So, by now we got Components everywhere. We got the Expenses, ExpenseItem, ExpenseDate, App and others as well. React is all about Components and it's about props for configuring these Components for passing data into them. Generally, this approach of building a user interface from smaller building blocks is called composition.

Now there's one specifically interesting aspect of composition, which we haven't seen thus far. What if we wanted to create a Component, which actually just serves as a shell around any kind of other content. At the moment, we have highly specific Components. The ExpenseDate Component is about outputting a date. The ExpenseItem Component is just about outputting an Expense Item.

All these Components are also just configured through props, through the date prop, the title prop, the amount prop in the case of ExpenseItem. Sometimes, if you want to have a Component where you don't configure everything through props but instead you must be able to pass content between the opening and closing tags of that Component. And here's an example where this would make sense.

If we look at our current output, then we see two kinds of boxes or containers. We got a container around all the ExpenseItems, it has a light gray background and rounded corners, and a slight drop shadow which might be hard to see. And we have a container around the overall Expenses list, with a darker gray background but also rounded corners and a slight drop shadow.

Now, the idea behind all these Components is to have a reusable building blocks also to avoid code duplication. And here at the moment, we at least have some style duplication also may be some HTML structure duplication (because we have four ExpenseItem defined in Expenses.js rather than traversing them through loop).

Nonetheless, we could extract this surrounding container div which we have both in ExpenseItem and Expenses and extract the styles they have in common, like rounded corners and a drop shadow, into a separate Component. And we could name such a Component Card.js because this is a specific card look. And this is not a term I came up with instead in general web development when you hear the term card, it typically means some kind of container look with rounded corners, drop shadows and elements like those presents in our Component.

Now, Card Component is still a regular Component but such a Card Component could now do one main thing. It could return a div or any other kind of container HTML element with a className. From the Expenses.css we can cut border-radius and box-shadow properties. And, we can add those changes into the Card.css file. Similarly, border-radius and box-shadow properties also exists in ExpenseItem.css also we can simply remove those properties from there.

Now, why I am doing that? Just to show this kind of Component (Card.js). Now we got this Card container component, which should act as a shell around either our ExpenseItem content or our Expenses content.

The Card component will now not be configured through some attributes. Instead, my idea would be that in ExpenseItem, we can replace the built in div with our custom Card Component. And, then we get these predefined styles for the Card automatically. Maybe we then just want to make them extensible but we get these predefined styles automatically. For that we need to import Card.

If you wrap your ExpenseDate Component with Card Component you will see that all the ExpenseItem changes are lost and there would be no UI. And the reason for that is out of the box, you can't use your custom Components as wrappers around other kind of content. Having content between opening and closing tags doesn't work just like that.

But of course, it does work for built-in HTML elements, like the div or h2 tag. So, it would be nice if we make it work for our custom Components. To build such reusable wrapper Components like Card for example. In Card.js we again have to accept props. But now as I mentioned we'll not work with some attributes but instead we will use one special prop which is built into React, which every Component receives even if you're never setting it explicitly.

And that's a prop which value I would output between the opening and closing tag of the div (inside of the Card Component function) and it's a {props.children}. Children is a reserved name and the value of this special children prop will always be the content between the opening and closing tags of your custom Component. So, in our case the content between opening and closing tag is:



That is what will be available on props.children inside of that card. And therefore, if I now save this, we got some content back. However, it's also a bit broken and the reason for that is I extracted some styles which ExpenseItem and Expenses had in common, but I had more styles to find for ExpenseItems and these styles are important. But they're missing now.

I am setting a class name prop on card here but you must not forget that card is now a custom Component defined by you. All the default HTML Components out of the box supports className for adding CSS classes to the rendered HTML elements.

But your custom Components only support what you tell them to support. So if you want to make sure that a className can be set on your Card Component and then has an effect, we have to tweak the code in the Card Component.

And here we would probably wanna tweak it such that we add whatever is set as a className on Card to the className string, we're setting as a className on that div (Card.js). So, in Card.js we can add a classes constant which is card as a default class which is always applied + props.className.



So, anything we receive as a className from outside is added to that string. And in div we can dynamically point at this classes’ constant. With that we are now making sure that any value set on the className prop is added to the string of classNames which is finally set on the div inside of the Card.

If we now save that we get same look as before, not yet for Expenses but for the ExpenseItems but now we have the reusable wrapper Component. And we can also use that in Expenses now, instead of div we will have to use Card. And if we do that, now we get the rounded corners back on that Expenses container as well. Now, why would we do that? What did we gain by doing that?

Well, in this case, of course, not a whole lot but we were able to extract some code duplication from inside of our CSS files into this separate wrapper Component. And it's not just a duplicate CSS code, we also were able to extract the HTML code (below code):



In this case, it's just one div but throughout the course you will also see more complex wrapper Components which might have a more complex JSX structure. Things like modals and alerts. And in such cases, being able to extract that often allows you to save a lot of code duplication and it often allows you to keep your other Components clean. And this is another aspect of composition.

We compose our ExpenseItem Component by using Card as a wrapper, by using some built-in HTML elements, and by then also putting in the ExpenseDate. And all these Components and elements are composed together to form the overall ExpenseItem component, which then again, is used in other Components to build the overall user interface.

So, composition is important. You use it all the time when working with React. Whenever you combine Components, you are using composition. And an especially important part of composition is the {props.children} feature, which allows you to also create wrapper Components which is a special type of Component.