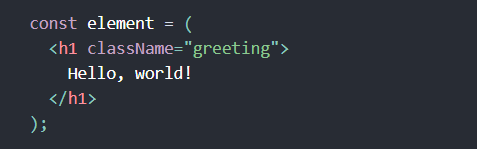
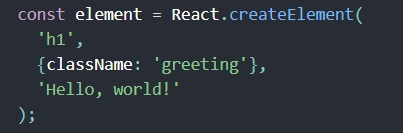
**Introducing JSX**

JSx produces React “element”.

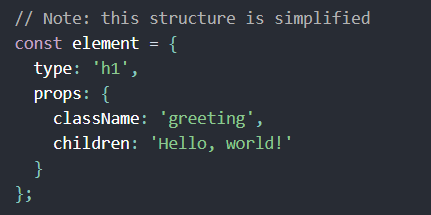
VIệc render logic thì được gắn kết với UI logic khác => thay vì chia các công nghệ bởi việc đặt markup và logic trong các files. React tạo ra components để chứa cả 2 (tức là cả JS và HTML).

JSX Represents Objects





React.createElement() perform a few checks to help you write bug-free code but essentially it creates an object like this:

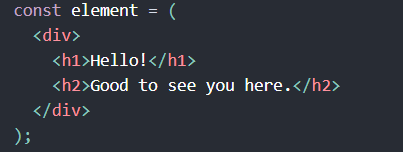


Specifying Children with JSX

If a tag is empty, you may close it immediately with />

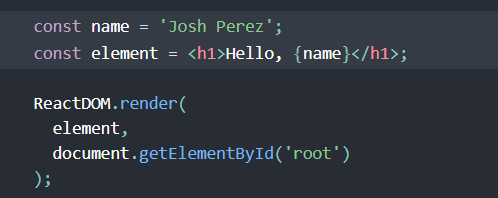


JSX tags may contain children:



Embedding Expressions in JSX

You can put any valid JS expression inside the curly braces in JSX





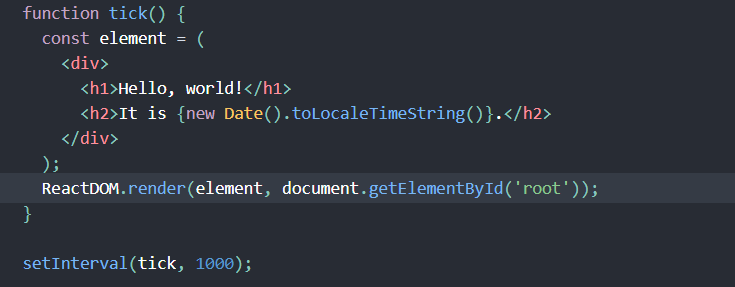
**Rendering Elements**

**An element is like a single frame in a movie: it represents the UI at a certain point in time.**

* **The only way to update the UI is to create a new element, and pass it to ReactDOM.render()**



**We call this a “root” DOM node becase everything inside it will be managed by React DOM**

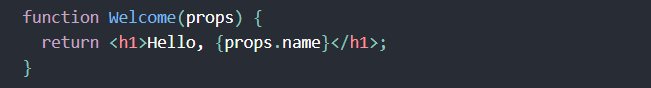


**It calls ReactDOM.render() every second from a setInterval() callback**

**Components, Props, State and Lifecycle**

**Function and Class Components**

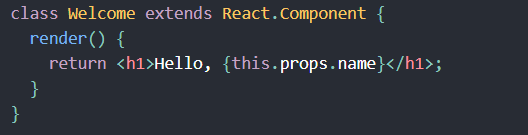
**The simplest way to define a component is to write a JS function:**



