# Testing Components

There is no required library for testing Fela components. Since Fela is framework-agnostic, you may use any javascript testing library to test your components. Fela's source is already fully tested ↗, so your tests should only confirm that Fela's styles are interacting correctly with the components.

## Testing Rules

The most straight-forward way is to test our rules directly. This is the same regardless of which framework and bindings are used.
Rules are just pure functions. Eric Elliot, a respected Javascript instructor in the industry, defines pure functions for us ↗. A pure function is a function which:

- Given the same input, will always return the same output.
- Produces no side effects.
- Relies on no external mutable state.

Since our rules will follow this definition, our tests can simply ensure our rules *output the correct styles for every input*.

## Snapshot Testing

While testing pure rules is a nice way to ensure the styles are correctly computed, the benefits are quite limited. Instead, we should test the integration with our actual components.
The best and most simple method to do so is snapshot testing ↗. We will use Jest ↗ to demonstrate that, but any other library to supports something similar to snapshot testing with just work fine as well.

A basic snapshot test for a React component looks something like this:

```
import * as React from 'react'
import renderer from 'react-test-renderer'

import MyComponent from './MyComponent'

describe('<MyComponent />', () => {
  it('renders correctly', () => {
    const tree = renderer.create(<MyComponent>Hello</MyComponent>)
    const snapshot = tree.toJSON()
    expect(snapshot).toMatchSnapshot()
  })
})
```

### CSS Snapshots

The problem is that this snapshot only includes the rendered HTML markup as of now which only includes the class name references e.g.

**Snapshot**

```
<div class="a b c d">Hello</div>
```

It **doesn't** include the rendered CSS output and thus is not sufficient for us to also test our styles accordingly.
Luckily, there's yet another package to achieve that in seconds!

## jest-react-fela

The jest-react-fela ↗ package exports a single createSnapshot function.

> **Note:** It is also available for Preact ↗ and Inferno ↗.

We can replace our `react-test-renderer` code with a simple createSnapshot call to get a reasonable snapshot with HTML and CSS markup.

```
import * as React from 'react'
import { createSnapshot } from 'jest-react-fela'

import MyComponent from './MyComponent'

describe('<MyComponent />', () => {
  it('renders correctly', () => {
    const snapshot = createSnapshot(<MyComponent>Hello</MyComponent>)
    expect(snapshot).toMatchSnapshot()
  })
})
```

The snapshot might now look something like this:

**Snapshot**

```
.a {
  padding-left: 10px
}

.b {
  color: red
}

.c {
  font-size: 16px
}

@media (min-width: 1024px) {
  .d {
    padding-left: 15px
  }
}

<div class="a b c d">Hello</div>
```

### Passing a Theme

createSnapshot accepts a couple of additional arguments.
Probably the most important one is the second argument which accepts a custom theme to be passed to the component.
If your component renders styles depending on a theme, it will now automatically pick the passed one.

> Chek the API reference for more information on all accepted arguments.

```
import * as React from 'react'
import { createSnapshot } from 'jest-react-fela'

const theme = {
  colors: {
    primary: 'red',
    secondary: 'blue',
  },
  breakpoints: {
    small: '@media (min-width: 480px)',
    large: '@media (min-width: 1024px)',
  },
}

describe('<MyComponent />', () => {
  it('renders correctly', () => {
    const snapshot = createSnapshot(<MyComponent>Hello</MyComponent>, theme)
    expect(snapshot).toMatchSnapshot()
  })
})
```