**Some of the insights for better development, scalability and durability of a REACT project.**

Author- Venkat Avinash Karra

**Contents:**

* Introduction (DRY)
* Redux and Sagas
* Adaptive Styling & and VS Responsive Styling
* Scalability & Maintainance
* Folder Structure & for API ?
* Why MaterialUI ?
* Units (px, em, rem)
* Thinking in React
* Future Insights (PWA)

Introduction

What is clean code, and why do I care?

Clean code is a consistent style of programming that makes your code easier to write, read, and maintain. Often a developer spends time on a problem, and once the problem is solved, they make a pull request. I contend that you aren’t done just because your code “works.”

Now is your chance to clean it up by removing dead code (zombie code), refactoring, and removing any commented-out code! Strive for maintainability. Ask yourself, “Will someone else be able to understand these code six months from now?”

In simpler terms, write code that you would be proud to take home and show your mother.

Why do you care? Because if you’re a good developer, you’re lazy. Hear me out – I mean that as a compliment. A good developer, when faced with a situation where they must do something more than once, will generally find an automated (or better) solution to complete the task at hand. So, because you’re lazy, subscribing to clean-code techniques will decrease the frequency of changes from pull-request code reviews and the need to come back to the same piece of code over and over.

Clean code passes the “smell test”

Clean code should pass the smell test. What do I mean by that? We’ve all looked at code (our own or others’) and said, “Something’s not quite right here.” Remember, if it doesn’t feel right, it probably isn’t. Code that’s well thought out just comes together. If it feels like you’re trying to fit a square peg into a round hole, then pause, step back, and take a break. Nine times out of 10, you’ll come up with a better solution.

Clean code is DRY

DRY is an acronym that stands for “Don’t Repeat Yourself.” If you are doing the same thing in multiple places, consolidate the duplicate code. If you see patterns in your code, that is an indication it is prime for DRYing. Sometimes this means standing back from the screen until you can’t read the text and literally looking for patterns.

Occasionally, DRYing your code may actually increase code size. However, DRYing your code also generally improves maintainability. Be warned that it’s possible to go too far with DRYing up your code, so know when to say when.

Naming things

We should all give serious thought to variable names, function names, and even filenames.

**Here are a few guidelines**:

* Boolean variables, or functions that return a boolean value, should start with “is,” “has” or “should.”

Eg:

// Dirty

const done = current >= goal;

// Clean

const isComplete = current >= goal;

* Functions should be named for what they do, not how they do it. In other words, don’t expose details of the implementation in the name. Why? Because how you do it may change some day, and you shouldn’t need to refactor your consuming code because of it. For example, you may load your config from a REST API today, but you may decide to bake it into the JavaScript tomorrow.

// Dirty

const loadConfigFromServer = () => {

...

};

// Clean

const loadConfig = () => {

...

};

**Here are some best practices to follow when architecting your React applications:**

* Use small functions, each with a single responsibility. This is called the single responsibility principle. Ensure that each function does one job and does it well. This could mean breaking up complex components into many smaller ones. This also will lead to better testability.
* Be on the lookout for leaky abstractions. In other words, don’t impose your internal requirements on consumers of your code.
* Follow strict linting rules. This will help you write clean, consistent code.

Clean code doesn’t (necessarily) take longer to write

I hear the argument all the time that writing clean code will slow productivity. That’s a bunch of hooey. Yes, initially you may need to slow down before you can speed up, but eventually your pace will increase as you are writing fewer lines of code.

And don’t discount the “rewrite factor” and time spent fixing comments from code reviews. If you break your code into small modules, each with a single responsibility, it’s likely that you’ll never have to touch most modules again. There is time saved in “write it and forget it.”

Practical examples of dirty code vs. clean code

DRY up this code

Look at the code sample below. Go ahead and step back from your monitor as I described above. Do you see any patterns? Notice that the component **Thingie** is identical to **ThingieWithTitle** with the exception of the **Title** component. This is a perfect candidate for DRYing.

// Dirty

import Title from './Title';

export const Thingie = ({ description }) => (

<div class="thingie">

<div class="description-wrapper">

<Description value={description} />

</div>

</div>

);

export const ThingieWithTitle = ({ title, description }) => (

<div>

<Title value={title} />

<div class="description-wrapper">

<Description value={description} />

</div>

</div>

);

Here we’ve allowed the passing of children to **Thingie**. We’ve then created **ThingieWithTitle** that wraps **Thingie**, passing in the **Title** as its children.

// Clean

import Title from './Title';

export const Thingie = ({ description, children }) => (

<div class="thingie">

{children}

<div class="description-wrapper">

<Description value={description} />

</div>

</div>

);

export const ThingieWithTitle = ({ title, ...others }) => (

<Thingie {...others}>

<Title value={title} />

</Thingie>

);

In conclusion…

I hope that I’ve helped you see the benefits of writing clean code and that you can even use some of the practical examples presented here. Once you embrace writing clean code, it will become second nature. You (and your future self) will soon appreciate the “write it and forget it” way of life.

Source- AmericanExpress.io