**We call such components “function components” because they are literally JS functions.**

**This function is accepts a single “props” object argument with data and returns a React element.**

**You can also use an ES6 class to define a component:**



**Rendering a Component**

**We encountered React elements that represent DOM tag (là thể hiện tag nguyên thủy):**

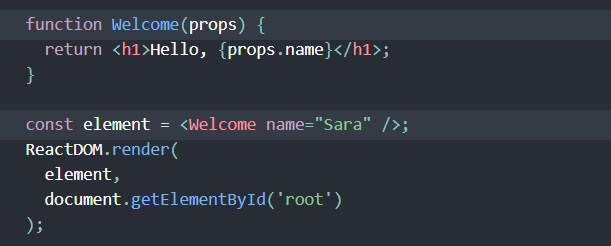


**Howerver, elements can also represent user-defined components:**



**When React see an element representing a user-defined component, it passes JSX attreibutes to this component as a single object. We call this object “props”**

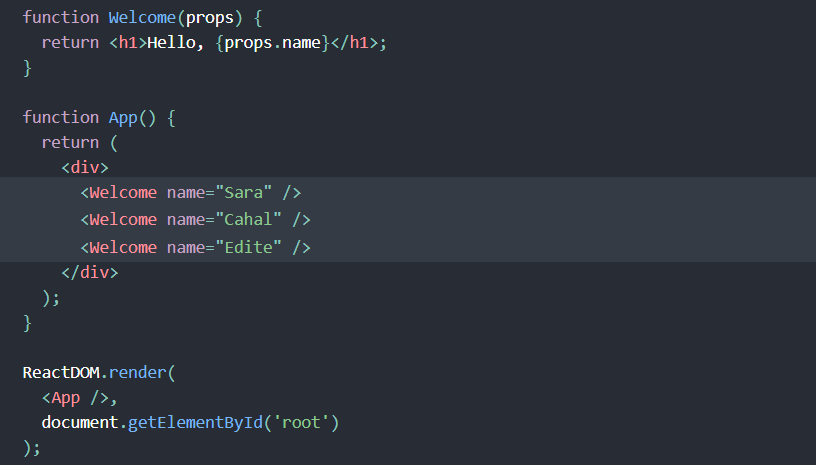
**For example**



1. **We call ReactDOM.render() with the <Welcome name=”Sara” /> element.**
2. **React calls the Welcome component with {name: ’Sara’} as the props**
3. **Our Welcome component returns a <h1>Hello, Sara</h1> element as the result**
4. **React DOM efficiently update the DOM to match <h1>Hello, Sara </h1>**

**Composing Components**

**Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. Abutton, a form, a dialog, a screen: in React apps, all thoes are commonly expressed as components.**



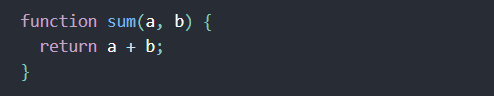
**Extracting Components**

**Don’t be afraid to split components into smaller components.**

**Khá giống với chúng ta cấu trúc hàm và class trong ngôn ngữ server side bình thường.**

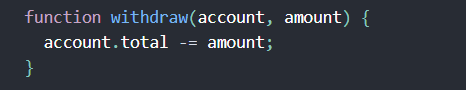
**Props are Read-Only**

**Consider this sum function**



**Such functions are called “pure” because they do not attempt to change their inputs, and always return the same result for the same inputs.**

**In constrast, this function is impure because it changes its own input:**



**React is pretty flexible but is has a single strict rule:**

**All React components must act like pure functions with respect to their props.**

**State and Lifecycle**

***State is similar to props, but it is private and fully controlled by the component.***

**Converting a Function to a Class**

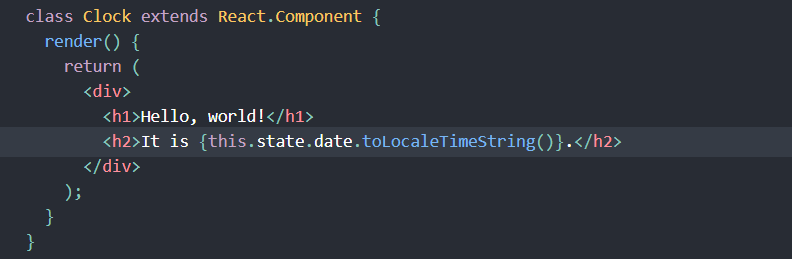
1. **Create an ES6 class, with the same name, that extends React.Component.**
2. **Add a single empty method to it called render()**
3. **Move the body of the function into the render() method**
4. **Replace props with this.props in the render() body**
5. **Delete the remaining empty function declaration.**



