# **Dynamic events and how to handle them**

## Types of events

Recall that events are the process by which JavaScript interacts with HTML and can occur when the user or the browser manipulates a page. They provide enhanced interactive experiences, such as responding to mouse clicks, movements, or keyboard commands. Because events usually rely on some interaction, they need to wait and listen in the background for that interaction to occur before they can be triggered.

Developers can use events to execute JavaScript's code in response to an action based on user interactivity, like clicking a button. This process by which the HTML button communicates to the JavaScript event handler to execute some code and respond to the event action is known as triggering.

In React code, events are handled using JSX event attributes, which are very similar to HTML event attributes that you may be familiar with. For example, the click handling attributes in HTML is the onclick attribute with all the letters lowercased. The equivalent click handling attributes in React's JSX is the camelCased onClick attribute.

*Choose a React event-handling attribute.*

* *onClick*

## Common event handling

*Graphical user interface, text

Description automatically generated*

*Event handlers make sure that the actions of events are executed. To set up the click handling behavior for a button, you can use an expression named clickHandler.*

* *False*

## Syntax for handlers

Text

Description automatically generated

Going back to React. The biggest difference in syntax involves the use of the addEventListener method. In React. The rule is to avoid manipulating the DOM directly as much as possible. You should set everything up declaratively, meaning that you describe updates to React and let it figure out the rest. This is best done using event attribute, fortunately, one-to-one mapping between HTML event attributes and JSX event attributes means it's easier to learn. Event handling in React is overall quite similar to HTML.

But note that there is no function invocation syntax in event handling attribute in React. In other words, while in plain JavaScript, you would need to pass an invocation to an event handling function as a value to the on click events.

In React, you should not invoke a function. Instead, you just pass a reference to the event handling function without invoking it.

Chart

Description automatically generated with medium confidence

*True or false? You can use the onclick event-handling HTML attribute to handle click events.*

* *True*

## User events

Text

Description automatically generated

*In React, a click handler is placed inside a JSX expression, and only needs a click handler function's name - without the parentheses to invoke it.*

* *True*

# **Data and Events**

## Parent-child data flow

Graphical user interface, text, application

Description automatically generated

*What are the advantages of utilizing a centralized point of data - a "single source of truth" - in your React apps? Choose all that apply.*

* *It allows you to edit multiple items from a single point*
* *It reduces the possibility of typing errors in your code*
* *It offers a more efficient way of working when data frequently changes*

## Children and data

Why is one-way flow in React important? Let me tell you. This type of data flow ensures that the data is moving from top to bottom through the component hierarchy. It also ensures that changes are transmitted through the system.

In React, data is passed down from parent components to a child components via props. A child components can't mutate or change its props. It can only read them and re-render. This means that the data comes from the parents and it's just consumed in the child components.

You've learned about passing data to a child component using props. However, there's another way to work with data in React components, and that data is referred to as state. All the data in React can be divided into props data and states data. Props data is data outside the components that it receives and works with but cannot mutate. State data is data inside the components that it controls and can mutate. It also helps to think of it like this. The prop data belongs to the parent that renders the components. The state data belongs to the component itself.

*Which of the following statements about data flow in React are correct? Select all that apply.*

* *The props data is data outside the component and cannot mutate.*
* *State data is data inside the component, and the component can control and mutate the data.*

## What are hooks?

Keeping track of state across components can become quite a task and this is where React's hooks can help. One key benefit of hooks is that they solve the problem of unnecessary code reuse across components.

Hooks are functions that were introduced in React version 16.8. They let you hook into React state and lifestyle features from components.

Text, letter

Description automatically generated

You can use the useState hook to track any type of data. It could be strings, numbers, arrays, Booleans or objects. For instance, you can even track the number of times a button is pressed. In addition to the hooks that come out of the box with React, you can also build your own hooks, which will let you extract custom component logic into reusable functions.

*Which of these rules apply to hooks in React? Check all that apply.*

* *Hooks can be called only at the top level.*
* *Hooks can be called only from React functions.*

## Using Hooks

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

## What is state?

It helps to think of State as a component's internal data that determines the current behaviour of a component. It's often used to store data that affects the behaviour of a component. States is important because it allows components to stay in sync with each other and ensure that your app behaves as intended, for example, if one component updates its State, all other components that depend on that State will automatically update too.

This means that a component sends its State to its children by using props. If the child components have their own grandchild components, then the child components might have some States that they send as props to those grandchild components.

In React, State is kept in a state of variables. The main way to change State is to alter these variables. When a component is created, it gets an initial State. The State is used to initialize the component's properties. Components can be either stateful or stateless.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

*A parent component is able to pass its state onto children components. True or false?*

* *True*

## Observing state

Text

Description automatically generated

The useState Hook allows a components to define and track state. It does this with two arguments. The first of which accesses state and the second of which updates it with a function.

Text

Description automatically generated

If you call the setWord hook on its own it will not work and will result in error, you must call it from a react function such as the example above it, its called from a handleClick function which is then called from the buttons onClick attribute.

Text

Description automatically generated

The above setWord hook is called directly and will not work and results in an error.

