button'} clickHandler={callback}/>)
        .find('button');
    expect(button).toMatchSnapshot();
});
```

```
exports[`Button should style a button 1`] = `
.c0 {
  background-color: #0f7956;
  border: none;
  border-radius: 4px;
  padding: 6px 24px;
  color: white;
  font-weight: bold;
  float: right;
  margin-top: 16px;
}

.c0:hover {
  background-color: #21ae80;
  cursor: pointer;
}

<button
  className="c0"
  onClick={[MockFunction]}
>
  Button
</button>
`;
```

One thing stands out: very little lines of test code validate both, our CSS and HTML code. That looks awesome and saves time. But be aware of the hidden traps.

Snapshot Testing can break your teams discipline

What happens a lot, when developers get excited for snapshot testing is clear. They over use it. But how can this happen? The answer is simple. Lets asume you have a component or page like the following and use a snapshot test

Related posts


Kotlin: How to Do Higher-Order Functions
Read more →


Kotlin Assertion Libraries - Conclusions
Read more →

Kotlin Assertion Libraries - Kluent
Read more →


Kotlin Assertion Libraries - Atrium
Read more →

to verify it.

```
const Application = styled.div`
  background-color: #f3f3f3;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  color: #333;
  text-align: center;
`

const App = () => {

    const defaultText = "Enter some text";
    let initialState = localStorage.load();
    const [label, setLabel] = useState(initialState ?? defaultText);
    const [input, setInput] = useState('');


    const updateLabel = () => {
        setLabel(!!input ? input : defaultText);
        setInput("");
        localStorage.save(input);
    }

    const inputLabel = 'Enter your text';
    return (
        <Application data-testid={'application'}>
            <Container>
                <Headline text={label}/>
                <TextInput value={input}
                           changeHandler={setInput}
                           label={inputLabel}
                           id={'text'}/>
                <Button label={'UPDATE'} clickHandler={updateLabel}/>
            </Container>
        </Application>
    );
}

export default App;
```

As you might assume, the used components have some code on their own. A snapshot of this will result in a wall of text. This leads to three main problems. First, the test will fail with every change in our application. Second, it will never fail alone, since we hopefully test our components as well. And third: think about a project deadline. Would you really read the following snapshot?

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`Application should never be tested as a snapshot 1`] = `
.c4 {
  display: block;
  padding: 6px 12px;
  border-bottom: 1px solid #333;
  border-top: none;
  border-right: none;
  border-left: none;
  background-color: #f3f3f3;
  font-size: 1.25rem;
  width: 100%;
  box-sizing: border-box;
}

.c4:focus {
  border-bottom: 2px solid #0f7956;
  padding-bottom: 5px;
}

.c3 {
  display: block;
  margin-top: 16px;
  font-size: 0.75rem;
  background-color: #f3f3f3;
  padding: 4px 12px 0;
  width: 100%;
  text-align: left;
  box-sizing: border-box;
}

.c2 {
  text-align: left;
  font-size: 2.5rem;
}

.c5 {
  background-color: #0f7956;
  border: none;
  border-radius: 4px;
  padding: 6px 24px;
  color: white;
  font-weight: bold;
  float: right;
  margin-top: 16px;
}

.c5:hover {
  background-color: #21ae80;
  cursor: pointer;
}

.c1 {
  background-color: #eaeaea;
  border-radius: 4px;
  padding: 32px;
  -webkit-box-shadow: 2px 2px 5px 0px rgba(0,0,0,0.25);
  -moz-box-shadow: 2px 2px 5px 0px rgba(0,0,0,0.25);
```

```
  box-shadow: 2px 2px 5px 0px rgba(0,0,0,0.25);
}

.c0 {
  background-color: #f3f3f3;
  min-height: 100vh;
  display: -webkit-box;
  display: -webkit-flex;
  display: -ms-flexbox;
  display: flex;
  -webkit-flex-direction: column;
  -ms-flex-direction: column;
  flex-direction: column;
  -webkit-align-items: center;
  -webkit-box-align: center;
  -ms-flex-align: center;
  align-items: center;
  -webkit-box-pack: center;
  -webkit-justify-content: center;
  -ms-flex-pack: center;
  justify-content: center;
  color: #333;
  text-align: center;
}

<App>
  <styled.div
    data-testid="application"
  >
    <div
      className="c0"
      data-testid="application"
    >
      <Container>
        <styled.div>
          <div
            className="c1"
          >
            <Headline
              text="Enter some text"
            >
              <styled.h1>
                <h1
                  className="c2"
                >
                  Enter some text
                </h1>
              </styled.h1>
            </Headline>
            <TextInput
              changeHandler={[Function]}
              id="text"
              label="Enter your text"
              value=""
            >
              <styled.label
                htmlFor="text"
              >
                <label
                  className="c3"
                  htmlFor="text"
                >
                  Enter your text
                </label>
              </styled.label>
              <styled.input
                id="text"
                onChange={[Function]}
                type="text"
                value=""
              >
                <input
                  className="c4"
                  id="text"
                  onChange={[Function]}
                  type="text"
                  value=""
                />
              </styled.input>
            </TextInput>
            <Button
              clickHandler={[Function]}
              label="UPDATE"
            >
              <styled.button
                onClick={[Function]}
              >
                <button
                  className="c5"
                  onClick={[Function]}
                >
                  UPDATE
                </button>
              </styled.button>
            </Button>
          </div>
        </styled.div>
      </Container>
    </div>
  </styled.div>
</App>
`;
```