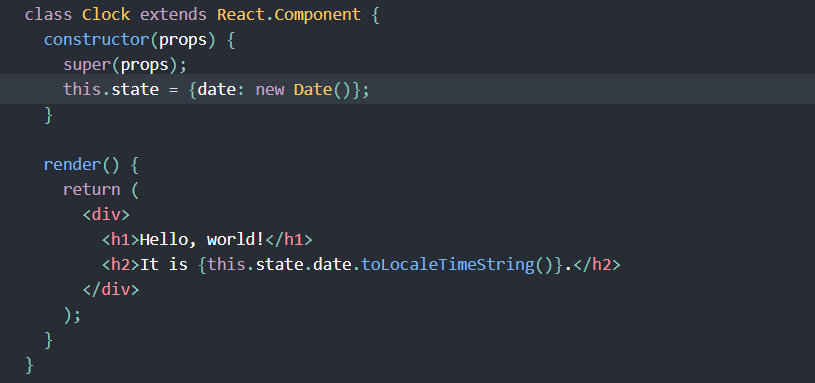
**Adding Local State to a Class**

**We will move the date from props to state in three steps:**

1. **Replace this.props.date with this.state.date in the render() method**

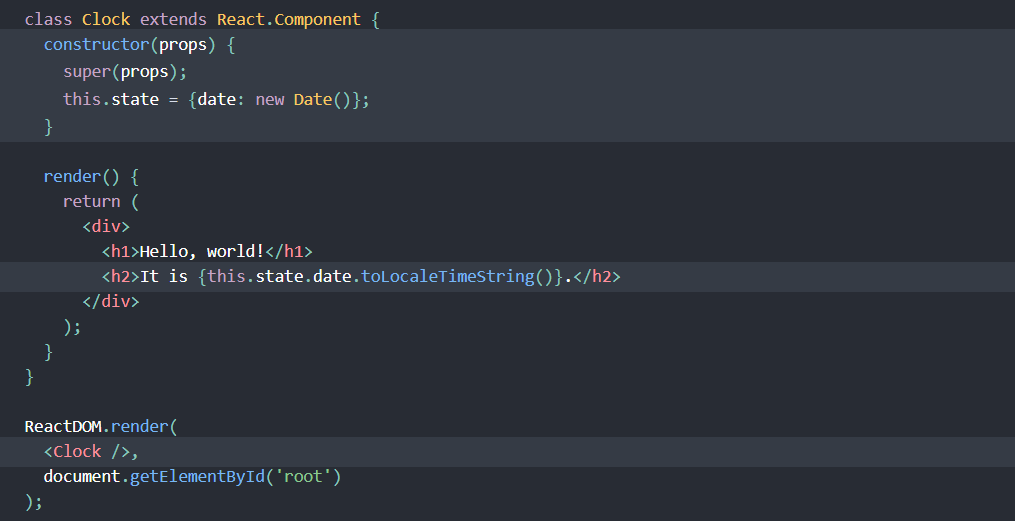


1. **Add a class constructor that assigns the initial this.state:**



1. **Remove the date prop from the <Clock /> element**

**The result looks like this:**



**Adding Lifecycle Methods to a Class**

**In applications with many components, it’s very important to free up resources taken by the components when they are destroyed.**

**We want to set up a timer whenever the Clock is rendered to the DOM for the first time. This is called “mounting” in React.**

**We also want to clear that timer whenever the DOM produced by the Clock is removed. This is called “unmounting” in React.**



**These methods are called “lifecycle methods”.**

**The componentDidMount() method runs after the component has been rendered to the DOM. This is a good place to set up timer:**

Let’s quickly recap what’s going on and the order in which the methods are called:

