



# React Hooks & & testing practices

Hi 🖐️

my name is  
**Konstantinos**

const [👤, 📺, 🇬🇷, 🇬🇧, ☕, 🏔️] = 👋





# What is the talk about?

- State of react testing
- First pitfalls
- How to solve them?
- What did I learn?
- DEMO




# 2018

Working with React is great 🎉

Finally I can understand how to test React Components 🙌

# 2019

# React Hooks released

This new feature looks great, I'll turn all my components to React Hooks! 

# Later in 2019...

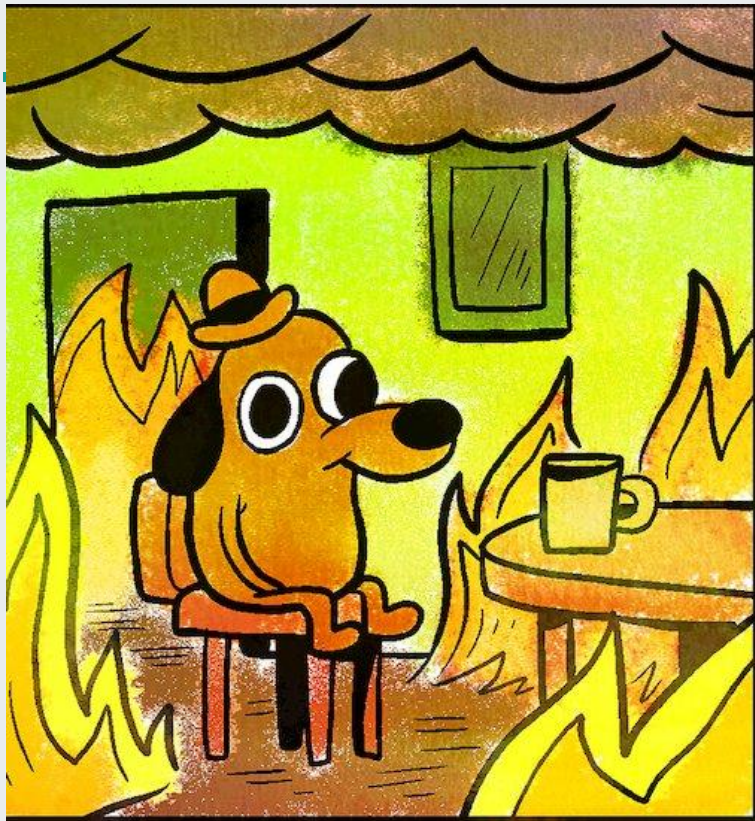
But how can I test React hooks?





# Jest + Enzyme

First attempt to test React hooks





FAIL src/ToDoList.spec.js

- ToDoList › renders

Invariant Violation: Hooks can only be called inside the body of a function component.

```
3 |  
  4 | export const ToDoList = () => {  
  > 5 |   const [todos, setTodos] = useState([
```

# What's this error?

Is it my code?

Is it hooks?

Is it enzyme?

Let's try to understand what's going on under the hood



# Let's take a step back

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
    this.increaseCount = this.increaseCount.bind(this);
  }

  componentDidMount() {
    this.setState({ count: 30 });
  }

  increaseCount() {
    this.setState(({ count }) => ({
      count: count + 1,
    }));
  }

  render() {
    return (
      <button className="button" onClick={this.increaseCount}>
        {this.state.count}
      </button>
    );
  }
}
```



```
function Counter() {
  const [count, setCount] = React.useState(0);

  React.useEffect(() => {
    setCount(30);
  }, []);

  function increaseCount() {
    setCount((count) => count + 1);
  }

  return (
    <button className="button" onClick={increaseCount}>
      {count}
    </button>
  );
}
```

Invariant Violation: Hooks can only be called inside the body of a function component.



