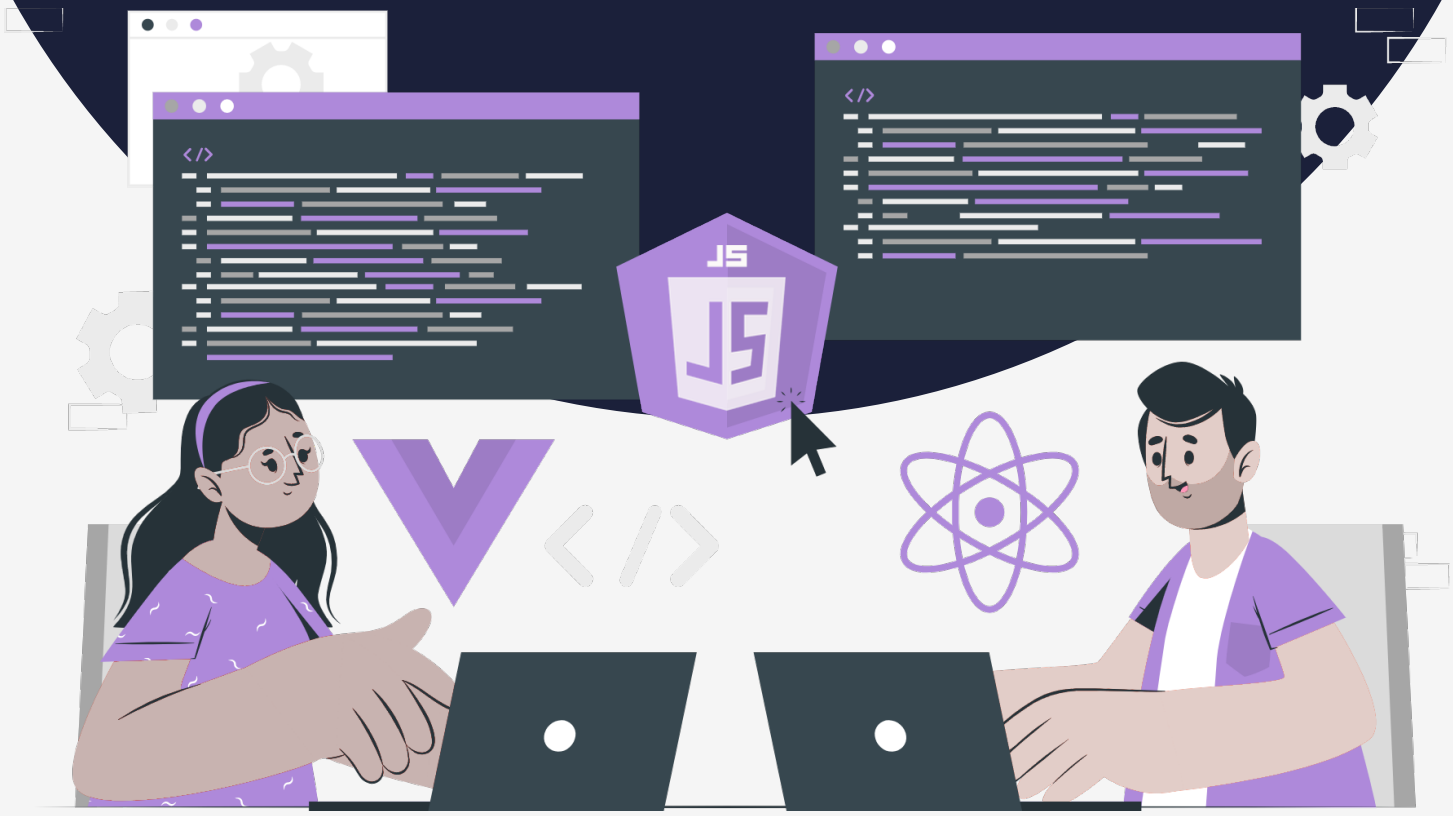
Lesson:

Props and its uses



PW SKILLS

Topics Covered:

1. What are props?
2. Destructuring of props.
3. Default props.
4. Advantages Of props.
5. Why are props read only?
6. How to pass props from grandparent to grandchild?

What are props?
In React, "props" is an abbreviation for "properties". Props are a mechanism for passing data from a parent component to a child component.