1. When <Clock /> is passed to ReactDOM.render(), React calls the constructor of the Clock component. Since Clock needs to display the current time, it initializes this.state with an object including the current time. We will later update this state.
2. React then calls the Clock component’s render() method. This is how React learns what should be displayed on the screen. React then updates the DOM to match the Clock’s render output.
3. When the Clock output is inserted in the DOM, React calls the componentDidMount() lifecycle method. Inside it, the Clock component asks the browser to set up a timer to call the component’s tick() method once a second.
4. Every second the browser calls the tick() method. Inside it, the Clock component schedules a UI update by calling setState() with an object containing the current time. Thanks to the setState() call, React knows the state has changed, and calls the render() method again to learn what should be on the screen. This time, this.state.date in the render() method will be different, and so the render output will include the updated time. React updates the DOM accordingly.
5. If the Clock component is ever removed from the DOM, React calls the componentWillUnmount() lifecycle method so the timer is stopped.

**The Data Flows Down**

**Props thì tương tự như property. Còn state thì giống như là attribute**

**Neither parent nor child components can know if a certain component is stateful of stateless, and they shoundn’t care whether it is defined as a dunction or a class.**

**Acomponent may choose to pass its state down as props to its child components**

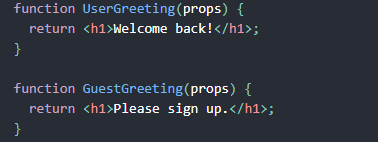


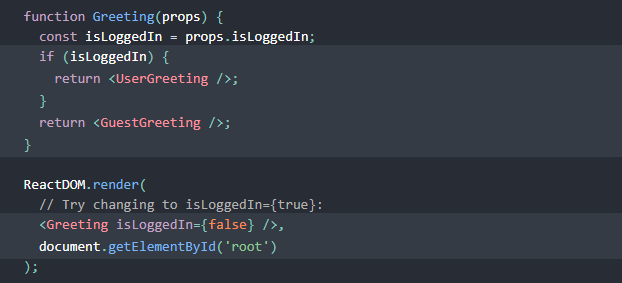
**Handling Events**

* **React events are named using camelCase, rather than lowercase (sử dụng camelCase không dùng lowercase)**
* **With JSX you pass a function as the event handler, rather than a string (sử dụng event handler thay vì string)**

**Conditional Rendering**

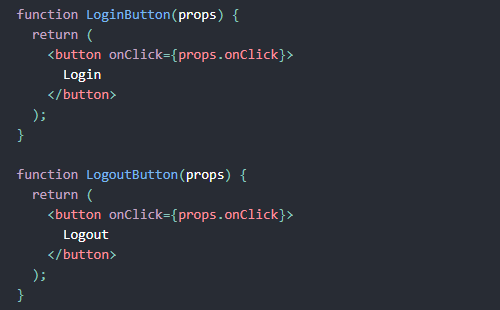
**Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state and let React update the UI to match them.**

