we will start with a bad code example, apply the first principle of SOLID, and see how this can help us to write small, beautiful, and clean React components with clear responsibilities.

Let’s get started.

**What Is the Single-Responsibility Principle?**

What the single-responsibility principle tells us is that **each class or component should have a single purpose of existence**.

**Components should do only one thing and do that well.**

Let’s refactor a bad but working piece of code and make it cleaner and better by using this principle.

**Let’s Start With a Bad Example**

Let’s first see some code that violates this principle. The comments are added for better understanding:

SingleresponsibilityViolation.js

**What this code does**

This is a functional component where we fetch data from a remote source, filter the data, and then show it in the UI. We also detect the loading state of the API call.

I have kept this example short for better understanding. But you can find these in the same component almost anywhere! There are a lot of things going on here:

1. Remote data fetching
2. Data filtering
3. Complex state management
4. Complex UI functionality

So let’s explore how we can improve the design of the code and make it tight.

**1. Move Data Processing Logic Out**

Never keep your HTTP calls inside the component. It’s a rule of thumb. There are several strategies you can follow to remove these codes from the component.

The least you should do is create a custom Hook and move your data-fetching logic there. For example, we can create a Hook named useGetRemoteData that looks like this:

useGetRemoteData.js

Now our main component will look like this:

FirstRefactor.js

Look how our component is nowsmaller and easier to understand! This is the simplest and first thing you can do in a convoluted codebase.

But we can do better.

**2. Reusable Hook for Data Fetching**

Now when we see our useGetRemoteData Hook, we see that this Hook is doing two things:

1. Fetching data from a remote source
2. Filtering data

Let's extract the logic of fetching remote data to a separate Hook named useHttpGetRequest that takes the URL as a component:

useHttpGetrequest.js

We also removed the reducer logic to a separate file:

LoadingReducer.js

So now our useGetRemoteDatabecomes:

**useGetRemoteData.js**

Much cleaner, right? Can we do better? Sure, why not?

**3. Decompose UI Components**

Take a look at our component where we show the details of a user. We can create a reusable UserDetailscomponent for that purpose:

UserDetails.js

Finally, our original component becomes:

Users.js

We slimmed our code down from 60 lines to 12 lines!And we created five separate components, each with a clear and single responsibility.

**Let's Review What We Just Did**

Let’s review our components and see if we achieved the SRP:

* Users.js— Responsible for displaying the user list
* UserDetails.js— Responsible for displaying details of a user
* useGetRemoteData.js— Responsible for filtering remote data
* useHttpGetrequest.js— Responsible for HTTP calls
* LoadingReducer.js**—**Complex state management

Of course, we can improve a lot of other things, but this should work as a good starting point for you.

**Conclusion**

This was a simple demonstration of how you can reduce the amount of code in each file and create beautiful and reusable components with the power of SOLID.