In React, components are the building blocks of user interfaces, and they are designed to be modular and reusable. Props are a way to customize a component and make it more flexible. When a parent component passes props to a child component, the child component can use those props to render content or behaviour in a way that is specific to the data it receives.

Inside the components, we can add attributes called props. We can't modify the props inside the component so it is immutable.

But the important thing is that data with props are being passed in unidirectional data flow i.e one way from parent to child.
Data coming from the parent should not be changed by child components.

```javascript
//parent component

function App(){

  return(
      <div>
         <UserDetails name ='PWSKILL' bio="Advanced Skill" />
      </div>
 )

}

//child component

function UserDetails(props){
return (
    <div>
      <h1>{props.name} </h1>
      <h1>{props.bio} </h1>
    </div>
)
}

export default App;
```

In this example,the parent component(App) passes the “PWSKILL” value as name and “Advanced Skill” as bio to the userDetails component. The UserDetails component then uses name and bio props to display PWSKILL and Advanced Skill.

The child component can use the props passed by the parent component but cannot change the value of the prop. This ensures that the state of the parent component is not affected by the child component and that any changes made to the state of the child component are isolated and do not affect the parent component.

Another example:

We have one component named UserInfo and here we want to display the user's name and age. In the App.js we pass the `User name` and `age` as props.

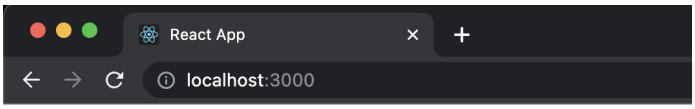
UserInfo.js

```
JavaScript
import React from 'react`

const UserInfo = (props)=>{
  return (
      <div>
         <h1>User Details </h1>
         <h1>{props.name} </h1>
         <h1> {props.age} </h1>
      </div>

)


 }

 export default UserInfo;
```

Import the UserInfo component into the App.js.

```javascript
import UserInfo from "./components/UserInfo";

     function App() {
          return (
              <div>
                  <UserInfo name="PW" age={29} />

              </div>

)

}

export default App;
```

Output:



User Details

PW

29

For example with destructuring:

Destructuring is a fundamental attribute that is employed in React, primarily when passing props, to make the code much clearer and more readable.

```javascript
import UserInfo from "./components/UserInfo";

       function App() {
            return (
               <div>
                  <UserInfo name="PW" age={28} />
               </div>
         )



  }
 export default App;
```

UserInfo.js:

```javascript
import React from "react";

const UserInfo=({name, age})=>{

     return (
        <div>
          <h1>User Details </h1>
          <h1>{name}</h1>
          <h1>{age} </h1>

        </div>

   )

}

export default UserInfo;
```

In this example, we can destructure the name and age props and use them directly.

Default props:

When a component's props are not provided, default props in React allow you to provide default values for those props. When a prop is optional and you need to supply a default value in case it is not passed, this can be helpful.

To set default props for a functional component, you can use the defaultProps property on the component. For example, if a component has a prop called name that is optional, you can set a default value like this:

Let's take an example of how default props works:

```javascript
import React from "react"

     function App(props){

          return (
             <div>
                <h1>{props.name} </h1>
             </div>
        );
    }

  App.defaultProps = {
        name: "PW SKILL",
        };

  export default App;
```

In this example, if the name prop is not provided when the component is rendered, it will default to the value 'Default Name'.

It's worth noting that default props are only used if the prop is undefined, if the prop is passed with a falsy value like false, 0, '' etc, the default prop will not be used.

Advantages:

- Destructuring props in React allows for more concise and readable code by directly extracting the specific props that a component needs, rather than referencing the entire props object.
- This can also make it easier to understand what props a component is using and can prevent bugs caused by wrong value of props
- Additionally, destructuring can also make it easier to assign default values to props, and can improve performance by reducing the number of times the props object is accessed.

props are used to pass information from parent component to child component. props are considered to be read-only because they are passed down from the parent component and can not be modified by the child component.

React follows one way data flow which means that data should flow in a single direction from parent component to child component. This makes it easier to understand and manage the state of the application in a large application. If a child component were able to modify props passed down from its parent, it would introduce the possibility of unexpected side effects and make it harder to understand how the data is being used and modified.