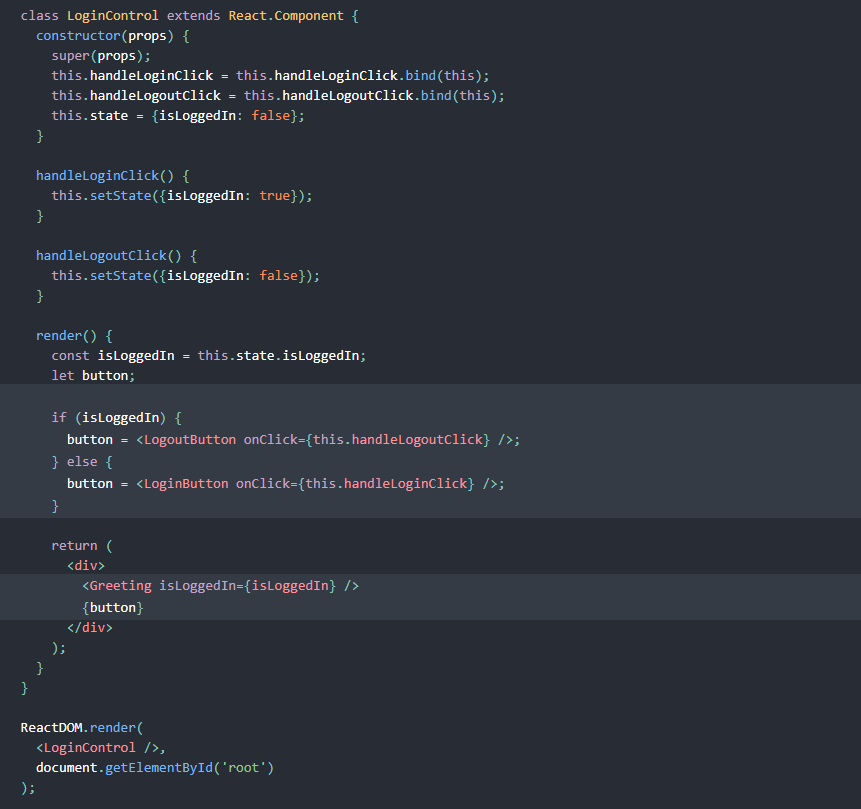




**Element Variables**

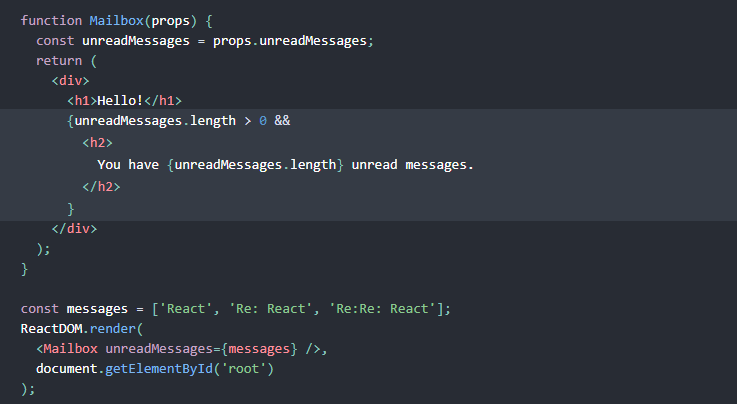
**You can use variables to store elements. This can help you conditionally render a part of the component while the rest of the output doesn’t change**





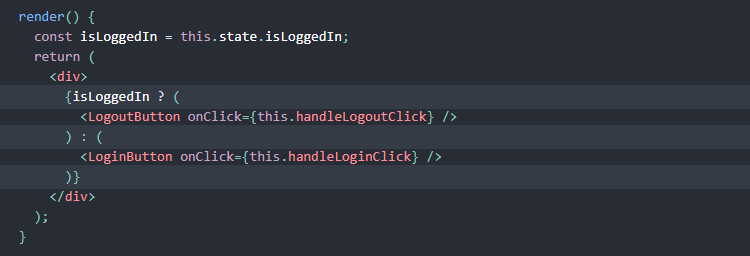
**Inline If with Logical && Operator**

**You may embed any expressions in JSX by warpping them in curly braces. This includes the JavaScript logical && operator. It can be handy for conditionally including an element.**



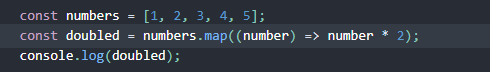
**Inline If-Else with Conditional Operator**

