**What’s This Principle All About?**

According to [Stackify](https://stackify.com/interface-segregation-principle/" \t "_blank), the **Interface Segregation Principle** says

Clients should not be forced to depend upon interfaces that they do not use

In ReactJS **we don’t use any interface**, at least not in the sense of object-oriented programming. So the main takeaway for this scenario is:

*Components should not depend on things they don’t need.*

Now, let’s see how this principle can help us to write clean and beautiful ReactJS components.

**A Practical Approach**

Let’s say we have a **User** component that’s responsible for displaying the details of a user. Our user object looks something like this:

user.ts

However, it uses two children components, named PersonalDetails and BankingDetails to show the details.

Our User component looks something like this:

Similarly our PersonalDetails looks like this:

PersonalDetails.js

And our BankingDetails.js looks like this:

BankingDetails.js

So, what’s the problem with this approach? Well, two things.

Firstly, our PersonalDetails component doesn't need banking information and our BankingDetails component doesn't need personal details to function so it’s clearly violating the Interface Segregation Principle.

Secondly, In the future, if we want to add typescript to our project (which you should) then to test PersonalDetails you’ll be required to mock the whole user object, though the banking information has nothing to do with PersonalDetails.

So, we need to fix this. But how?

**Let’s Fix This**

There are several approaches for fixing this problem, but the underlying principle is the same.

We need to pass only the relevant information to the children components.

So we will break down our data object and pass the appropriate parts to the respective components only.

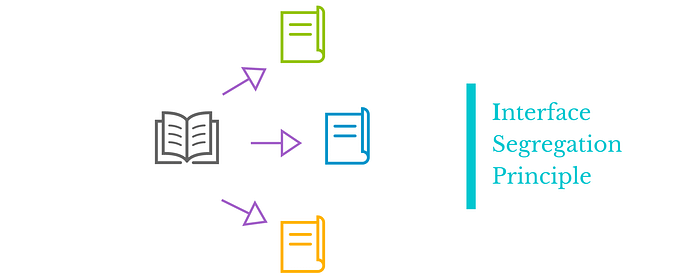


Image Credit: <https://medium.com/@learnstuff.io/interface-segregation-principle-dd885e59aec9>

Something like this:

User.js

We’ve broken our user data into two parts with new keys personalDetails and bankingDetails and we passed this specific piece of data to our child components.

**The Second Approach**

The previous solution is perfect, but what if you don’t have control over the data? Perhaps it’s fetched from a remote source. Or what if you don’t want to modify the data structure for some reason?

Don’t worry. We can apply another technique to solve this:

User.js

In our child components, we can use this as the following:

PersonalDetails.js

And:

BankingDetails.js

Now our child components only get what they need. They are cleaner and easier to understand.

Now our components don't depend on anything that they don’t need.

**Final Nail in The Coffin**

Up to this point, we introduced the concept of the **Interface Segregation Principle** to make our components tight. But in our previous [article](https://betterprogramming.pub/applying-the-liskov-substitution-principle-in-react-3a0614a42a08) on **Liskov Substitution Principle,**we saw that our current solution has a problem and that is anything can be passed onto the children component.

Let’s introduce typescript in our project and solve that issue.

Similarly, we can do the same to BankingDetails component.

Now we are forced to pass only the relevant data to the children components. **No more unwanted bugs for you!**