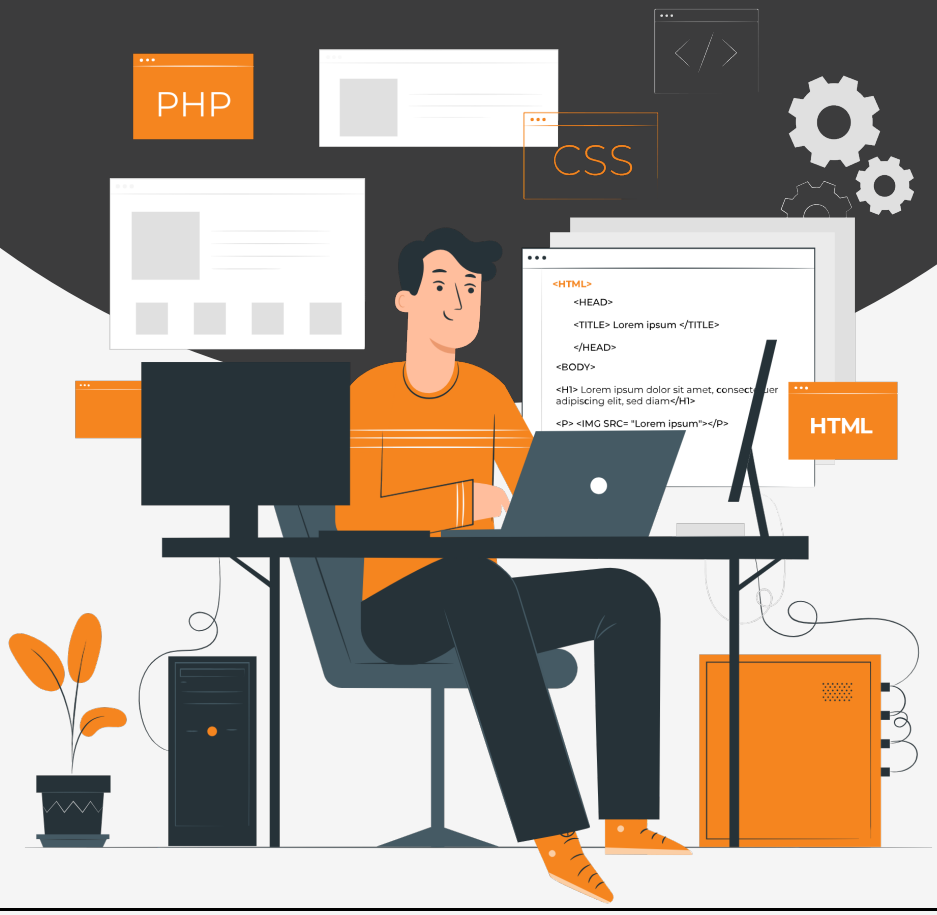
Lesson:

Components and Props: Building Blocks of React



PW SKILLS

Topics to be covered

1. What are Components?
2. Component Example: Car
3. Using Components with Props
4. Real-World Analogy: Components as Lego Bricks
5. Common Pitfalls
6. Conclusion

In React, components are the building blocks of your user interface. They allow you to encapsulate reusable and self-contained pieces of UI logic and functionality. With components, you can create modular and scalable applications. Let's explore components and how they interact with props.

What are Components?

Think of components as individual parts of a larger machine. Each component represents a specific functionality or piece of content that can be reused throughout your application. Just as a car is made up of various components like the engine, wheels, and seats, a React application is composed of different components working together to create the user interface.

Components can be classified into two types: **class components** and **functional components**. Class components are created using JavaScript classes, while functional components are defined as functions.

Component Example: Car

Imagine you're building an app that displays information about cars. You can create a **Car** component to represent each car and its details. Here's an example of a functional component:

```
import React from 'react';
import { View, Text } from 'react-native';

const Car = (props) => {
  return (
    <View >
      <Text>{props.make}</Text>
      <Text>Model: {props.model}</Text>
      <Text>Year: {props.year}</Text>
    </View>
  );
};

export default Car;
```

In the above example, the Car component receives props as an argument. Props are a way to pass data from a parent component to a child component. Here, props contain the car's make, model, and year, which are displayed within the JSX tags.

Using Components with Props

Props allow you to customize and configure components, similar to how you can customize a car with different features and options. You can pass any data, including primitive values, objects, or even functions, as props to a component.

Let's create an instance of the **Car** component and pass it some props:

```
import React from 'react';
import { View, Text } from 'react-native';
import Car from './Car';

const App = () => {
  return (
    <View>
      <Text>My Cars</Text>
      <Car make="Tesla" model="Model S" year={2022} />
      <Car make="BMW" model="X5" year={2021} />
    </View>
  );
};

export default App;
```

In the App component, we render two instances of the Car component, each with different props. These props are accessed within the Car component using the props object, allowing us to display specific car details.

Real-World Analogy: Components as Lego Bricks

To better understand components, imagine building a complex structure with Lego bricks. Each Lego brick represents a component. You can stack bricks together to create different structures, just like composing components to build an application.

Components act as self-contained units, making it easy to reuse and rearrange them to construct various parts of your UI. This modular approach simplifies development and maintenance, enabling you to manage and update individual components without affecting the entire application.

Common Pitfalls

While working with components and props, it's essential to be aware of common pitfalls:

1. Forgetting to Use Props
Make sure to pass the necessary props when using a component. Forgetting to provide required props can lead to errors or unexpected behavior.

2. Mutating Props Directly
Props should be treated as read-only within a component. Modifying props directly can introduce bugs and lead to unpredictable results. Instead, create local variables or use state to manage component-specific data.

3. Overusing or Underusing Components
Finding the right balance is crucial. Over

using components can lead to a complex and hard-to-maintain codebase, while underusing components can result in duplicated code and reduced reusability. Strive for a logical and modular component structure.

Conclusion

Components and props are fundamental concepts in React. Components represent individual UI elements, while props allow you to customize and configure these components. By combining and reusing components, you can build complex and interactive user interfaces.