```
it("increaseCount updates the count state", () => {  
  const wrapper = shallow(<Counter />);  
  expect(wrapper.state("count")).toBe(30);  
  wrapper.instance().increaseCount();  
  expect(wrapper.state("count")).toBe(31);  
});
```



```
it("increaseCount updates the count state", () => {  
  const wrapper = shallow(<Counter />);  
  expect(wrapper.find("button").prop("children")).toEqual(30);  
  wrapper.find("button").simulate('click');  
  expect(wrapper.find("button").prop("children")).toEqual(31);  
});
```

Invariant Violation: Hooks can only be called inside the body of a function component.





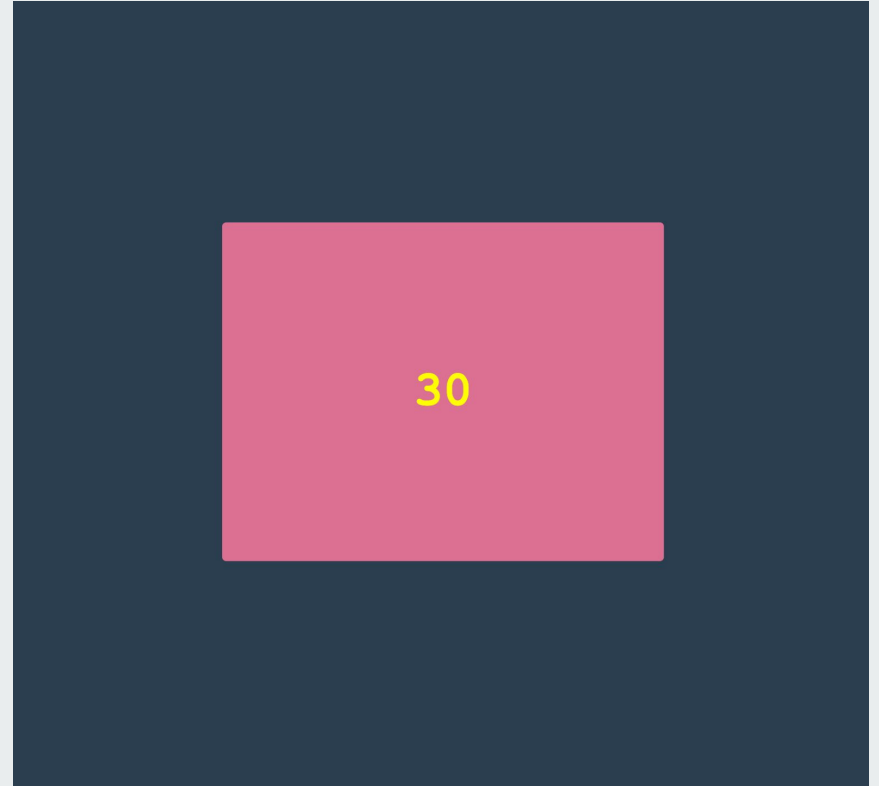
# React Testing Library



```
it("button click increases the count by 1", () => {  
  const { getByRole } = render(<Counter />);  
  expect(getByRole("button").textContent).toEqual("30");  
  fireEvent.click(getByRole("button"));  
  expect(getByRole("button").textContent).toEqual("31");  
});
```



# Example



# useState



```
function Counter() {  
  const [count, setCount] = React.useState(0);  
  return (  
    <button className="button" onClick={() => setCount((count) => count + 1)}>  
      {count}  
    </button>  
  );  
}
```

# useState



```
it("should initialize count to 0", () => {  
  const { getByRole } = render(<Counter />);  
  expect(getByRole("button").textContent).toEqual("0");  
});  
  
it("should increase count by 1", () => {  
  const { getByRole } = render(<Counter />);  
  fireEvent.click(getByRole("button"));  
  expect(getByRole("button").textContent).toEqual("1");  
});
```

# useEffect

```
function Counter() {  
  const [count, setCount] = React.useState(0);  
  
  React.useEffect(() => {  
    setCount(30);  
  }, []);  
  
  return (  
    <button className="button" onClick={() => setCount((count) => count + 1)}>  
      {count}  
    </button>  
  );  
}
```

# useEffect



```
it("should initialize count to 30", () => {  
  const { getByRole } = render(<Counter />);  
  expect(getByRole("button").textContent).toEqual("30");  
});
```