*Consider the following line of code, which contains a useState hook:*

*const [date, setDate] = React.useState(new Date());*

*Which argument updates the state?*

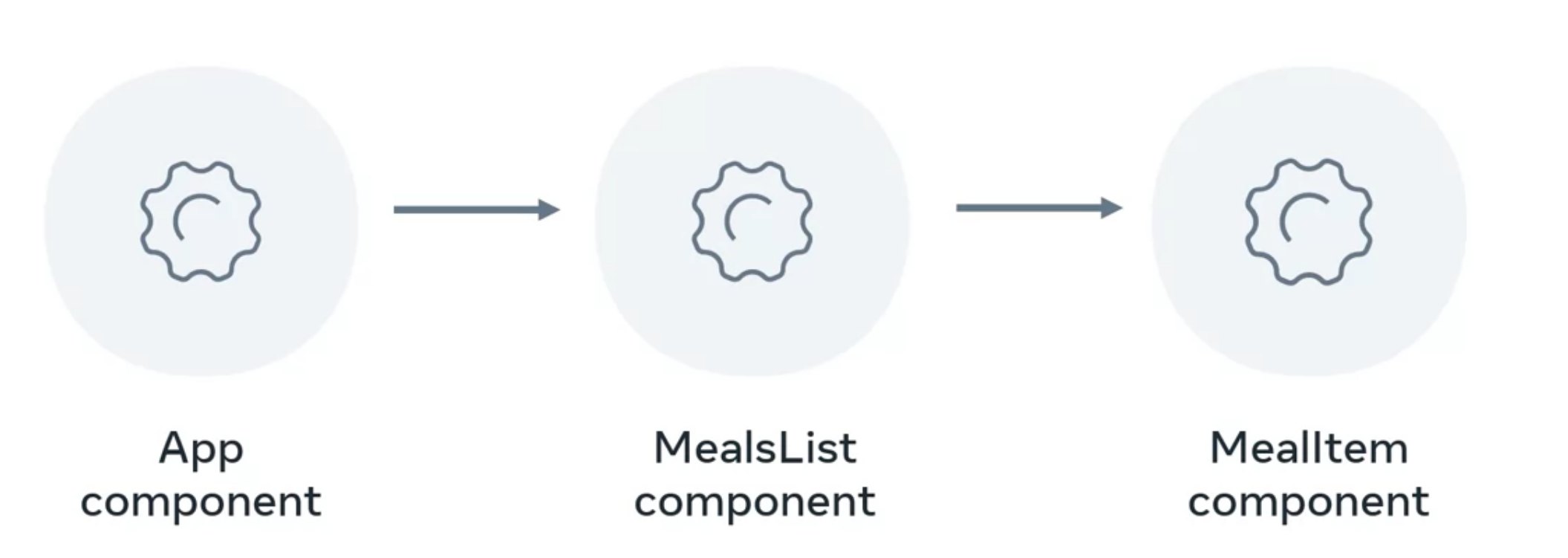
* *setDate*

## Managing state

Prop drilling is a commonly used term to describe having to pass state free props in several layers of components, from the parent to the child, to the grandchild, and so on.

A picture containing diagram

Description automatically generated



You move the state up from the meal list component to the app component so that it can be passed down as props.

A picture containing graphical user interface

Description automatically generated

There's another way to phrase this problem through the viewpoint of global state. Whenever I have states that might have to be used in various places in my app, that's a global state issue. An elegant solution for this issue is Reacts Context API. One way to think about the context API is that it cuts out the middleman. There's no need for prop drilling and lifting state up. Instead, the component that needs the data simply gets it from the context API. The way that this is achieved is by extracting the state into a separate file that holds the state in context, then, any file that needs it simply imports it and uses it.

*Lifting state up is coding your app so that the state from the child component is moved to the parent component and the child component simply receives it via props.*

* *True (Lifting state up is about cutting the state from the child component and moving it to the parent component's code, with the intent of making the state available in sibling components.)*

## React state management

A picture containing text, clock, watch, gauge

Description automatically generated

While passing props helps to manage states, it is like taking a bus and going through each stop before you get off at the end. In comparison, using the context API is like teleporting to your destination instantly. It's a way to bypass the redundant passing of data through multiple levels of components. To set it up, you need to add a piece of code that will be your context provider. It's also where the state will be stored. When a component needs to use the state, it becomes a context consumer.

Text

Description automatically generated

*What's a context consumer?*

* *It's the component that uses the context provider's state.*

## useContext (medium blog)

<https://medium.com/geekculture/react-hooks-usecontext-e8c468bce91a>

## Stateful vs. stateless

The distinction between stateful and stateless components is that a stateful component holds states as internal data and its state changes based on the way that the app is built; often as a result of user actions.

A stateless component, however, doesn't store states and any changes must be inherited through props.

The distinction between stateful and stateless components is that a stateful component holds states as internal data and its state changes based on the way that the app is built; often as a result of user actions.

A stateless component, however, doesn't store states and any changes must be inherited through props.

Diagram

Description automatically generated

A common approach for organizing components in React is to have a stateful component as the parent which then sends its states down to several stateless components that then receive the state and render it on the screen. The children components are stateless because they don't have their own state and only receive their parent state when passed down by using props.

*You are creating a page for your app consisting of two components:*

*Parent: displays different information based on whether or not user has logged in.*

*Child: contains an event handler that sends data to Parent when ‘Log in’ button is clicked.*

*In this scenario, would it be better to use a stateful or stateless parent component?*

* *Stateful (The parent component needs to maintain either a 'logged in' or 'logged out' state in order to work as intended, so it should be stateful.)*