By keeping props read-only, React ensures that the parent component has full control over the data and can make any necessary changes, while the child component can only read and display the data. This makes it easier to understand and manage the state of the application and ensures that the data remains consistent and predictable.

```javascript
import React from 'react';

// Parent component
function ParentComponent(props) {
  return (
    <ChildComponent name={props.name} age={props.age}/>
  );
}

// Child component
function ChildComponent(props) {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}

// Render the parent component

ReactDOM.render(
  <ParentComponent name="PW SKILL" age={30} />,
  document.getElementById('root')
);
```

In this example, the `ParentComponent` renders the `ChildComponent` and passes it two props: `name` and `age`. The child component

receives these props as arguments and displays them in a paragraph element.

As you can see, the `ChildComponent` is only reading the data passed down from the parent component and displaying it, but it can't modify it.

Let's take an example of passing props from grandparent to grandchild:

Prop passing from grandparent to grandchild component in React is the process of passing data from a component higher up in the component tree to a component lower down in the tree, through multiple levels of intermediate components.

Here is an example of how prop passing from grandparent component to a grandchild component:

The grandparent component `App`, renders the parent component, `Userboard`, and passes a prop called `user` to it:

```
JavaScript
function App() {
   const user = {name:"PW SKILL"};

return (
    <Userboard user={user} />
     );

}
```

The parent component, `Userboard` receives the `user` prop and renders the grandchild component, `Details`, passing the `user` prop to it as well:

```
JavaScript
function Userboard(props) {
  return (
    <Details user={props.user} />
  );
}
```

The grandchild component `Details` receives the `user` prop and can access its value:

```javascript
function Details(props) {
  console.log(props.user.name);  // Output: "PW SKILL"
  // ...
}
```

In this example, the `App` component is the grandparent component, the `Userboard` component is the parent component, and the `Details` component is the grandchild component. The App component passes the user prop to the `Userboard` component, which in turn passes it to the `Details` component. This allows the `Details` component to access the data that was passed to it by the grandparent component, even though the two components are not directly connected in the component tree.

Here's another example:

Let's say we have a grandparent component called `**App**`, a parent component called `**Page**`, and a grandchild component called `**Comment**`. The `App` component wants to pass a prop called post to the `Comment` component. Here's how it can be done:

The `App` component renders the `Page` component and passes the post prop to it:

```javascript
function App() {
  const post = { id: 1, title: 'PW SKILL', comments:
[{...},{...},{...}] };
  return (
    <Page post={post} />
  );
}
```

The `Page` component receives the post prop and maps through the post.comments array and renders the `Comment` component, passing the comment object to it:

```javascript
function Page(props) {
  return (
    <div>
        <h1>{props.post.title}</h1>
        {props.post.comments.map(comment => (
          <Comment comment={comment} key={comment.id}/>
        ))}
    </div>
  );
}
```

Note:

- In React, a "key" is used to identify each item in a list of elements. This is necessary because when React renders a list, it updates the DOM by removing and re-adding elements as necessary.
- If a key is not provided, React uses the index of the element in the list to identify it. However, if the list is re-ordered or elements are added or removed, the indexes will change and React will not be able to keep track of which element is which.
- By providing a unique key for each element, React can efficiently update the list and maintain the correct order and state of the elements.

The `Comment` component receives the `comment` prop and can access its value:

```javascript
function Comment(props) {
  return <div>{props.comment.text}</div>
}
```

- In this example, the `App` component is the grandparent component, the `Page` component is the parent component, and the `Comment` component is the grandchild component.
- The `App` component passes the post prop to the `Page` component, which in turn maps through the post.comments array and passes each comment object to the `Comment ` component.
- This allows the `Comment` component to access the specific comment data that was passed to it by the parent component, even though the two components are not directly connected in the component tree.

What is the State?

In React, the term "state" refers to the current condition or data of a component that can be updated and manipulated over time. State is a fundamental concept in React as it enables components to manage and update their own data dynamically and respond to user interactions.

In React, both state and props are used to manage data within a component, but they have different roles and use cases.

- State refers to the internal data that a component maintains, and can be changed over time through user interactions or other events. State is typically used for data that affects the component's behaviour, rendering, or interaction with other components.
- Props, on the other hand, are short for properties, and they are data that are passed into a component from its parent

component. Props are read-only, meaning that a component cannot modify its props directly. Props are used to customize a component's behaviour or appearance based on the data passed from its parent component.

In summary, state is used for data that is managed internally by a component and can change over time, while props are used for data passed down from a parent component to its child component