```
it("should increase count by 1", () => {  
  const { getByRole } = render(<Counter />);  
  fireEvent.click(getByRole("button"));  
  expect(getByRole("button").textContent).toEqual("31");  
});
```

# useCounter

```
function Counter() {  
  const [count, setCount] = useCounter();  
  
  return (  
    <button className="button" onClick={() => setCount((count) => count + 1)}>  
      {count}  
    </button>  
  );  
}  
  
function useCounter() {  
  const [count, setCount] = React.useState(0);  
  
  React.useEffect(() => {  
    setCount(30);  
  }, []);  
  
  return [count, setCount];  
}
```



# useCounter




```
it("should initialize count to 30", () => {  
  const { getByRole } = render(<Counter />);  
  expect(getByRole("button").textContent).toEqual("30");  
});  
  
it("should increase count by 1", () => {  
  const { getByRole } = render(<Counter />);  
  fireEvent.click(getByRole("button"));  
  expect(getByRole("button").textContent).toEqual("31");  
});
```

# useReducer

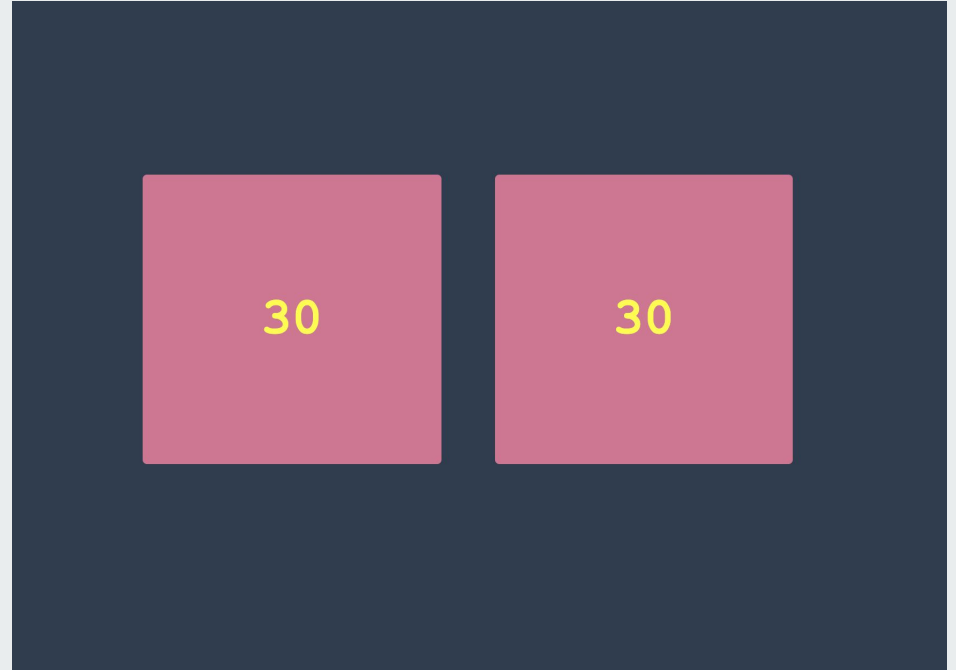
```
function Counter() {  
  const [state, dispatch] = React.useReducer(reducer, { count: 30 });  
  
  return (  
    <button className="button" onClick={() => dispatch({ type: "increment" })}>  
      {state.count}  
    </button>  
  );  
}  
  
function reducer(state, action) {  
  switch (action.type) {  
    case "increment":  
      return { count: state.count + 1 };  
    case "decrement":  
      return { count: state.count - 1 };  
    default:  
      throw new Error();  
  }  
}
```

# useReducer

```
it("should initialize count to 30", () => {  
  const { getByRole } = render(<Counter />);  
  expect(getByRole("button").textContent).toEqual("30");  
});  
  
it("should increase count by 1", () => {  
  const { getByRole } = render(<Counter />);  
  fireEvent.click(getByRole("button"));  
  expect(getByRole("button").textContent).toEqual("31");  
});
```



# Example update



# useContext