**Conditional ? true : false**

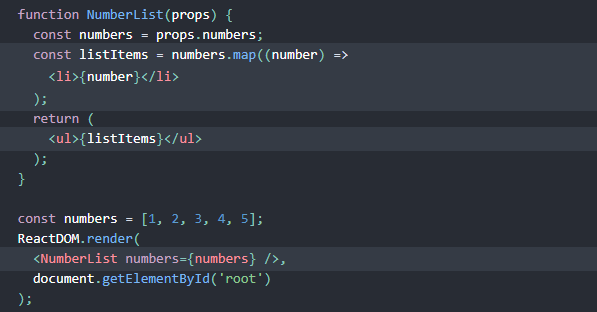


**List and Keys**

**We use the map() function to take an array.**



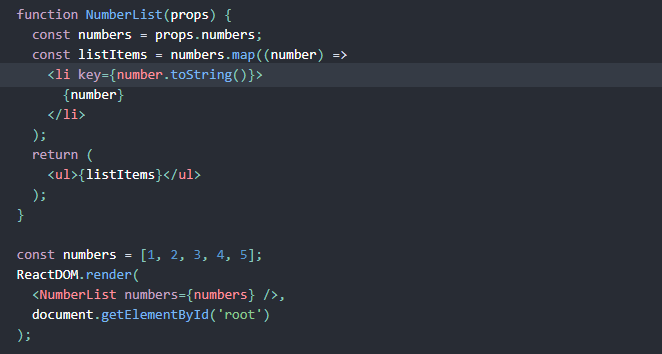
**You can build collections of elements and include them in JSX using curly braces {}**



**When you run this code, you’ll be given a warning that a key should be provided for list items.**

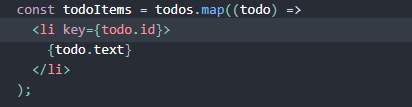
**A “key” is a special string attribute you need to include when creating lists of elements.**

**Let’s assign a key ti our list items inside number.map() and fix missing key issue**

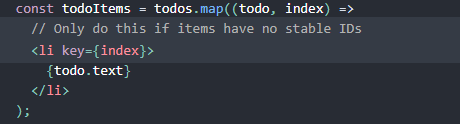


**Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity.**

**The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys**



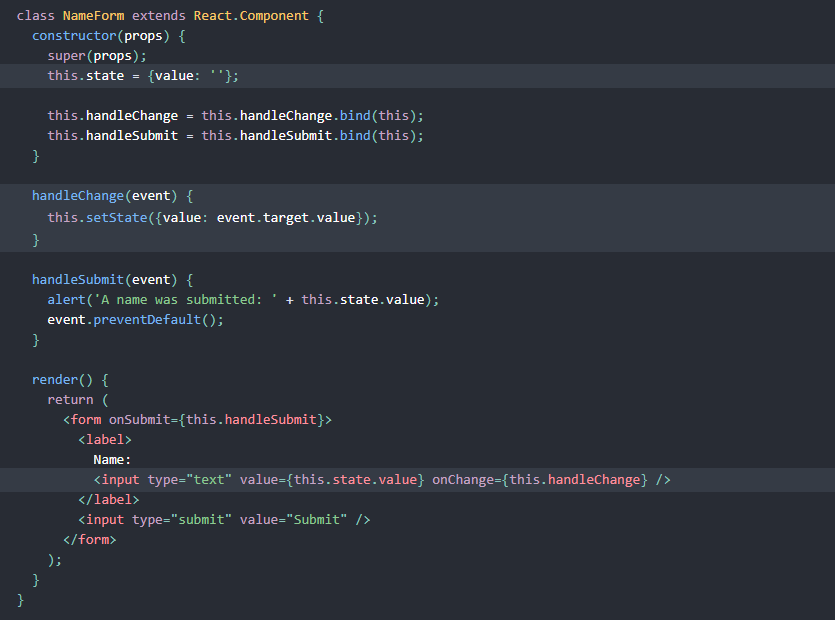
**Or you may use the item index as a key**



**We don’t recommend using indexed for keys if the order of items may change**

**Keys Must Only Be Unique among Siblings. However they don’t nedd to be globally unique. We can use the same keys when we produce two different arrays**

**Forms**



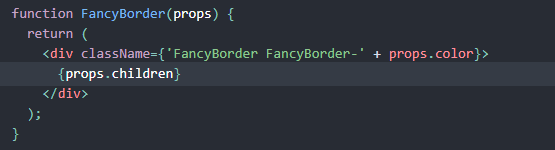
**Since the value attribute í set on our form element, the displayed value will always be thí.state.value, making the React state the source of truth. Since handleChange runs on every keystroke to update the React state, the displayed value will update as the user types.**