Did you notice the buttons color? I didn't. Most developers I know would be confident enough to trust their capabilities. They would enter `jest -u` and "solve" the problem at hand. And thats not even 200 lines for our small sample application. Imagine real world pages or templates, rendering up to thousands of lines.

While the first two points are just annoying, but not a big deal, the last is bad enough to cause serious headaches. At

this point, the usefulness of your snapshot tests relies completely on the discipline of your team! As did the value of all of your tests, since those snapshots make it hard to find real problems. And it will happen, that someone updates the snapshots without even realizing, that there is a problem within another component.

Is Snapshot Testing bad?

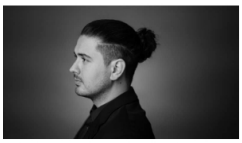
No! Snapshot Testing is awesome. I love it. The moment i discovered it, I started to save insane amounts of time. All I am saying is: Use the right tool for the job. If you combine multiple small or medium sized components to a template or a page, snapshot testing is often the wrong choice. You could either use visual regression for that purpose or just check if all your components are displayed individually.

Are you looking at a small component, leading to a snapshot with less than 100 lines? Use it. It is awesome! And never forget: It is more than just a tool for validating DOM integrity. Do you have an algorithm producing data structures? Snapshot testing can be useful as well. But always remember: If you can not read the snapshot within half a minute, consider using something else.

Some final words

Personally, I am done talking about snapshot testing for today. But if you want to learn more about testing for your Frontend Code, stay with us. We will cover several other topics in the near future. And one important thing before I am done: leave a comment. I'd love to learn from your experience!

Authors



Dominik Kriese →

Andre Wruszczak →

Comment article

First name, name*

Email address*

Your comment*

I have read the privacy policy and agree.

Anti-Robot Verification
Click to start verification
FriendlyCaptcha

Send Comment

Recent posts

February 2023 | Tioman Gally

May 2022 | Alexander Miller & Sebastian Letzel

May 2022 | Sebastian Letzel & Alexander Miller

Kotlin: How to Do Higher-Order Functions

Read more →

Kotlin Assertion Libraries - Conclusions

Read more →

Kotlin Assertion Libraries - Kluent

Read more →

December 2021 | Alexander Miller & Sebastian Letzel

Kotlin Assertion Libraries - Atrium

Read more →

August 2021 | Katharina Hersztowski

Digital Design - User-centered Product Development

Read more →

July 2021 | Sebastian Letzel & Alexander Miller

Kotlin Assertion Libraries - Strikt

Read more →

NOVATEC

Novatec Consulting GmbH
Bertha-Benz-Platz 1
D-70771 Leinfelden-Echterdingen

+49 711 22040-700
info@novatec-gmbh.de

GEMEINWOHL UNTERNEHMEN

Follow us on social media