```
const CounterContext = React.createContext();

function CounterContainer() {
  const [count, setCount] = React.useState(30);
  return (
    <CounterContext.Provider value={{ count, setCount }}>
      <ul>
        <li>
          <Counter />
        </li>
        <li>
          <Counter />
        </li>
      </ul>
    </CounterContext.Provider>
  );
}
```

# useContext

```
function Counter() {  
  const { count, setCount } = React.useContext(CounterContext);  
  
  return (  
    <button className="button" onClick={() => setCount((count) => count + 1)}>  
      {count}  
    </button>  
  );  
}
```

# useContext



```
it("should initialize count to 30", () => {  
  const { getAllByRole } = render(<CounterContainer />);  
  const buttons = getAllByRole("button");  
  expect(buttons[0].textContent).toEqual("30");  
  expect(buttons[1].textContent).toEqual("30");  
});
```

# useContext

```
it("should increase count by 1", () => {  
  const { getAllByRole } = render(<CounterContainer />);  
  
  fireEvent.click(getAllByRole("button")[0]);  
  
  let buttons = getAllByRole("button");  
  expect(buttons[0].textContent).toEqual("31");  
  expect(buttons[1].textContent).toEqual("31");  
  
  fireEvent.click(getAllByRole("button")[1]);  
  
  buttons = getAllByRole("button");  
  expect(buttons[0].textContent).toEqual("32");  
  expect(buttons[1].textContent).toEqual("32");  
});
```





# Bonus: useContext & sideEffect

```

function CounterContainer() {
  const [count, setCount] = React.useState(0);
  const [fetchStatus, setFetchStatus] = React.useState(FETCH_STATUS.LOADING);

  React.useEffect(() => {
    (async function () {
      try {
        const countFromAPI = await fetchCount();
        setCount(countFromAPI);
        setFetchStatus(FETCH_STATUS.SUCCESS);
      } catch (e) {
        setFetchStatus(FETCH_STATUS.FAIL);
      }
    })();
  }, []);

  return (
    <CountContext.Provider value={{ count, setCount }}>
      {fetchStatus === FETCH_STATUS.LOADING && <p>Counter is loading...</p>}&&
      {fetchStatus === FETCH_STATUS.SUCCESS && (
        <ul>
          <li>
            <Counter />
          </li>
          <li>
            <Counter />
          </li>
        </ul>
      )}
    </CountContext.Provider>
  );
}

```




```
function Counter() {  
  const { count, setCount } = React.useContext(CounterContext);  
  
  return (  
    <button className="button" onClick={() => setCount((count) => count + 1)}>  
      {count}  
    </button>  
  );  
}
```



```
jest.mock("./fetchCount", () => ({  
  fetchCount: () => new Promise(resolve =>  
    setTimeout(() => resolve(44), 1000)  
  }),  
}));
```



```
it("should load with initial count fetched", async () => {  
  const { getByText, getAllByRole } = render(<CounterContainer />);  
  
  expect(getByText(/is loading/)).toBeInTheDocument();  
  
  await waitForElementToBeRemoved(() => getByText(/is loading/i));  
  
  const buttons = getAllByRole("button");  
  expect(buttons[0].textContent).toEqual("44");  
  expect(buttons[1].textContent).toEqual("44");  
});
```



```
it("should increase count by 1 when clicking on any button", async () => {
  const { getByText, getAllByRole } = render(<CounterContainer />);

  await waitForElementToBeRemoved(() => getByText(/is loading/i));

  fireEvent.click(getAllByRole("button")[0]);

  let buttons = getAllByRole("button");
  expect(buttons[0].textContent).toEqual("45");
  expect(buttons[1].textContent).toEqual("45");

  fireEvent.click(getAllByRole("button")[1]);

  buttons = getAllByRole("button");
  expect(buttons[0].textContent).toEqual("46");
  expect(buttons[1].textContent).toEqual("46");
});
```



# Wrap Up

- Testing with Confidence
- No implementation details
- Result of the hooks
- Effortless refactoring



<https://github.com/leimonio/react-hooks-testing>



# Thank you!



hey@leimon.io



<https://leimon.io/>



<https://github.com/leimonio>



<https://twitter.com/leimonio>