**Composition vs Inheritance**

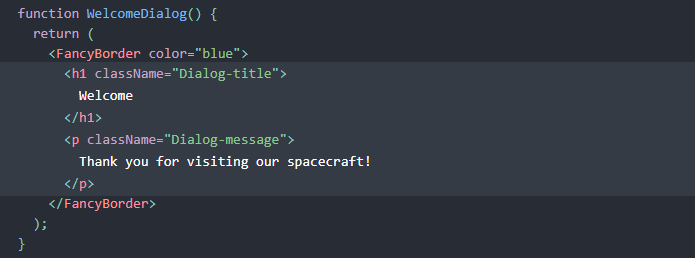
**React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.**

**Containment (nói về các child giữa thẻ của các thằng cha)**

**Component-Contaner use the special children prop to pass children elements directly into their output:**



**This lets other components pass arbitary children to them by nesting the JSX:**



**Anything inside the <FancyBorder> JSX tag gets passed into the FancyBorder component as a children prop. Since FancyBorder renders {props.children} inside a <div>, the passed elemens appear in the final output.**

**React elements like <Contacts /> and <Chat /> are just objects, so you can pass them as props like any other data. This approach may remind you of “slots” in other libraries but there are no limitations on what you can pass as props in React.**

**Thinking in React**

1. **Break The UI Into A Component Hierarchy**

**The first thing you’ll want to do is to draw boxes around component in the mock and give them all names. Use single principle for component.**

**Since you’re often displaying a JSON data model to a user, you’ll find that if your model was built correctly, your UI will map nicely. That’s because UI and data models tend to adhere to the same infomation architechture. Separate your UI into components, where each component matches one piece of your data model.**



**You’ll see here that we have five components in our app. We’ve italicized the data each component represents.**

1. **FilterableProductTable (orange): contains the entirety of example**
2. **SearchBar (blue): receives all user input**
3. **ProductTable (green): display and filters the data collection based on user input**
4. **ProductCategoryRow (turquoise): display a heading for each category**
5. **ProductRow (red): display a row for each product**

**Now that we’ve identified the components in our mock, let’s arrange them into a hierachy. Components that appear within another component in the mock should appear as a child in the hierarchy:**

* **FilterableProductTable**
* **SearchBar**
* **ProductTable**
* **ProductCategoryRow**
* **ProductRow**

1. **Build A Static Version in React**

**We have a component hierarchy, it’s tiem to implement our app. The easiest way is to build a version that takes your data model and renders the UI but has no interactivity. It’s best to decouple these process because building a static version requires a lot of typing and no thinking, and adding interactivity requires a lot of thinking and not a lot of typing. We’ll see why.**

**You can build top-down or bottom-up. It simpler examples, it’s usually easier to go top-down, and on larger projects, it’s easier to go bottom-up and write tests as you child.. React’s one-way data flow (alseo called one-way binding) keeps everything modular and fast.**

1. **Identify The Minimal (but complete) Representation Of UI State**

**To make your UI interactive, you need to be able to trigger changes to your underlying data model. React achieves this with state.**

**To build your app correctly, you first need to think of the minimal set of mutable state that your app needs.**

**Let’s go through each one and figure out which one is state. Ask three questions about each piece of data.**

1. **Is it passed in from a parent via props? If so, it probably isn’t state.**
2. **Does it remain unchanged over time? If so, it probably isn’t state**
3. **Can you compute it based on any other state or props in your component? If so, it isn’t state.**

**The original list of products is passed in as props, so that’s not state. The search text and the checkbox seem to be state since they change over time and can’t be computed from anything. And finally, the filtered list of products isn’t state because it can be computed by combining the original list of products with the search text and value of the checkbox.**

**So finnaly, our state is:**

* **The search text the user has entered**
* **The value of the checkbox**

1. **Identify Where Your State Should Live**

**Ok, so we’ve identified what the minimal set of app state is. Next, we need to identify which component mutables, or owns, this state.**

**Remember: React is all about one-way data flow down the component hierarchy.**

**For each piece of state in your application:**

* + **Identify every component that renders something based on that state.**
  + **Find a common owner component (a single component above all the components that need the state in the hierarchy)**
  + **Either the common owner or another component higher up in the hierarchy should own the state.**

**Accessibility**